UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

Analysis and Coding of Visual Objects: New Concepts and New Tools

Manuel Pinto da Silva Menezes de Sequeira

(Mestre)

Dissertação para obtenção do grau de doutor em

Engenharia Electrotécnica e de Computadores

Orientador:
Doutor Carlos Eduardo do Rego da Costa Salema,
Professor Catedrático do Instituto Superior Técnico da Universidade Técnica de Lisboa

Co-Orientador:
Doutor Augusto Afonso de Albuquerque,
Professor Catedrático do Instituto Superior das Ciências do Trabalho e da Empresa

Constituição do Júri:

Presidente:
Reitor da Universidade Técnica de Lisboa

Vogais:
Doutor Augusto Afonso de Albuquerque,
Professor Catedrático do Instituto Superior das Ciências do Trabalho e da Empresa
Doutor José Manuel Nunes Leitão,
Professor Catedrático do Instituto Superior Técnico da Universidade Técnica de Lisboa
Doutor Luis António Pereira Menezes Corte-Real,
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto
Doutor Mário Alexandre Teles de Figueiredo,
Professor Auxiliar do Instituto Superior Técnico da Universidade Técnica de Lisboa
Doutor Jorge dos Santos Salvador Marques,
Professor Auxiliar do Instituto Superior Técnico da Universidade Técnica de Lisboa
Doutora Maria Paula dos Santos Queluz Rodrigues,
Professora Auxiliar do Instituto Superior Técnico da Universidade Técnica de Lisboa

Lisboa, Março de 1999

Aos meus amigos.

Je n'ai fait celle-ci plus longue que parce que je n'ai pas eu le loisir de la faire plus courte.

Blaise Pascal

Acknowledgements

It is sometimes said that the order by which names are mentioned in the acknowledgments has no special meaning. Not in this case. Without Prof. Augusto Albuquerque's friendship, without his encouragements, and without his supervision, this thesis would not have been possible. Thanks.

To João Luis Sobrinho and Carlos Pires, for their friendship and support.

To all my colleagues in ISCTE, for their good will. Special thanks to Luis Nunes and Filipe Santos, they know why.

To the staff of the Image Group of the Universitat Politècnica de Catalunya, for accepting me as their own during a month of valuable discussions. It was a fundamental month.

To Prof. Carlos Salema, for his willingness to supervise this thesis all this time. For his patience.

To the Image Group of IST, for their friendship and for the pleasant lunch discussions, always stimulating, seldom technical...

To the Instituto de Telecomunicações, for the office and equipment I used for so long.

To Prof. Fernando Pereira whom, by allowing me to work for the CEC RACE MAVT project, greatly facilitated my contacts with the international video coding community.

To my friends, who always trusted me more than I did myself.

This thesis was partially funded by the CIENCIA and PRAXIS programs of JNICT. A substantial part of this work was done while I was working for the CEC MAVT RACE project, and while I was working as a research assistant in ISCTE.

Agradecimentos

Costuma-se dizer nos agradecimentos que a ordem pela qual os nomes aparecem não tem qualquer significado. Não é o caso. Sem a amizade do Prof. Augusto Albuquerque, sem os seus encorajamentos e sem a sua orientação, esta tese não teria realmente sido possível. Bem haja.

Ao João Luis Sobrinho e ao Carlos Pires, pelo apoio e amizade.

Aos colegas do ISCTE, agradeço a sua enorme boa vontade. Agradecimentos especiais ao Luis Nunes e ao Filipe Santos, eles sabem porquê.

A todos os investigadores do Grupo de Imagem da Universidade Politécnica da Catalunha, que me aceitaram no seu seio durante um mês de profícuas discussões. Foi um mês fundamental.

Ao Prof. Carlos Salema, por se dispor a orientar esta tese, ao longo de todo este tempo. Pela sua paciência.

Ao grupo de imagem, pela amizade e camaradagem, e pelas discussões ao almoço, sempre estimulantes, raramente técnicas...

Ao Instituto de Telecomunicações, pelas infraestruturas e equipamento informático que me permitiu utilizar.

Ao Prof. Fernando Pereira, por, através da participação no projecto MAVT, me ter permitido manter contactos científicos internacionais muito mais difíceis de outra forma.

Aos meus amigos, que sempre acreditaram em mim muito mais do que eu próprio.

Esta tese foi parcialmente financiada pela JNICT, programas CIENCIA e PRAXIS. Uma parte substancial deste trabalho foi realizada no âmbito projecto RACE MAVT, e também na minha qualidade de assistente do ISCTE.

Abstract

Video coding has been under intense scrutiny during the last years. The published international standards rely on low-level vision concepts, thus being first-generation. Recently standardization started in second-generation video coding, supported on mid-level vision concepts such as objects.

This thesis presents new architectures for second-generation video codecs and some of the required analysis and coding tools.

The graph theoretic foundations of image analysis are presented and algorithms for generalized shortest spanning tree problems are proposed. In this light, it is shown that basic versions of several region-oriented segmentation algorithms address the same problem. Globalization of information is studied and shown to confer different properties to these algorithms, and to transform region merging in recursive shortest spanning tree segmentation (RSST). RSST algorithms attempting to minimize global approximation error and using affine region models are shown to be very effective. A knowledge-based segmentation algorithm for mobile videotelephony is proposed.

A new camera movement estimation algorithm is developed which is effective for image stabilization and scene cut detection. A camera movement compensation technique for first-generation codecs is also proposed.

A systematization of partition types and representations is performed with which partition coding tools are overviewed. A fast approximate closed cubic spline algorithm is developed with applications in partition coding.

Keywords: visual coding, second-generation video coding, image analysis, image segmentation, temporal coherence, motion estimation.

Resumo

A codificação de vídeo tem sido intensamente estudada nos últimos anos. As normas internacionais já publicadas baseiam-se em conceitos da visão de baixo nível, sendo portanto de primeira geração. Começou recentemente a normalização de técnicas de codificação de segunda geração, suportada em conceitos da visão de médio nível tais como objectos.

Esta tese apresenta novas arquitecturas para codificadores de vídeo de segunda geração e algumas das correspondentes ferramentas de análise e codificação.

Apresentam-se fundamentos de teoria dos grafos aplicada à análise de imagem e propõem-se algoritmos para generalizações do problema da árvore abrangente mínima. Mostra-se que versões básicas de vários algoritmos de segmentação orientados para a região resolvem o mesmo problema. Estuda-se a globalização de informação e mostra-se que confere propriedades diferentes a esses algoritmos, transformando o algoritmo de fusão de regiões no algoritmo de árvores abrangentes mínimas recursivas (RSST). Mostra-se a eficácia de algoritmos RSST que tentam minimizar o erro global de aproximação e que usam modelos de região afins. Propõe-se um algoritmo baseado em conhecimento prévio para segmentação em vídeo-telefonia móvel.

Desenvolve-se um um algoritmo de estimação de movimentos de câmara eficaz na estabilização de imagem e na detecção de mudanças de cena. Propõe-se também uma técnica de compensação de movimentos de câmara para codificadores de primeira-geração.

Sistematizam-se os tipos e as representações de regiões, revendo-se depois técnicas de codificação de partições. Desenvolve-se um algoritmo rápido e aproximado para cálculo de *splines* cúbicas fechadas.

Palavras chave: codificação visual, codificação de vídeo de segunda geração, análise de imagem, segmentação de imagem, coerência temporal, estimação de movimento.

Contents

List of Acronyms

2D	Two-dimensional
3D	Three-dimensional
ADSL	Asymmetric Digital Subscriber Line
ANSI	American National Standards Institute (a standards body)
CAG	Class Adjacency Graph
CATV	Cable Television (formerly Community Antenna Television)
CCIR	Comité Consultatif Internationale des Radio Communications (now ITU-R)
CCITT	Comité Consultatif Internationale de Télégraphique et Téléphonique (now ITU-T)
CD	Compact Disk
CD	Committee Draft (of an ISO standard)
CEC	European Community Commission
CERN	Conseil Europeen pour la Recherche Nucleaire
CIE	Commission Internationale de l'Éclairage
CIF	Common Intermediate Format
CPU	Central Processing Unit
CRT	Cathode-Ray Tube
DCT	Discrete Cosine Transform
DFS	Depth First Search
DFS	Discrete Fourier Series
DPCM	Differential Pulse Code Modulation

DSig	Digital Signatures (of W3C) `http://www.w3.org/DSig/Overview.html`
FCT	Four-Color Theorem
FIR	Finite Impulse Response
FLC	Fixed Length Code
FSF	Free Software Foundation
FTP	File Transfer Protocol
GOB	Group of Blocks (a syntax element in ITU-T H.261)
GPL	GNU General Public Licence (of FSF)
HDTV	High Definition Television
HSV	Hue, Saturation, and Value (a color space)
HTML	Hypertext Markup Language (of W3C)
HTTP	Hypertext Transfer Protocol (of IETF and of W3C)
HVS	Human Visual System
IEC	International Electrotechnical Commission (a standards body) `http://www.iec.ch/`
IEEE	The Institute of Electrical and Electronics Engineers, INC. `http://www.ieee.org/`
IETF	Internet Engineering Task Force (a standards body) `http://www.ietf.org/`
IIR	Infinite Impulse Response
IP	Internet Protocol
IPR	Intellectual Property Rights
IRC	Internet Relay Chat
ISDN	Integrated Services Digital Network (of ITU-T)
ISO	International Organization for Standardization (a standards body) `http://www.iso.ch/`
ITU	International Telecommunication Union (a standards body) `http://www.itu.ch/`
ITU-R	ITU Radiocommunication Sector (a standards body, see ITU) `http://www.itu.ch/ITU-R/`

ITU-T	ITU Telecommunication Standardization Sector (a standards body, see ITU) `http://www.itu.ch/ITU-T/`
JPEG	Joint Photographic Experts Group (of ISO and IEC)
LMedS	Least Median of Squares
LoG	Laplacian of Gaussian
LSF	Longest Spanning Forest
LST	Longest Spanning Tree
MAVT	Mobile Audio-Visual Terminal
MB	MacroBlock (a block of 16×16 pixels, on ITU-T and ISO/IEC video coding standards)
MBA	MacroBlock Address (a syntax element in ITU-T H.261)
MC	Motion Compensated
MF	Model Failure
MMREAD	Modified MREAD (see MREAD)
MPEG	Moving Picture Experts Group (of ISO and IEC)
MREAD	Modified Relative Element Address Designate
MTYPE	Macroblock Type (a syntax element in ITU-T H.261)
MVD	Motion Vector Data (a syntax element in ITU-T H.261)
NTSC	National Television Systems Committee (a standards body)
OALDCE	Oxford Advanced Learner's Dictionary of Current English
OCR	Optical Character Recognition
PAL	Phase Alternating Line
PC	Personal Computer
PEI	Picture Extra Insertion Information (a syntax element in ITU-T H.261)
PICS	Platform for Internet Content Selection (of W3C) `http://www.w3.org/PICS/`
PNG	Portable Network Graphics (of W3C)
POTS	Plain Old Telephone Service
PPV	Pay-Per-View

PSNR	Peak Signal to Noise Ratio
PSPARE	Picture Spare Information (a syntax element in ITU-T H.261)
PSTN	Public Switched Telephone Network
PTYPE	Picture Type (a syntax element in ITU-T H.261)
QCIF	Quarter-CIF (see CIF)
QPT	Quartic Picture Tree
RAG	Region Adjacency Graph
RAMG	Region Adjacency MultiGraph
RBPG	Region Border PseudoGraph
RGB	Red, Green, and Blue (a color space)
RLE	Run-Length Encoding
RM8	Reference Model 8 (a reference model for ITU-T H.261)
RSST	Recursive SST (see SST)
SIF	Standard Interchange Format
SMIL	Synchronized Multimedia Integration Language (of W3C) `http://www.w3.org/TR/WD-smil`
SSF	Shortest Spanning Forest
SSkT	Shortest Spanning k-Tree
SSSkT	Shortest Seeded Spanning k-Tree
SSSSkT	Smallest Shortest Seeded Spanning k-Tree
SST	Shortest Spanning Tree
TCP	Transmission Control Protocol
TR-RSST	Time-Recursive RSST (see RSST)
TV	Television
UMTS	Universal Mobile Telecommunication Service
UPC	Universitat Politècnica de Catalunya
URC	Uniform Resource Characteristics (of IETF)
URI	Uniform Resource Identifier (of IETF)
URL	Uniform Resource Locator (of IETF)

URN	Uniform Resource Name (of IETF)
VLC	Variable Length Code
VO	Video Object (a syntax element in ISO/IEC MPEG-4)
VOD	Video-On-Demand
VOL	Video Object Layer (a syntax element in ISO/IEC MPEG-4)
VOP	Video Object Plane (a syntax element in ISO/IEC MPEG-4)
VQ	Vector Quantization
VRML	Virtual Reality Modeling Language (a standard of ISO and IEC)
W3C	World Wide Web Consortium (a standards body) `http://www.w3c.org/`
WWW	World Wide Web

Chapter 1

Introduction

"The time has come," the Walrus said,
"To talk of many things:"

Lewis Carroll

The performance of classical video coding algorithms, in terms of the classical coding criteria (bitrate, distortion, and cost), seems to be reaching a plateau [161, 3]. That is, the marginal performance gains of tuning these algorithms are now nearly negligible. According to Adelson et al. [3, 195], the classical approaches use concepts usually related to low-level vision, such as luminance, color, spatial frequency, temporal frequency, local motion, and low-level operators such as linear filtering and transforms. New approaches, using mid-level visual concepts, such as regions, textures, surfaces, depth, global motion, and lighting, are deemed necessary for a breakthrough in video coding performance. This need has been recognized for some time now [144, 141, 96], though limited computing capabilities have hindered somewhat the advances towards the implementation of complete mid-level vision video (second-generation) coding algorithms.

During the last years, and following the ever increasing advances of technology, the use of image and video in everyday life has been growing continuously. This has sparkled new needs among users: interactivity, content editing, and content based indexing are just a few examples. These needs require the access to the content of video sequences. This access may, in some cases, be done after encoding and decoding, i.e., by performing analysis at the receiver side. In most cases, though, it is essential to have this capability directly at bit stream level. Content access should thus be done with a minimum of effort: a "fourth [coding] criterion" has been identified, coined by Picard [162] as "content access effort." This criterion is related to the complexity or effort required to access the video content, and hence to provide content-based facilities.

The results obtained until now by mid-level vision video coding algorithms, though extremely important, do not show performance improvements as large as initially expected [177, 40, 30].

However, this apparent lack of success is truly a misjudgment, since the performance has been measured, until now, using only the bitrate, distortion, and cost criteria. When the fourth criterion is introduced, the newly developed algorithms certainly have a leading edge over the classical ones: objects and regions, rather than square blocks, are what an user wants to interact with.

The new users' needs have also been recognized by MPEG-4. These ideas were introduced in MPEG-4 [138] by asking for some "new or improved functionalities" [139]: content-based manipulation and bit stream editing, content-based multimedia data access tools, and content-based scalability.

This thesis summarizes a series of proposals towards coding of visual objects. The work has progressed over a number of years and can be seen as a contribution to the development of second-generation visual coding standards of which MPEG-4 is an example.

1.1 Structure of the thesis

Chapter 2, "Video and multimedia communications", contains a brief overview of multimedia, the Internet and video communications. It can be seen as a motivation for the work developed. Video codecs are classified as first-, second-, or third generation according to the analysis tools required: first-generation for low-level vision analysis, second-generation for mid-level vision analysis, and third-generation for high-level vision analysis. A brief summary of the analysis and coding tools proposed in this thesis, organized according to the presented structure, can be found in Section 2.6.

Chapter 3, "Graph theoretic foundations for image analysis", defines most of the theoretical concepts that are used throughout. In this chapter the important theory of spanning trees, a branch of graph theory, and related concepts using seeds, is discussed together with the corresponding algorithms. An amortized linear time algorithm is also presented for an important class of spanning tree problems.

Chapter 4, "Spatial analysis", contains proposals for a knowledge-based mobile videotelephony segmentation algorithm, an extended RSST (Recursive SST) segmentation algorithm using an affine region model, a supervised RSST segmentation algorithm, i.e., a RSST algorithm using seeds, and a time-recursive version of the RSST algorithm providing time coherent segmentation of moving images. The classical segmentation algorithms, such as region growing, region merging, edge detection followed by contour closing, are all described in the framework of the theory of spanning trees introduced in the previous chapter. The relations between these algorithms is discussed in the common framework of spanning trees. The effects on these algorithms of globalization of information are also discussed.

Chapter 5, "Time analysis", proposes a simple algorithm for estimating camera movement in moving images and a method for its cancellation (image stabilization) to improve image quality in hand-held or car-mounted cameras. The algorithm is based on a motion vector field obtained through block matching. Several results are shown which demonstrate its effectiveness.

Chapter 6, “Coding”, proposes a method of encoding camera movement information using a simple extension to the H.261 standard (the discussions on quantization are general and transposable to any other codec using motion vector fields with reduced resolution relative to that of the underlying images) and reviews the important issue of partition representation and coding. A fast approximation to the calculation of closed cubic splines is also proposed. The analysis and coding tools presented in this and the previous two chapters can be seen as steps towards the building of tools for a new codec architecture.

Chapter 7, “Conclusions: Proposal for a new codec architecture”, proposes a new second-generation codec architecture, makes some suggestions for future work, and lists the thesis contributions.

Finally, Appendix A describes the test sequences used and their formats, and Appendix B contains a very brief description of the Frames video coding library, which was developed by the author as the basis for the implementation of all the algorithms.

Chapter 2

Video and multimedia communications

It is supposed that because a thing is the rule it is right.

Oscar Wilde

2.1 Trends of multimedia communications

"Medium" literally means "middle". According to the OALDCE (Oxford Advanced Learner's Dictionary of Current English) [71], it means "that by which something is expressed," i.e., that by which a message is expressed, since, according to Negroponte [142], "the medium is not the message." Messages can be expressed using a variety of media. Multimedia is the process of expressing a message using several media. In this sense, multimedia is not new. Multimedia exists since there are books with images,[1] actually even before that, since humans communicate by speech and gestures.

Until last century our ability to store and transmit messages was very limited. Only text and still images and diagrams could be stored for future use (e.g., in books), and long range transmission was limited to physical transport of printed or handwritten material, with rare exceptions. The telegraph, for long range transmission of text, the telephone, for long range transmission of voice, the radio, for long range transmission of sound, changed that picture considerably. But perhaps the most important inventions of the last century were the phonograph, for storing sounds, and the cinematograph, by which storing of moving images became possible.

[1]Different media can share the same sense (or "channel") into the human brain. Text and imagery, though different media, are both sensed using vision.

In the beginning of this century it was possible, at least in principle, to express messages using multimedia as we know it today and store them for future use. In practice, this happened only in the thirties, with the introduction of sound synchronized with image in the cinema. Stereoscopic imagery was also available at that time.

2.1.1 Distribution methods

A message, as expressed through moving images and sound in a film, is meant to be conveyed to a receptor. Although movie theaters are still a very successful and profitable way of doing it, they involve considerable delay and trouble. Using Negroponte's [142] "bits" and "atoms" definitions, the producer distributes the film cartridges (atoms) containing encoded images and sounds (bits) which are then broadcasted from a screen and speakers to a restricted audience.[2] A new distribution paradigm was clearly necessary.

TV (Television) partially solved the distribution problem, by using radio broadcast of analogically encoded moving images and sound. However, TV also introduced some new problems: being broadcasted, anybody with a TV set could enjoy it. Who (and how) should then pay for the content conveyed? From TV taxes (virtually unchargeable), to income taxes (in the case of subsidized television), through advertisements and mixtures thereof, several solutions have been proposed, most of which are still being used to this day. These solutions were not enough. Point-to-point communication, such as that provided by the telephone, was necessary.

Computer networks, providing point-to-point communications in a different framework, were also an important development. In the 1970's the TCP (Transmission Control Protocol)/IP (Internet Protocol) protocols were developed and put to use mostly by the government and educational institutions in the USA. By the eighties it was spread all over the world, though mostly restricted to the academic world. In the beginning of the nineties, following the development by the CERN (Conseil Europeen pour la Recherche Nucleaire) of the suite of WWW (World Wide Web) protocols and formats, viz. UR*,[3] HTTP (Hypertext Transfer Protocol), and HTML (Hypertext Markup Language), the Web exploded: it became attractive to the common user, and hence economically viable.

In the late forties, TV started to be distributed by cable in areas where the broadcast signal could not be received with normal antennas (community antenna television). Cable television was soon found to offer considerable advantages relative to broadcast television: increased quality, increased number of channels through a larger available bandwidth, no need for antennas and thus lower visual impact (important in certain urban areas), etc. Recently, CATV (Cable Television) operators, typically diffusion oriented, realized they had deployed over the years an almost ubiquitous broadband network which could be improved with small investments to provide up-links to the user. Thus, with the help of cable modems, providers started building a sort of "residential area networks", connecting users in the neighborhood to the cable head-end

[2]Images and sounds in film are mostly encoded in an analog format, even though digital sound is expanding quickly. These images and sounds contain information, which can be measured in bits, even if digital encoding is not used.

[3]E.g., URC (Uniform Resource Characteristics), URI (Uniform Resource Identifier), URL (Uniform Resource Locator), and URN (Uniform Resource Name).

and thence to the world.

The explosion of the Web in the nineties, together with the personal computer and the almost ubiquitous wide band CATV networks, suddenly allowed different content to be delivered to different consumers. Consumers could now choose and even interact with the material delivered (and pay accordingly): the age of the Web, teleshopping, PPV (Pay-Per-View), VOD (Video-On-Demand) and WebTV was born.

2.1.2 Activity paradigms

There are essentially two activity paradigms for information provided to the consumers. The push paradigm, when the information provider pushes the information to a passive user, and the pull paradigm, when the active user requests information from the service provider.

TV broadcast is push, since the information is pushed to the consumer without requiring any action on his part (besides turning the TV on and choosing a channel). However, VOD is pull, since the user requests whatever interests her.

The Web, until recently, exhibited only the pull behavior. All the action was on the part of the end user, which would always make specific requests as to what information should be delivered to him. Nowadays, the push paradigm has been implemented by most browsers, through the concept of automatically updated channels, in a clear parallel with TV diffusion.

2.1.3 Convergence tendencies

Convergence of distribution methods and technologies

A wealth of communication services exist today. Most of the channels involved in these services are slowly being enhanced to provide bidirectional communications and improved bandwidth. For instance, CATV networks now provide bidirectional data channels through cable modems, satellite constellations are being deployed for personal mobile communications, and the UMTS (Universal Mobile Telecommunication Service), providing a wider bandwidth than today's cellular phones, is expected in the near future. Also, the analog channels offered by the old PSTN (Public Switched Telephone Network) are slowly being digitized to provide ISDN (Integrated Services Digital Network). Recently, ADSL (Asymmetric Digital Subscriber Line) started to be used to establish wideband data channels on the telephonic copper loop.

At the same time, all fields of multimedia and communications are being enhanced through the use of digital technology. Digital recorded sound is already used in the movie theaters (probably to be followed soon by digital moving images) and digital TV will soon be available, and affordable, in all developed countries. There is, thus, a clear convergence towards both wideband (except where physically impossible) and bidirectionality.

Convergence of services and activity paradigms

The services available are also converging. There is a tendency to support both broadcast and point-to-point distribution, different media, and both push and pull paradigms. Cable modems allow point-to-point communications where formerly broadcast was the rule, and modems over POTS (Plain Old Telephone Service) allow broadcast (or at least the Web equivalent of broadcast, multicast) where formerly only point-to-point communications was used. Videotelephone over POTS is now possible (and soon will be also possible on cellular phones), and the TV service was long ago upgraded to include teletext. Videotelephony, on the other hand, is also possibly on the Web, and supplements the old pear-to-pear communications services of the Internet such as email and (electronic) talk, and, more recently, IRC (Internet Relay Chat).

Convergence of contents

On the demand side, consumers require high quality content. The production of multimedia content, in which the entertainment industries (TV, cinema and games) excel, is thus thriving. Consumers are also demanding more and more interactive control over the information they receive, an issue which is a specialty of the informatics (software/computer) industries. Thus the tendency for mergers and acquisitions between companies in the entertainment, informatics and network businesses.

Consumers also require mobility and compatibility. Thus large informatics companies are also investing on global, satellite based, mobile networks, and more and more care is taken nowadays with standardization and compatibility by content providers, TV companies, and informatics companies.

2.1.4 A distributed database

It seems reasonable to expect that the convergence process will lead to universal access to information. There will probably be little difference between TV, phone, fax, and the PC (Personal Computer). In fact, the PC is already doubling as TV, a phone, and a fax. The Web will connect almost everything and everyone. It is expected to provide people with a complete leisure, work, and social environment, accessed through a wealth of different interfaces, such as screens together with remote controls, desktop screens, keyboards, pointing devices, microphones and speakers, voice controlled hand-held devices with handwritten character recognition (e.g., evolutions of the PalmPilot™), data-gloves, etc. Such a network of information can be seen a huge distributed, chaotic, database. Even today, the amount of information is such that special purpose indexing mechanisms and search engines are being developed.

2.2 Media representation

Take a CD (Compact Disk) of an orchestra playing Mozart's symphony 41, Jupiter. What is the essential part: the score or the sound (a unique interpretation)? Although 600 megabytes are used in a typical CD to store the sound, the corresponding score may be stored in much less space. CD audio does not encode the structure: it encodes, as faithfully as possible, a copy of the original sound. The same thing happens with fax: I still find it frustrating to explain to users of fax modems why it is they can not import the received faxes (mostly text) directly into their word processors, without using the (still) error prone OCR (Optical Character Recognition) software.

Consider, however, that faxes were made intelligent: they would analyze the input page, detect text zones, recognize the text, and encode it as text, instead of black and white raster images of characters.[4] This would clearly lead to improved usability, if not also to a reduction of transmission time.

Visual data, especially video (taken here as sequences of images sampled from the natural world scene), is a very important part of today's multimedia, and its importance tends to increase with the convergence of entertainment and informatics industries. However, video is still encoded with the same "blindness" that affects fax and CD sound: the structured contents of video scenes are simply ignored in the encoding process, leading to a representation which is not at all structural [142], faithful as it may be to the original.

Visual analyzers would do the same for video as the hypothetical fax analyzer for a black and white image: from a sequence of video images, they would extract a structural representation of the scene therein, the scene's "score" plus "interpretation nuances". Such a structural representation, aside from the expected economies in encoded size, would allow the user to manipulate the scene at will: a big step towards complete interactivity.

The exponential growth of digital technology, where clock frequencies duplicate almost every year and memory densities (bits per volume) almost triplicate in the same period of time, has led to an ever increasing use of computers by content providers (such as film producers and TV companies). Synthetic imagery corresponds nowadays to an important part of the bits exchanged worldwide. However, not much effort was put until now into the efficient (soon to be defined) representation of synthetic data, which is inherently structural.

Hence, two important problems must be solved urgently: how to obtain structural representations from natural data (the score and the interpretation nuances from a symphony recording, the text from a printed document, the description of the scene seen in a video sequence) and how to efficiently encode structural representations, either synthetic or obtained from natural data.

The first of these problems is analysis. In the case of visual data, analysis is addressed by computer vision which, according to [68, Haralick and Shapiro], is "the science that develops the theoretical and algorithmic basis by which useful information about the world can be auto-

[4]When the original text is composed using a word processor, sending it by fax to a remote computer is a bit of a paradox, even though it is still quite common.

matically extracted and analyzed from an observed image, image set, or image sequence from computations made by special-purpose or general-purpose computers."

The second problem is related to the encoding of the structural description of the data. In the case of visual data, several encoding methods have been devised in the past, ranging from the analog television standards such as NTSC (National Television Systems Committee) and PAL (Phase Alternating Line), to the digital video coding standards ITU-T (ITU Telecommunication Standardization Sector) H.261 [62], ISO (International Organization for Standardization)/IEC (International Electrotechnical Commission) MPEG-1 [136], and, more recently, ITU-T H.263 [63] and ISO/IEC MPEG-2 [137] (also ITU-T H.262). These standards have typically dealt with non-structural representations of imagery. The first standard to address structured moving image representations will be ISO/IEC MPEG-4.

2.3 Visual analysis

Even though synthetic data amounts to a relevant part of the available multimedia material, natural data will always be present. Natural data corresponds to data which is obtained, usually through sampling, from the real world. While it is reasonable to expect that sensors, such as video cameras, will increase in complexity over the years, for instance by incorporating distance or depth sensors, it is unlikely that they will ever provide a structural representation of the sampled data at their output.

Hence, analysis, that is, the decomposition of the input data into a meaningful set of some model parameters, is a very important task. Automatic visual analysis, as stated before, is almost the same as computer vision: "building a description of the shapes and positions of things from images" [107]. With one difference, however. The purpose of computer vision is ultimately the comprehension of the scene captured by the camera, through an emulation of the HVS (Human Visual System), while analysis usually has more modest objectives.

Analysis, as stated, is the identification of some model parameters. This makes modeling one of the most important tasks in research leading to automatic analysis of video sequences, since it seems clear that sophisticated models can lead to a very accurate representation of the world, but only at the cost of a very sophisticated, or even impossible, analysis: visual analysis is often an ill-posed problem [187, 9].

Visual analysis can have several purposes [29]:

Analysis for coding
The obtaining of a parametric description of the observed scene. The description can later be used to reconstruct the scene so that little or no information is lost. The description can also be encoded and decoded efficiently (analysis for bandwidth saving), and can also be manipulated (analysis for easy access), so that the user can interact with the represented world. The analogy with fax helps here. With "blind" fax, such as exists today, to edit text just received is a nightmare. With intelligent fax, however, text would be received as such, and thus be fully editable. The same applies to video.

Analysis for description or indexing

The obtaining of a parametric description, though in this case it is not necessary to be able to reconstruct the observed scene or at least the original sampled (or sensed) data. The parameters of the description have mostly a semantic meaning, which may help the task of searching visual data in a database. The model parameters, or features, estimated or identified, will be used as keys of the database.

Analysis for understanding

The process leading to understanding of the observed scene. While visual analysis tools in general are tools leading to artificial intelligence, or so one expects, analysis for understanding *is* artificial intelligence proper.

Analysis can be manual, automatic, or partially automatic, when an automatic algorithm is guided by user input (hints). An usual path in the research in this area, which, though it progresses very quickly, has still a long way to go, is to allow the algorithms to be supervised and then attempt to make them automatic. This is a polemic issue, however, as can be seen in the article "Ignorance, myopia, and naivete in computer vision systems" [81] and in the subsequent dialogue in [7] and [94].

2.3.1 Levels of visual analysis

Some authors divide the vision process into levels [107, 195] which are related to the types of models or primitives assumed:

Low-level[5] vision

The model is a sequence of pixel matrixes. The correlation between pixels is assumed to be high. Evolution from one image to the next is described by a simple motion field, uniform almost everywhere.

Mid-level[6] vision

The model is a possibly hierarchical set of edge segments, blobs, uniformly textured regions (or equivalently boundaries) or regions of uniform motion. Surfaces and their relative position may also be used. Motion can be associated with segments and/or edges or boundaries.

High-level[7] vision

The model is a set of 3-D objects arranged hierarchically. Objects are semantically identified. Each object has an associated complex motion.

Understanding

The role, class or identity of (almost all of) the objects is known.

[5]Or image.

[6]Or primal plus 2 1/2-D sketches.

[7]Or 3-D model representation.

This division is here a mere matter of convenience. It is also somewhat arbitrary, since feedback mechanisms seem to exist between the upper and the lower levels of the vision process. Visual analysis will be classified in the following according to the first three levels, since understanding is not one of the purposes here. The terms low-level, mid-level and high-level analysis will be used throughout this thesis.

2.3.2 Tools for visual analysis

Analysis can be seen as being done at three levels: low-, mid-, and high-level. Different image analysis tools have been developed over the years which can be classified as belonging to each of these levels. Restricting attention to those tools more closely related to analysis for coding, the following (rather incomplete) classification can be used:

Low-level vision analysis
Linear transformations (transforms), frequency analysis, motion estimation (optical flow, block matching), etc.

Mid-level vision analysis
Edge detection, contour detection, segmentation into syntactically uniform regions, motion estimation (motion of edges and regions), etc.

High-level vision analysis
3D (Three-dimensional) structure from shading and motion, 3D structure from disparity (stereo vision), etc.

2.4 Visual coding

Coding[8] is the process of translating a sequence of symbols belonging to a given alphabet, the message,[9] into a sequence of symbols of a different alphabet (usually the binary alphabet). Coding is said to be lossless if the original message can be recovered exactly from the encoded one.

Visual coding is the process by which the parameters of the structural representation of a visual scene obtained either by analysis or directly, in the case of synthetic imagery, are encoded. When the representation is obtained by analysis of natural data, the term video coding if often used.

[8]Coding should always be understood as referring to source coding throughout this thesis, as opposed to channel coding.

[9]Note the different meanings of the word "message". In a communications framework, it is the set of ideas expressed using a given medium or ensemble of media. In the context of information theory, it is a sequence of symbols, to which a measure of information can be associated [183].

2.4.1 Objectives

Encoding, the translation between one alphabet and another, can have several objectives. It can be seen as the process of minimizing a cost functional given some constraints. There are several measures which can be used to express both the cost functional and the constraints, and which, weighted differently, reflect the objectives of each particular coding scheme:

Compression ratio (or, inversely, bitrate)
: The size of the original message divided by the size of the encoded message, both expressed in bits. By maximizing compression, the bandwidth or space requirements are reduced, according to whether the data is transmitted or stored.

Quality (or, inversely, distortion)
: A measure of the difference between the original message and the one obtained by decoding the encoded message. Error resilience is accounted for in this measure by allowing errors to affect the encoded data.

Cost
: The cost of the encoder and decoder (weighted appropriately).

Content access effort
: A measure of the easiness with which only specified parts of the original message can be recovered from the encoded message. By maximizing ease of access, simple terminals can still allow the user to manipulate the scene. Video trick modes can also be seen as requiring easy access to contents (in this case to single video images).

Delay
: The interval between the instant a symbol of the original message is input to the encoder and the corresponding symbol is output from the decoder, assuming no channel delay.

Quality is perhaps the most difficult measure to make, in the case of visual coding. How can an objective measure of quality reflect the quality of the reconstructed scene as perceived by humans? Even though studies have been conducted over the years to develop such a measure, based on the properties of the HVS, no single universally accepted measure exists. Two measures of quality are typically used today in the case of video coding: a simple objective measure, called PSNR (Peak Signal to Noise Ratio), and subjective quality measures based on evaluation by a significant set of persons.

Cost is related mostly to implementation of encoders and decoders, though it can be related also to the required bandwidth, which is dependent on the compression ratio, and thus already considered through that measure. Implementation costs can be related to the memory and CPU (Central Processing Unit) power required for both encoders and decoders.

The cost functional and constraints can be constructed from the measures above so as to reflect the different requirements of an application. Some applications may require quality as high as possible for a minimum allowed compression ratio, others the highest possible compression for a minimum allowed quality. The cost of coders and decoders may be weighted differently:

applications where content is encoded once and decoded many times put a larger weight on the cost of decoders.

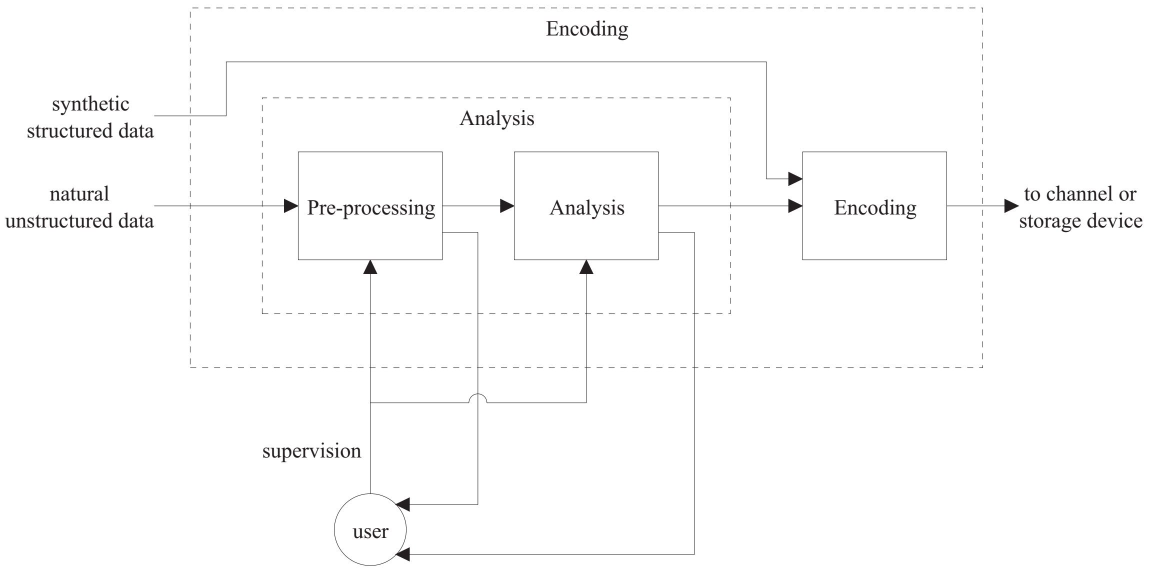
2.4.2 Main codec blocks

Figure 2.1 shows a typical block structure of a codec. The encoder part consists of an analysis block, which obtains a structural scene representation from given natural data, followed by the encoder, which encodes this representation so as to be sent down a logical channel (either a real channel or some physical storage medium). If synthetic data is available, it is input directly to the encoder without being analyzed, provided it is already described in an appropriately structured way. The decoder performs the opposite tasks. The encoded data is decoded so as to obtain the structural scene representation which is then used by the renderer to create appropriate stimuli to the human receivers, which can have different levels of interactivity with the system.

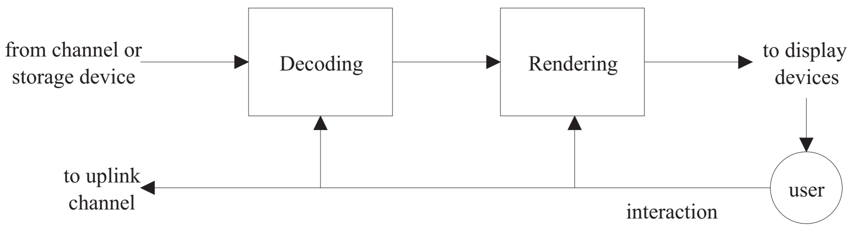
Often some processing is performed on the natural data before the analysis proper. This processing usually intends to filter or condition the data so as to render the analysis simpler or more effective. Since it takes place before analysis and encoding, it is called pre-processing. It is often taken as being part of the analysis itself.

The word encoder is used here with two different meanings: in the case of natural data, which requires analysis, encoder can both mean the complete system, from natural data representation to the resulting encoded message, or simply the block which translates the structural representation into the encoded message, which is the strict meaning. In the sequel the exact meaning will be evident from the context.

An encoder, in the broad sense of the word, serves two main purposes. Firstly, it is supposed to strip irrelevant information (from the point of view of the assumed receiver of the information, usually the HVS) from the input. Irrelevancy removal is done by the analysis block, since, according to Marr [107], "vision is a process that produces from images of the external world a description that is useful to the viewer and not cluttered with *irrelevant information* [our emphasis]," and to emulate vision is the ultimate purpose of analysis. Secondly, the encoder, again in the broad sense, is supposed to remove redundancy. This is a role which is shared by the analysis and the encoder blocks, though the kind of redundancy removed is different. The analysis block removes representation redundancy by fitting the input data to a structural model. For instance, the highly redundant image of a sphere can be described, with an appropriate model, by the position and size of the sphere, its surface characteristics, and a set of light sources. Such a description is much less redundant than the original array of pixels. The encoder block, on the other hand, removes statistical redundancy from the sequence of symbols corresponding to the structural representation. It must be stressed here that removal of redundancy is a reversible process, while removal of irrelevancy is not. In a sense, thus, it is desirable that losses in coding correspond as much as possible to removal of irrelevancy.


Encoding
synthetic structured data
Analysis
natural unstructured data
Pre-processing
Analysis
Encoding
to channel or storage device
supervision
user

(a) Encoder.



(b) Decoder.

Figure 2.1: Basic block structure of a codec.

2.4.3 Generations

In the case of natural scenes, i.e., video coding, analysis is performed before encoding proper, as can be seen in Figure 2.1. Video encoding techniques can thus be classified according to the level of analysis typically required. The terms first- and second-generation video coding were coined by Kunt et al. [96], and correspond approximately to the two first levels of analysis presented before. The requirements in terms of analysis of these two generations of video coders, plus a third one related with high-level analysis are as follows:

First-generation

Coders which require low-level analysis. Hybrid coders [64] and motion compensated hybrid coders [145] belong to this generation. The fundamental tools used in these coders, DCT (Discrete Cosine Transform) and block matching motion compensation, were already developed in the beginning of the eighties. All the already issued video coding standards belong to this generation.

Second-generation
Coders which require mid-level analysis. This type of analysis is typically more complex than low-level analysis. Even though a lot of effort has been put into this field, a truly reliable set of mid-level analysis tools is not yet mature. This thesis contributes mainly to the problem of developing tools at this level.

Third-generation
Coders which require high-level analysis. No truly reliable automatic analysis set of tools exists at this level. Most tools still rely on human supervision, and it probably will remain so for a few more years: most of the semantic features/descriptors can only be extracted by humans at the present time [29].

This classification, though useful, is somewhat artificial. For instance, a mid- or high-level analysis tool can be used to enhance a first-generation video encoder. This often happens when video encoding algorithms are being enhanced.

2.5 Standards

Standards are fundamental for universality of service and interworking, both of which are of paramount importance for the end consumer. Standardization, however, is a time-critical process: if done too soon, it may not benefit from the ongoing research in the area, if done too late, it may have to face proprietary solutions proposed by industries of sufficient weight to make the standard useless.

Standards may be of two very different natures. English is a *de facto* language standard in most of the western world. French, on the other hand, is a *de jure* standard, at least in France: it is standardized by the Academie Française and imposed by the French state in official documents. The case with technologies is similar.

Standards, whether *de facto* or *de jure*, can be created in different ways. Some are developed by an open group of companies, universities and individuals which work towards the standard under some national, e.g., ANSI (American National Standards Institute), or international, e.g., ISO, standardization body. Others are developed by similar groups, though working on the framework of non-official organizations such as the W3C (World Wide Web Consortium) or the IETF (Internet Engineering Task Force). Others still are developed by single institutions and their specification made public and accepted as *de facto* standards by the rest of the members of the market. Often *de facto* standards are later accepted as *de jure* standards by official standardization bodies.

In the world of multimedia, examples can be found in each of these cases. The video coding standards MPEG-1 and MPEG-2, and H.261 and H.263, were developed under international standard organizations, viz. ISO and ITU (International Telecommunication Union), and thus are *de jure* standards. The Java™ language, on the other hand, was developed by a single company, Sun Microsystems, and is being accepted quickly as a *de facto* standard (it has also been proposed to ISO to become a *de jure* standard). The Web standards, such as HTTP and HTML, are being developed in the framework of the IETF and W3C non-official organizations.

Convergence can also be found in the world of standards: the MPEG (Moving Picture Experts Group) community, traditionally video-oriented, and the WWW community, more multimedia oriented, are converging. The MPEG community is finalizing the first version of MPEG-4. MPEG-4 version 1 will be much more than video and audio coding with a multiplexing layer, as MPEG-1 and MPEG-2 were: MPEG-4 will standardize audio-visual 3D scene description methods, by inclusion of the ISO/IEC 14772 VRML (Virtual Reality Modeling Language) standard. The WWW community, on the other hand, is issuing documents, which will probably become *de facto* standards, that address similar subjects: PNG (Portable Network Graphics) for encoding of still images, support of VRML for 3D virtual worlds (which includes video nodes), and SMIL (Synchronized Multimedia Integration Language) for synchronizing different multimedia objects in a single presentation. More than a convergence, what is being witnessed is an overlap, a competition. The future will tell whether the minimalist, text-based, W3C and IETF standards or the overwhelming MPEG standards will win. Market does not always choose the best technology: often timing, as mentioned before, is the critical factor.

2.5.1 Standardization challenges

Nowadays standardization of multimedia communications faces several challenges. Different technologies (some of them standards), by different organizations, will address distinct subsets of the challenges listed below:

Content

Interesting contents will soon include complex 3D scenes, containing a mixture of synthetic and natural dynamic objects, which can be manipulated by the end user. Who will provide this type of information or content? How? I.e., using what tools?

Bandwidth

Network bandwidth and mass storage capacities both continue to grow exponentially. Even in the unlikely event that they will continue to increase exponentially forever, "technological malthusianism" tells us that the bandwidth/capacity will never be enough, since content will always grow at a faster pace. Hence, there will always be money to be gained, or spared, through compression of the multimedia data transmitted or stored.

The issue of compression has typically been much more of a concern for the video rather than the multimedia people. A number of standards, aiming at different applications, have been developed for the compression of video and still images: H.261 and H.263, MPEG-1, MPEG-2, MPEG-4 (soon to be born), and ISO/IEC JPEG (Joint Photographic Experts Group). From the multimedia world, less concerned, unfortunately, with bandwidth waste, little more than the W3C PNG exists today.

Access

How should the information be stored/transmitted to be easily accessible, and hence manipulated? This is a subject in which the multimedia people excel, but the video community has only recently started to address in a thorough way, in MPEG-4. There are good reasons for the late convergence: compression and easy access are quite incompatible, and for some time bandwidth was more important than interactivity. The balance is likely to change.

Classification

The Web is a huge, distributed database, whose size tends to increase exponentially. How can users navigate through this apparent chaos in a useful way? How can multimedia information such as text, 2D (Two-dimensional) pictures, 2D drawings, 2D videos, sound clips, movies, TV programs, 3D objects, and mixtures thereof, be indexed, searched, and retrieved in a meaningful way? Will the indexing, or classification, be done automatically?

This is an issue which is being simultaneously addressed by W3C and MPEG, through the recently born MPEG-7 effort. W3C is working on Metadata, or information about information, while MPEG-7 aims at standardizing multimedia indexing methods.

Rights protection

Providers of interesting content, individual authors or companies, will be interested in getting paid. How can IPR (Intellectual Property Rights) be protected on the Web? What will the network economics be like? How will IPR information be included on multimedia objects?

Access control and rating

Should all information on the Web be available to all? Who should control? How to control? How to rate information? How to cipher sensitive information?

W3C has addressed this question through a type of Metadata called PICS (Platform for Internet Content Selection), which aims at standardizing the method of including rating information (labels) into Web content.

Trust

Is the information available on the Web trustworthy? How to ascertain its real origin? How can information be certified? How can one assure that a signature certifies a given piece of information and that this information has not changed in any way?

W3C is also working on DSig (Digital Signatures), and there are some CEC (European Community Commission) funded projects working on watermarking of visual information.

Interworking

How to avoid needless duplication of hardware/software needed to access information of the same type stored in different formats? This is the basic objective of standardization efforts.

Evolution

How to produce standards that encourage, rather than prevent, competition and technical evolution?

MPEG-4 had the provision for evolution as one of its objectives. However, due to timing problems, MPEG-4 was divided into two phases. Phase 1, which is scheduled for the late 1998, will not provide for much evolution. Only phase 2 will include provision for programmable terminals, and hence allow, or even encourage, evolution.

2.5.2 Evolution of visual coding standards

Section 2.4.1 presented the various measures which can be used to construct the cost functional that video coders minimize (or at least attempt to minimize). Most of them have been used in one form or another by encoders compliant with the available video coding standards. However, easiness of access to content was first considered only in MPEG-1 and MPEG-2, in the form of provision for quick access to anchor images. These images, known as I images (I of Intra), are independently encoded and spread evenly in time, thus allowing the so-called trick modes of video recorders: fast-forward, backtrack, etc. This allowed only for a rather terse access to content. It was only MPEG-4 which started to consider a more useful form of content, objects, and which provided means for expressing complex 3D audio-visual scenes with mixtures of 2D and 3D objects, natural or synthetic. The real revolution was from MPEG-2 to MPEG-4. MPEG-2 was essentially a revamped version of MPEG-1, using the same basic tools, but allowing for increased resolution [95]: HDTV (High Definition Television) required it. True breakthroughs in the video coding area have been quite rare. Most of the tools used by encoders compliant to MPEG standards, even MPEG-4, are small variants, however well-engineered, of tools developed decades ago [149], e.g., DCT and block matching motion compensation. However, the integral of all the incremental hardware and software technology advances over the last decades corresponds to an impressive evolution.

2.5.3 Consequences of standardization

Standards don't specify encoders: they specify a bit stream syntax and a decoder. Hence, they implicitly define a model for the structural data to be encoded. In this sense, video coding standards can also be classified as belonging to one of the three generations presented before.

In a slightly more formal way, let B be the space of bit streams compliant with a given standard. Let E be the space of the encoders compliant with the same standard. Then, a given encoder $e(\cdot)$, in the broad sense, is a function from the space R, of scene representation, to B, i.e., $e(\cdot) : R \to B$. Space E is thus clearly limited by the nature of B. Standards specify decoders, that is, they specify a function $d(\cdot)$ from B back to R. Typically, space E, though restricted by the nature of B, is very large. Even if one restricts it to the space of compliant encoders providing appropriate reconstruction, that is, such that $d(e(\cdot))$ is approximately the identity, the space is too large.

One can pose the encoding problem mathematically, though the complexity of the solution usually leads to heuristic solutions: how to encode a given scene representation r? This question can be answered by finding $\arg\min_{b \in B} z(d(b), r)$, where z is a distortion measure. However, this includes only a distortion, or quality, measure. One may be interested in minimizing other measures. The generic problem is to find a generic encoder, i.e., an encoder leading to good decoding. A possibility is to find $\arg\min_{e \in E} \max_{r \in R} z(d(e(r)), r)$.

Whatever the approach taken, heuristic or optimizing, it is clear that standards introduce restrictions into the space of possible encoders. It also clear that they also leave a lot of room for competition, especially if error resilience is taken into account when designing encoders. Also, standards do define a decoder, but a decoder which operates only with error free encoded data. The design of decoders with good error concealment strategies and the design of encoders providing for good error resilience at the decoder is open to competition.

2.5.4 Standards and generations

Standards can be classified as first-, second- or third- generation, according to the character of the compliant encoders. However, nothing prevents the building of a second generation encoder (i.e., an encoder using mid-level analysis) which generates bit streams compliant with first-generation standards. For instance, MPEG-1 and MPEG-2 belong clearly to the first generation, while MPEG-4, which requires more sophisticated analysis tools but still uses a classical approach to encode the texture of the objects, can be said to be a step towards second-generation standards. Actually, this has been the typical road of evolution, as some of the work in this thesis demonstrates. When tools aimed at being used in one of these transition encoders are developed, one may classify them as belonging to transitions between generations.

2.6 Analysis and coding tools

Figure 2.2 shows the analysis, pre-processing and coding tools proposed or discussed in this thesis. The figure classifies these tools into the three generations, with two transition layers added. The tools are also listed below, together with pointers to the sections where they are described:

- Analysis tools:
  - Transition to second-generation:
    1. Knowledge-based segmentation [123, 125, 124] (Section 4.4).
    2. Camera movement estimation [129, 127, 130, 128, 113, 122] (Section 5.3).
  - Second-generation:
    1. RSST segmentation [32, 33] (Section 4.5).
    2. TR-RSST (Time-Recursive RSST) segmentation [119] (Section 4.7).
  - Transition to third-generation:
    1. RSST with human supervision [33] (Section 4.6).
- Pre-processing tools:
  - Transition to second-generation:
    1. Image stabilization [127, 130, 128, 113, 122] (Section 5.5).
- Coding tools:
  - Transition to second-generation:
    1. Camera movement compensation for improved prediction [129, 127, 130, 128, 113, 122] (Section 6.1).

– Second-generation:

1. Shape coding: a taxonomy and an overview of coding techniques [120, 121] (Sections 6.2 and 6.3), parametric curve coding tools [116, 114, 79, 78] (Section 6.4).

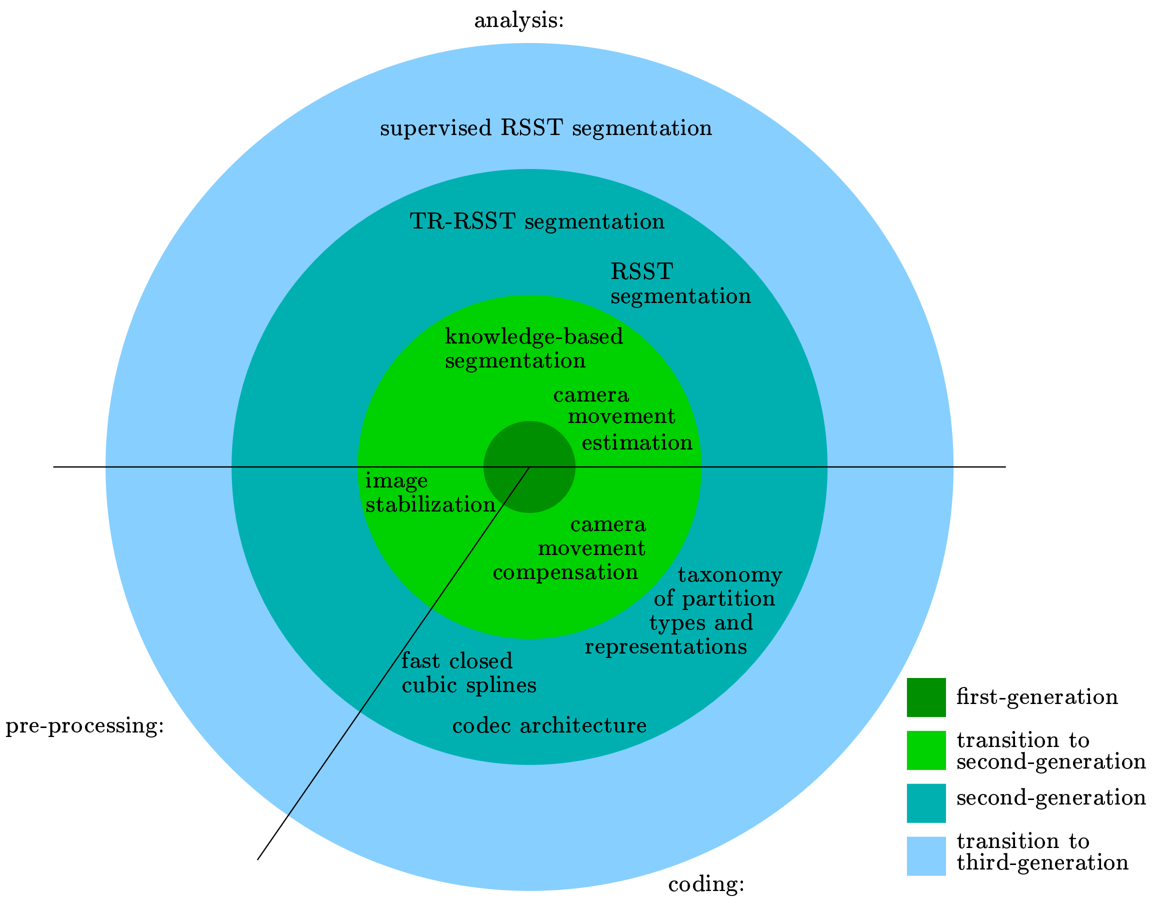

analysis:
supervised RSST segmentation
TR-RSST segmentation
RSST segmentation
knowledge-based segmentation
camera movement estimation
image stabilization
camera movement compensation
taxonomy of partition types and representations
fast closed cubic splines
codec architecture
pre-processing:
coding:
first-generation
transition to second-generation
second-generation
transition to third-generation

Figure 2.2: Analysis, pre-processing tools and coding tools proposed or discussed.

Chapter 3

Graph theoretic foundations for image analysis

Não devemos nunca procurar ser mais precisos
e exactos do que o problema em causa requer.

Karl Popper

This chapter defines the main concepts used throughout this thesis. It is divided into sections dealing with images, image lattices, image graphs, etc. Concepts are introduced, whenever possible, in a bottom-up manner: concepts are defined by using previously defined concepts.

Often the efficiency of algorithms known to solve problems related to the definitions given here is discussed: the usual $O(\cdot)$ notation of algorithmics is used [28].

3.1 Color perception

There are two types of light sensor cells in the retina: rods and cones. Rods are used for night (scotopic) vision, while cones are used for daylight (photopic) vision. Both are known to be used in twilight (mesopic) vision.

Rods greatly outnumber cones. However, the distribution of the rod cells is such that its density is nearly zero in the fovea, that is, the zone on the retina corresponding to the center of attention. In this zone cones are densely packed.[1] Rods are much more sensitive to light than cones: a single quantum is known to be sufficient to excite a rod. The different density distribution of rods and cones seems to be an evolutionary compromise between accuracy of

[1]A simple experiment confirms the absence of rods in the fovea. Look directly at a dim star and then look slightly to its side: its apparent lightness will increase.

vision (fundamental during daytime) and ability to detect threats (fundamental during dusk).

While rods are sensitive to a wide range of light frequencies, they all have the same type of response, hence scotopic vision is essentially "black and white": colors are not discriminated. Cones, on the other hand, are really three different types of cells with different frequency responses. One type of cones, say "red" cones, is especially sensitive to frequencies around pure red, another, "green" cones, to frequencies around pure green, and the last, "blue" cones, to frequencies around pure blue, where "pure" means consisting of single frequency. The overall response of the cones spans the visible light spectrum. However, the maximum sensitivity of the combined action of cones occurs at a slightly higher wavelength (towards red) than that of rods (towards blue): it is the so-called Purkinje wavelength shift. This seems to be related to the fact that during twilight light is more bluish than during daytime, since it is mostly indirect light diffracted by the atmosphere particles.

In the framework of image communications and multimedia, photopic (daytime) vision is the rule, so that the response of rods can be mostly ignored. The response of cones can be modeled as a nonlinear function of the inner product of a spectral sensitivity function, which is a characteristic of the given type of sensor cell in a Standard Observer, and the power spectrum of the light attaining the sensors (see for instance [189]). "Red", "green", and "blue" cones have different spectral sensitivity functions which partially overlap in frequency.

Further information on color perception may be found in [164, 1, 27].

3.1.1 Color spaces

Color reproduction uses the fact that the HVS has only three types of cones. In order for two light sources to be perceived as having equal color it is not necessary for their power spectra to be equal: they only have to produce the same response for each of the three types of cones. Hence, most image data is available in a three component format.

Color data is often presented in a CRT (Cathode-Ray Tube). Since the power emitted by such screens is typically proportional to a (arithmetic) power of the input voltage (the exponent being the so-called gamma value), cameras are usually designed to perform gamma correction. The correction specified by ITU-R (ITU Radiocommunication Sector)[2] Recommendation BT.709-2 [80] follows

$$I' = \begin{cases} 4.5I & \text{if } 0 \leq I \leq 0.018, \text{ and} \\ 1.099I^{0.45} - 0.099 & \text{if } 0.018 < I \leq 1, \end{cases} \tag{3.1}$$

which is the inverse of the ideal monitor power function

$$I = \begin{cases} \frac{I'}{4.5} & \text{if } 0 \leq I' \leq 0.081, \text{ and} \\ \left(\frac{I'+0.099}{1.099}\right)^{\frac{1}{0.45}} & \text{if } 0.081 < I' \leq 1, \end{cases}$$

[2]Formerly CCIR (Comitė Consultatif Internationale des Radio Communications).

where I is the light intensity and I' is the video signal, both linearly scaled so that they span the interval from zero to one (for the intensity range of interest).

It should be noted that the response of the HVS to intensity is approximately the inverse of the typical CRT nonlinearity [164]. Hence, since image analysis attempts to emulate the HVS, the nonlinear version of the color signals at the output of gamma corrected cameras can and should be used directly.

RGB

The power spectrum of the light input into the camera at each point in the image is transformed, by three different sensors, into a trio of values. The transformation can be modeled again as the inner product of the sensor spectral sensitivity function (which can actually depend on a set of filters) by the incident power spectrum. These three values, with appropriate sensors and possibly after a linear transformation, are R, G and B (for red, green, and blue): the linear RGB (Red, Green, and Blue) color space, where each value is linearly scaled to span the interval from zero to one. Each of the three linear RGB signals then undergoes the gamma correction nonlinear transformation in equation (3.1). The resulting color space is nonlinear RGB, or $R'G'B'$, where the prime stands for nonlinear.

Digital RGB video usually deals with a digitized version of the $R'G'B'$ color space. Since $R'G'B'$ has an analog unity excursion for each component, appropriate scaling and quantization must be used. According to the ITU-R Recommendation BT.601-2 [20], the $R'G'B'$ digital video color space components take integer values between 16 and 235 (excursion of 219), and thus are codable with eight bits,

$$\begin{aligned}
R'_{219} &= \text{round}(219R') + 16,\\
G'_{219} &= \text{round}(219G') + 16, \text{ and}\\
B'_{219} &= \text{round}(219B') + 16.
\end{aligned}$$

This color space will henceforth be known as $R'G'B'219$.

In digital computers, however, an excursion of 255 is often used, resulting in the $R'G'B'255$ color space, which leads to smaller quantization errors

$$\begin{aligned}
R'_{255} &= \text{round}(255R'),\\
G'_{255} &= \text{round}(255G'), \text{ and}\\
B'_{255} &= \text{round}(255B').
\end{aligned}$$

$Y'C_BC_R$

Luma is a signal which is related to brightness, and which is usually, though wrongly, referred to as luminance. Luminance, Y, is defined by CIE (Commission Internationale de l'Éclairage) as radiant power weighted by a spectral sensitivity function that is characteristic of vision [164]. It can be expressed as a weighted sum of the linear RGB components. Luma, Y', on the other hand, is a weighted sum of the non-linear, gamma corrected, $R'G'B'$components. Hence, Y' cannot be obtained from Y by gamma correction, as in (3.1).

In order to save bandwidth or storage space, video data is often provided in a format in which color is subsampled relative to luma. This owes to the fact that the HVS is less sensitive to spatial detail in color than in brightness. The corresponding color space separates color information from luma by subtracting luma from the R' and B' signals. The color signals, known as C_B and C_R, are then obtained by scaling, offsetting and quantizing. This color space, known as $Y'C_BC_R$, and the sampling format, are specified in ITU-R Recommendation BT.601-2 [20]. The components of $Y'C_BC_R$ can be computed from $R'G'B'$ by

$$\begin{aligned}
Y' &= \text{round}(16 + 65.481R' + 128.553G' + 24.966B'),\\
C_B' &= \text{round}(128 - 37.797R' - 74.203G' + 112.000B'), \text{ and}\\
C_R' &= \text{round}(128 + 112.000R' - 93.786G' - 18.214B'),
\end{aligned}$$

where Y' has an excursion from 16 to 235, and C_B and C_R have and excursion from 16 to 240 (with zero corresponding to level 128). Y' is usually known as the luma signal, and C_B and C_R are known as the chroma signals.

3.2 Images and sequences

3.2.1 Analog images

Definition 3.1. ([still] analog image) *A 2D function $f(\cdot) : \mathbf{R} \to \mathbb{R}^n$ defined in a bounded region $\mathbf{R} \subset \mathbb{R}^2$ and taking values in $\mathbb{R}^n$. It is assumed that the coordinate s_x in $s = \begin{bmatrix} s_x & s_y \end{bmatrix} \in \mathbb{R}^2$ grows rightwards and the coordinate s_y grows upwards.*

A still image usually corresponds to the projection of a 3D scene onto a 2D plane (e.g., the projection plane of a camera) at a given time instant.[3] The dimension n of the space where the function takes values is the number of color components of the color space used. Usual values of n are $n = 3$ for RGB, $R'G'B'$, or any other color spaces adapted to the trichromatic HVS, and $n = 1$ for grey scale images.

When time is allowed to flow, the previous definition must be extended to encompass moving images:

Definition 3.2. (moving analog image) *A 3D function $f(\cdot) : \mathbf{R} \times \mathbf{T} \to \mathbb{R}^n$ defined in a time interval $\mathbf{T} \subseteq \mathbb{R}$ and in a bounded space region $\mathbf{R} \subset \mathbb{R}^2$.*

[3]Actually, besides being the result of an integration over a short period of time, it is not a true projection, since the image is formed through a lens, and there is the question of focusing to consider.

3.2.2 Digital images

Still or moving analog images must be sampled and quantized, i.e., digitized, before they can be manipulated by digital computers. The result of sampling and quantizing is a digital image:

Definition 3.3. (image) *A 2D function $f[\cdot] : \mathbf{Z} \to \mathbb{Z}^n$ defined in a bounded discrete space region $\mathbf{Z} \subseteq \mathbb{Z}^2$ and taking values in $\mathbb{Z}^n$ (quantized component space).*

Definition 3.4. (moving image or video [or image] sequence) *A 3D function $f[\cdot] : \mathbf{N} \times \mathbf{Z} \to \mathbb{Z}^n$ defined in a discrete time interval $\mathbf{N} \subseteq \mathbb{Z}$ and in a bounded discrete space region $\mathbf{Z} \subset \mathbb{Z}^2$.*

Definition 3.5. (pixel) *Each of the elements of $v = \begin{bmatrix} v_i & v_j \end{bmatrix} \in \mathbf{Z}$ in the domain of a digital image. The name "pixel" stems from "picture element".*

In the case of moving images, pixels are extended with a, often implicit, time coordinate in the discrete time domain $\mathbf{N}$ of the image. Those 3D pixels are also known as voxels (from "volume element").

There is a special class of digital image or moving image which is often of interest: binary or black and white images. These images take values on a set with only two values, which can be integer values (usually 0 and 1, but often 0 and 255, for eight bits coding), or, for instance, the boolean values "false" and "true".

3.2.3 Lattices, sampling lattices and aspect ratio

Usually analog images are periodically sampled in space and time. The set of sample positions defines a sampling pattern, normally in the form of a lattice (a sampling lattice):

Definition 3.6. (lattice [181]) *Let u_j, with $j = 0, \ldots, m$, be a set of linearly independent vectors in $\mathbb{R}^m$ (the lattice basis). The set of sites $s \in \mathbb{R}^m$ such that $s = s[v] = \sum_{j=0}^{m-1} v_j u_j$, with $v = \begin{bmatrix} v_0 & \ldots & v_{m-1} \end{bmatrix} \in \mathbb{Z}^m$, is a lattice $\mathcal{L}$ in $\mathbb{R}^m$.*

A digital image $f[\cdot]$ can be obtained by sampling and quantizing an analog image $f(\cdot)$ according to a 2D sampling lattice

$$f[v] = q\big(\tilde{f}(s[v])\big) \text{ with } v \in \mathbf{Z},$$

where $\tilde{f}(\cdot)$ is an anti-aliased, filtered version of the analog original $f(\cdot)$, and $q(\cdot) : \mathbb{R}^n \to \mathbb{Z}^n$ is a quantization function. Notice that anti-alias filtering and quantization are usually performed independently on each color component.

Similarly, a digital sequence $f[\cdot]$ can be obtained by sampling and quantizing an analog sequence $f(\cdot)$ according to a 3D sampling lattice

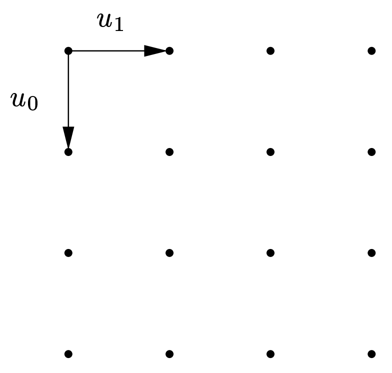
$$f_n[v] = f[n, v] = q\big(\tilde{f}(s[n, v], t[n, v])\big) \text{ with } n \in \mathbf{N} \text{ and } v \in \mathbf{Z},$$

where $\tilde{f}(\cdot)$ is an anti-aliased, filtered version of the analog original $f(\cdot)$, and $q(\cdot) : \mathbb{R}^n \to \mathbb{Z}^n$ is a quantization function. Notice that usually the anti-aliasing filter is separable between the space and time dimensions.

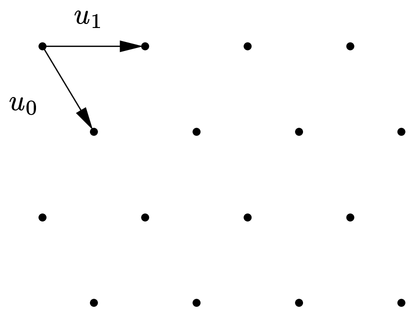
Hence, a sampling lattice relates the pixels with the corresponding positions in the original, analog image.

Usual 2D sampling lattices

The most common 2D sampling lattices are the rectangular, generated by $u_0 = \begin{bmatrix}0 & -b\end{bmatrix}^T$ and $u_1 = \begin{bmatrix}a & 0\end{bmatrix}^T$ (square when $a = b$), and the hexagonal, generated by $u_0 = \begin{bmatrix}a/2 & -a\sqrt{3}/2\end{bmatrix}^T$ and $u_1 = \begin{bmatrix}a & 0\end{bmatrix}^T$, as can be seen in Figure 3.1. The names for these lattices stem from the shape of the regions in a Voronoi tessellation corresponding to the lattice sites.[4] Often the term pixel is applied to these regions, instead of to the coordinates of the lattice sites. The meaning should be clear from the context. Notice that the lattice vectors invert the meanings of x and y (in the case of the rectangular lattice) and invert the direction of the y axis. This is to maintain the usual convention of thinking of a digital image f as a matrix with elements $f[i, j]$, where i grows downwards and j rightwards.



(a) Rectangular lattice.



(b) Hexagonal lattice

Figure 3.1: Examples of 2D lattices (u_0 and u_1 are the lattice basis vectors). Lattice sites are represented by dots.

In the case of rectangular lattices, $\alpha = a/b$ is the pixel aspect ratio. Even though the pixel aspect ratio rarely has the value 1 (corresponding to square pixels), this fact is often neglected. For instance, the soon to be issued MPEG-4 standard, the first in the MPEG family to specify a scene structure, and hence to mix video with computer graphics objects, seems to have

[4]Given a set of sites in space, the Voronoi tessellation surrounds each site with a region of influence corresponding to those points of the space which are closer to the given site than to any other site. If the space is $\mathbb{R}^2$ and the number of sites is finite (a sufficient, though not necessary, condition), the Voronoi tessellation will have polygonal, possibly unbounded regions. The dual concept is the Delaunay triangulation.

neglected this issue, at least in its first version.[5] The result is distorted renderings of video material whenever $\alpha \neq 1$.

A single lattice can be generated by different bases. The bases corresponding to the smallest possible vectors are said to be reduced [69]. The bases presented above for the rectangular and hexagonal lattices in $\mathbb{R}^2$ are reduced.

Usual 3D sampling lattices

In practice there are two sampling lattices used for moving, 3D images: progressive (or rectangular) and interlaced. The progressive lattice is generated by $u_0 = \begin{bmatrix}0 & 0 & \tau\end{bmatrix}^T$, $u_1 = \begin{bmatrix}0 & -b & 0\end{bmatrix}^T$ and $u_2 = \begin{bmatrix}a & 0 & 0\end{bmatrix}^T$ (spatially square when $a = b$). The interlaced lattice is generated by $u_0 = \begin{bmatrix}0 & -b/2 & \tau\end{bmatrix}^T$, $u_1 = \begin{bmatrix}0 & -b & 0\end{bmatrix}^T$ and $u_2 = \begin{bmatrix}a & 0 & 0\end{bmatrix}^T$ (spatially square when $a = b$). In both cases τ is the sampling time period. The interlaced scanning used in analog cameras introduces a time delay between the first and the last lines in a field. The digitized version of an analog interlaced signal thus has a more complex structure than the one presented here.

Notice that the lattice vectors, in the case of the rectangular lattice, again invert the meanings of x, y, and now also z (time), and invert the direction of the y axis. This is to maintain the usual convention of thinking of a digital moving image f as a sequence f_n, with $n \in \mathbf{N}$, of matrices with elements $f_n[i, j]$, where i grows downwards and j rightwards. Using the notation above, $f_n[i, j] = f[n, i, j]$.

This notation is usually violated in the case of interlaced 3D lattices. This is because the spatial domain of moving analog images is usually a rectangle in a fixed location. Hence, it is common to change the meaning of i in the notation $f_n[i, j]$ into: The number of the row, assuming that at each time instant row zero is the first row inside the given rectangular spatial domain.

In this thesis all moving images are assumed to have been sampled using a progressive lattice. Methods for converting digital images from interlaced to progressive sampling lattices are easily constructed, but outside the scope of this thesis.

3.3 Grids, graphs, and trees

This section presents some definitions and results regarding grids and graphs. Since some of the material on graphs can be found in good monographes on graph theory, such as [186], several results are presented informally and without proof. A good reference for engineering applications of graph theory is [23].

Definition 3.7. (grid [181]**)** *A grid $\mathcal{G}(\mathbf{V}, \mathbf{E})$ in $\mathbb{Z}^m$ is defined by two sets:*

1. $\mathbf{V} \subseteq \mathbb{Z}^m$ *– the set of vertices.*

[5]Actually the aspect ratio can now be specified in the VOL (Video Object Layer). A fortunate last-minute addition to the standard.

2. *$\mathbf{E}$ – a set of unordered pairs of points $\{v_a, v_b\}$ with $v_a \neq v_b \in \mathbf{V}$, i.e., a neighborhood system or the set of the edges in the grid.*

such that (see [181] for further details):

1. *the sets $\mathbf{V}$ and $\mathbf{E}$ are invariant with respect to some sub-group t of translations in $\mathbb{Z}^m$ (t-invariance condition).*
2. *the edges (elements of $\mathbf{E}$) do not cross (non-crossing condition).*

Figure 3.2 shows a schematic representation of the usual square and hexagonal grids in $\mathbb{Z}^2$, both with $\mathbf{V} = \mathbb{Z}^2$. The names for the grids are related to the number of neighbors of each vertex. The geometrical representation with squares is a mere matter of convenience.

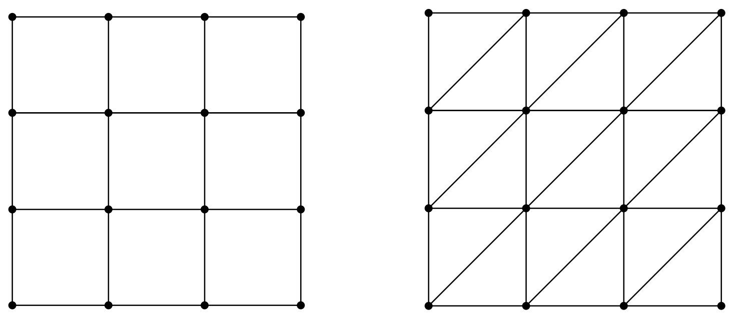

(a) Rectangular grid. (b) Hexagonal grid.

Figure 3.2: Examples of grids. Vertices are represented by dots and edges by lines.

A grid can be seen as a simple graph. Additionally, because of the non-crossing condition, grids in $\mathbb{Z}^2$ can be seen as planar simple graphs. Edges in grids correspond to arcs in graphs.

Definition 3.8. (simple graph [172]**)** *A simple (undirected) graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ consists of a nonempty set $\mathbf{V}$ of vertices and a set $\mathbf{A}$ of unordered pairs of distinct elements of $\mathbf{V}$ called arcs. Since $\mathbf{A}$ is a set, there are no repeated arcs. Since the arcs consist of unordered pairs of distinct elements, no arc connects a vertex to itself.*

Often it may be important to allow the existence of multiple or parallel arcs. Since simple graphs disallow them, a more generic definition may be needed:

Definition 3.9. (multigraph [172]**)** *A multigraph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ consists of a nonempty set $\mathbf{V}$ of vertices, a set $\mathbf{A}$ of arcs, and an arc function $g(\cdot) : \mathbf{A} \to \big\{\{u, v\} : u \neq v \in \mathbf{V}\big\}$. Two arcs a_1 and a_2 are multiple or parallel if $g(a_1) = g(a_2)$. Hence, if $g(\cdot)$ is injective, then the multigraph has no parallel arcs, and hence is a simple graph. By a slight abuse of notation, $\{u, v\} \in \mathbf{A}$ will be taken to mean $\exists a \in \mathbf{A} : g(a) = \{u, v\}$.*

Still another generalization allows the existence of arcs connecting a vertex to itself:

Definition 3.10. (pseudograph [172]**)** *A pseudograph $\mathcal{G}(\mathbf{V},\mathbf{A})$ consists of a nonempty set $\mathbf{V}$ of vertices, a set $\mathbf{A}$ of arcs, and an arc function $g(\cdot) : \mathbf{A} \to \big\{\{u,v\} : u,v \in \mathbf{V}\big\}$. The pseudograph can thus contain arcs a such that $g(a) = \{u\}$, i.e., arcs between a vertex and itself. A pseudograph without such self-connecting arcs is, of course, a multigraph.*

Properties which are true for pseudographs are also true for multigraphs. And properties which are true for multigraphs are also true for simple graphs. Hence, throughout this thesis, the word graph will be taken to mean pseudograph unless it is qualified with "multi-" or "simple". For simple graphs, the function $g(\cdot)$, which is injective, is also taken to exist.

Definition 3.11. (simplification) *The process of successively eliminating (or merging) multiple arcs from a multigraph until a simple graph is obtained. In the case of pseudograph, this process is preceded by the elimination of self-connecting arcs. Hence, simplification converts a pseudo- or multigraph into a simple graph.*

Definition 3.12. (planar graph [172]**)** *A graph is planar if it can be drawn in the plane without crossing arcs.*

A multigraph is planar if its simplification is planar. The same is true for a pseudograph. Planar graphs are especially important for image processing. Hence, results concerning planar graphs are given in a separate section.

Definition 3.13. (vertex adjacency or neighborhood and degree [172]**)** *Two vertices $u,v \in \mathbf{V}$ of a graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ are called adjacent or neighbor vertices if there is an arc a such that $g(a) = \{u,v\} \in \mathbf{A}$. In that case, a is said be incident with (or to connect) vertices u and v, and u and v are called the end vertices of a. The degree $d(v)$ of a vertex v is simply the number of arcs incident with it. The degree $d(u,v)$ of a pair of vertices u and v is the number of arcs incident on both vertices, i.e., $d(u,v) = \#\big\{a \in \mathbf{A} : g(a) = \{u,v\}\big\}$.*[6]

Hence, $d(u,v)$ is either zero or one in a simple graph and $d(u,u)$ is always zero on multi- or simple graphs, i.e., on graphs without self-connecting arcs. Also, the relations $d(u,v) \le d(u)$ and $d(u,v) \le d(v)$ always hold. Additionally, it can be easily proven that $\sum_{v\in\mathbf{V}} d(v) = 2\#\mathbf{A}$ for any simple, multi-, or pseudograph.

3.3.1 Graph operations

Basic graph operations

Given a graph $\mathcal{G}(\mathbf{V},\mathbf{A})$, then:

- If $a \in \mathbf{A}$, then $\mathcal{G} - a$ means the graph $\mathcal{G}(\mathbf{V},\mathbf{A} \setminus \{a\})$.
- If a is such that $g(a) = \{u,v\}$ with $u,v \in \mathbf{V}$, then $\mathcal{G} + a$ means the graph $\mathcal{G}(\mathbf{V},\mathbf{A} \cup \{a\})$.

[6] $\#\mathbf{S}$ is the number of elements, i.e., the cardinality, of the set $\mathbf{S}$.

- If $v \in \mathbf{V}$, then $\mathcal{G} - v$ means the graph $\mathcal{G}(\mathbf{V} \setminus \{v\}, \mathbf{A} \setminus \mathbf{A}')$, where $\mathbf{A}' = \{a \in \mathbf{A} : v \in g(a)\}$, the set of arcs incident on v.
- $\mathcal{G} + v$ means the graph $\mathcal{G}(\mathbf{V} \cup \{v\}, \mathbf{A})$.

Subdivisions, mergings, short-circuits, and reductions

Definition 3.14. (arc subdivision [172]**)** *A transformation of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ to graph $\mathcal{G}'(\mathbf{V}', \mathbf{A}')$ consisting in removing an arc $a \in \mathbf{A}$ incident on $u, v \in \mathbf{V}$ and inserting a new vertex w and two arcs b and c such that $g(b) = \{u, w\}$ and $g(c) = \{v, w\}$. That is, the process of dividing an arc into a sequence of two arcs. The resulting graph has $\mathbf{A}' = \mathbf{A} \setminus \{a\} \cup \{c\} \cup \{d\}$ and $\mathbf{V}' = \mathbf{V} \cup \{w\}$.*

Definition 3.15. (arc contraction or merging of adjacent vertices [186]**)** *A transformation of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ to a graph $\mathcal{G}'(\mathbf{V}', \mathbf{A}')$ such that an arc a with $g(a) = \{u, v\}$ is removed and the two adjacent vertices $u, v \in \mathbf{V}$ (which may be the same vertex) are replaced by a single new vertex $w \in \mathbf{V}'$. Arcs with end vertices u or v are changed so that they are incident on w.*

In the case of multi- or simple graphs, all arcs a such that $g(a) = \{u, v\}$ are removed in the transformation, for otherwise self-connecting arcs would be introduced. In the case of simple graphs, pairs of arcs between some vertex $x \neq u, v \in \mathbf{V}$ and u and v are merged into a single arc, so that no multiple arcs are introduced.

A related concept is that of short-circuiting:

Definition 3.16. (vertex short-circuiting) *A transformation of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ to a graph $\mathcal{G}'(\mathbf{V}', \mathbf{A}')$ such that two arbitrary vertices $u, v \in \mathbf{V}$ are replaced by a new vertex $w \in \mathbf{V}'$. Arcs with end vertices u or v are changed so that they are incident on w.*

The same considerations as for vertex mergings apply with respect to multi- and simple graphs.

Definition 3.17. (contraction) *A graph $\mathcal{G}'$ that can be obtained by a sequence of arc contractions performed on graph $\mathcal{G}$ is called a contraction of $\mathcal{G}$. The contraction $\mathcal{G}'$ is maximal if it contains no arcs.*

The inverse of an arc subdivision is often of interest. It will, however, be defined only for pseudographs, since for pseudographs it does maintain the graph circuits (and hence is invariant with respect to homeomorphism, see Definitions 3.23 and 3.43):

Definition 3.18. (arc reduction) *A transformation of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ to pseudograph $\mathcal{G}'(\mathbf{V}', \mathbf{A}')$ consisting in removing a vertex $v \in \mathbf{V}$ with $d(v) = 2$ and such that $\exists a, b \in \mathbf{A}$ with $a \neq b$ incident on v. I.e., the (two) arcs incident on v are not self-connecting (otherwise $d(v) = 4$). Let $g(a) = \{v, u\}$ and $g(b) = \{v, w\}$ ($v \neq u, w$, of course). The removal of v is then followed by the removal of arcs a and b and by the insertion of a new arc c such that $g(c) = \{u, w\}$ (which may be a self-connecting arc).*

Thus, an arc reduction can be seen as an arc contraction performed on one of the two distinct arcs connecting to the chosen vertex, which must be of degree two. Unlike arc contractions, however, it is easy to prove that arc reductions do not introduce nor remove circuits (see Definition 3.23) from the graph.

Definition 3.19. (reduction) *A graph $\mathcal{G}'$ that can be obtained by a sequence of arc reductions performed on graph $\mathcal{G}$ is called a reduction of $\mathcal{G}$. The reduction $\mathcal{G}'$ is maximal if no further arc reductions are possible.*

The vertices that remain after the maximal reduction of a graph depend in general on the order of the arc reductions. However, the resulting maximal reductions, while possibly different, are always isomorphic (see Definition 3.42).

3.3.2 Walks, trails, paths, circuits, and connectivity in graphs

Definition 3.20. (walk [186]**)** *A finite alternating sequence of vertices and arcs $v_0, a_1, v_1, \ldots, v_{k-1}, a_k, v_k$, with $v_i \in \mathbf{V}$ for $i = 0, \ldots, k$ and $a_i \in \mathbf{A}$ such that $g(a_i) = \{v_{i-1}, v_i\}$ for $i = 1, \ldots, k$, of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$, is a walk of length k between its end or terminal vertices v_0 and v_k (the other vertices are internal vertices). If $v_0 = v_k$ the walk is closed; otherwise it is an open walk. A walk with end vertices v_0 and v_k is called a v_0, v_k-walk.*

Definition 3.21. (trail [186]**)** *A trail is a walk with all arcs distinct. It is an open trail if its end vertices are distinct; otherwise it is closed.*

Definition 3.22. (path [186]**)** *A path is an open trail with all vertices distinct. A path with end vertices v_0 and v_k is called a v_0, v_k-path.*

It can be proved easily that all open walks or trails contain a path between their end vertices.

Definition 3.23. (circuit [186]**)** *A circuit is a closed trail with all vertices distinct except the end vertices.*

The shortest circuit in a pseudograph has length one, in a multigraph has length two, and in a simple graph has length three.

Since paths and circuits have no repeated arcs or vertices (except the end vertices of a circuit), both can be specified uniquely from their set of arcs (in the case of a circuit the terminal vertex will remain ambiguous, though). Hence, if $\mathbf{P}$ is a path and $\mathbf{C}$ is a circuit, then $\mathbf{P}$ and $\mathbf{C}$ will often be interpreted as the set of arcs in the path and the set of arcs in the circuit, respectively. It should be clear that all vertices in a circuit have degree 2 on that circuit, the same thing happening to the all vertices in a path except the end vertices, which have degree 1. Also, even though paths of length zero are precluded by the definition of path, they will sometimes be of use, in which case they will be assumed to consist of a single vertex and represented by an empty set of arcs (in this case the representation by the arcs is not sufficient, but this is rarely problematic).

A u, v-path is in general not unique in a graph. It is unique, if it exists, only in the case of acyclic graphs (graph with no circuits, see below). In these cases of uniqueness, it makes sense to write $\mathbf{P} = u, v$-path.

Definition 3.24. (circuit arc [186]**)** *An arc a is a circuit arc of graph $\mathcal{G}$ if there is a circuit in $\mathcal{G}$ which includes a.*

Definition 3.25. (connectivity) *Two vertices $u, v \in \mathbf{V}$ of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ are connected if there is a u, v-path (or walk or trail) in the graph. A subset $\mathbf{V}'$ of vertices of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ is connected if for all pairs of vertices $u, v \in \mathbf{V}'$ there is a u, v-path (or walk or trail) containing only vertices of $\mathbf{V}'$. A single vertex is connected by definition. A graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ is said to be connected if $\mathbf{V}$ is itself connected, i.e., if there is a path (or walk or trail) between any pair of vertices.*

It can be easily proved that the removal of a circuit arc from a connected graph leads to a graph which is still connected. If there are no circuit arcs in a graph, then the graph has no circuits and is said to be an acyclic graph.

Definition 3.26. (n-arc-connectivity) *A connected graph is n-arc-connected if at least n arcs must be removed to disconnect it.*

If all arcs in a graph are circuit arcs, then clearly that graph is 2-arc-connected.

Definition 3.27. (bridge) *An arc in a graph $\mathcal{G}$ is a bridge if its removal from $\mathcal{G}$ augments the number of connected components of $\mathcal{G}$ (see Definition 3.34).*

Notice that if a graph has a bridge, then it is 1-arc-connected, and that the number of connected components always augments by one when a bridge is removed.

3.3.3 Euler trails and graphs

Definition 3.28. (Euler trails) *An Euler trail is a closed trail containing all the arcs of a given graph. An open Euler trail is an open trail containing all the arcs of a given graph.*

Definition 3.29. (Euler graph) *A graph with all vertices of even degree is an Euler graph.*

It is known from graph theory [174] that a connected graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ has an Euler trail iff (if and only if) $\forall v \in \mathbf{V}$ $d(v)$ is even, i.e., iff it is an Euler graph. Also, it has an open Euler trail, but not a (closed) Euler trail, iff there are exactly two vertices with odd degree.

Another interesting result is that Euler graphs, connected or not, can be expressed, except for isolated vertices, as the union of arc-disjoint circuits.

A more general definition is:

Definition 3.30. (postman walk) *A postman walk is a closed walk containing all the arcs of a given graph. An open postman walk is an open walk containing all the arcs of a given graph.*

The Chinese postman problem

Let $w(\cdot) : \mathbf{A} \to \mathbb{R}_0^+$ be a weight function defined on the arcs $\mathbf{A}$ of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$. The Chinese postman problem is then to find a postman walk $v_0, a_1, v_1, \ldots, v_{k-1}, a_k, v_k$ such that $\sum_{i=1}^{k} w(a_i)$ is minimum (the path length k is $k >= \#\mathbf{A}$). Of course, if an Euler trail exists, it is also a solution of the Chinese postman problem.

The Chinese postman problem is solvable in polynomial time (i.e., it is not NP-complete) in the case of undirected, simple graphs. See [186] and [51] for details.

3.3.4 Subgraphs, complements, and connected components

Definition 3.31. (subgraph [174]**)** *A graph $\mathcal{G}(\mathbf{V}', \mathbf{A}')$ is a subgraph of $\mathcal{G}(\mathbf{V}, \mathbf{A})$ if $\mathbf{V}' \subseteq \mathbf{V}$ and $\mathbf{A}' \subseteq \mathbf{A}$. It is a proper subgraph if either $\mathbf{V}' \subset \mathbf{V}$ or $\mathbf{A}' \subset \mathbf{A}$.*

The concept of maximal subgraph is also important, and will be useful for defining connected components:

Definition 3.32. (maximal subgraph [172]**)** *A subgraph $\mathcal{G}(\mathbf{V}', \mathbf{A}')$ of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ is maximal if there is no arc $\{v_a, v_b\} \in \mathbf{A} \setminus \mathbf{A}'$ such that $v_a, v_b \in \mathbf{V}'$, that is, if all arcs in $\mathbf{A}$ connecting vertices of $\mathbf{V}'$ also belong to $\mathbf{A}'$. The maximal subgraph $\mathcal{G}(\mathbf{V}', \mathbf{A}')$ is said to be vertex-induced by $\mathbf{V}' \subseteq \mathbf{V}$.*

Hence, each subset $\mathbf{V}'$ of vertices from $\mathbf{V}$ induces a single maximal subgraph $\mathcal{G}(\mathbf{V}', \mathbf{A}')$ of $\mathcal{G}(\mathbf{V}, \mathbf{A})$, which can be constructed by including in $\mathbf{A}'$ all arcs with both end vertices in $\mathbf{V}'$.

A maximal subgraph can also be defined as a subgraph to which no further arcs of the original graph can be added without also adding some vertices.

Notice that a subgraph $\mathcal{G}(\mathbf{V}', \mathbf{A}')$ of $\mathcal{G}(\mathbf{V}, \mathbf{A})$ may also be arc-induced by some $\mathbf{A}' \subseteq \mathbf{A}$. The set of vertices $\mathbf{V}'$, in this case, is such that it contains only end vertices of arcs in $\mathbf{A}'$.

Definition 3.33. (complement [23]**)** *The complement $\overline{\mathcal{G}'}(\mathbf{V}'', \mathbf{A}'')$ of a subgraph $\mathcal{G}'(\mathbf{V}', \mathbf{A}')$ of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ is the subgraph arc-induced by $\mathbf{A}'' = \mathbf{A} \setminus \mathbf{A}'$ plus the vertices in $\mathbf{V} \setminus \mathbf{V}'$.*

Hence, a subgraph and its complement never have common arcs, but may have common vertices.

Definition 3.34. (connected component) *A maximal subgraph $\mathcal{G}(\mathbf{V}', \mathbf{A}')$ of $\mathcal{G}(\mathbf{V}, \mathbf{A})$ is a connected component of $\mathcal{G}(\mathbf{V}, \mathbf{A})$ if:*

$$\begin{array}{l} \mathbf{V}' \text{ is connected, and} \\ \forall \{u, v\} \in \mathbf{A} \text{ either } u, v \in \mathbf{V}' \text{ or } u, v \in \mathbf{V} \setminus \mathbf{V}'. \end{array}$$

Algorithms

The connected components of a graph can be identified using the DFS (Depth First Search) algorithm, whose running time for a generic graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ is $O(\#\mathbf{V} + \#\mathbf{A})$ [28]. If the graph is simple and planar, then the running time is also $O(\#\mathbf{V})$, i.e., it is linear on the number of vertices, see Section 3.4.1.

Even though this is not appropriate to an algorithm, the number of connected components of a graph can be seen to be equal to the number of vertices in its maximal contraction.

3.3.5 Rank and nullity

Definition 3.35. **(rank)** *The rank $\rho(\mathcal{G})$ of a graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ is defined as $\rho(\mathcal{G}) = \#\mathbf{V} - c$, where c is the number of connected components of $\mathcal{G}$.*

Definition 3.36. **(nullity)** *The nullity $\mu(\mathcal{G})$ of a graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ is defined as $\mu(\mathcal{G}) = \#\mathbf{A} - \#\mathbf{V} + c = \#\mathbf{A} - \rho(G)$, where c is the number of connected components of $\mathcal{G}$.*

3.3.6 Cut vertices, separability, and blocks

Definition 3.37. **(cut vertex** [23]**)** *If in a connected graph $\mathcal{G}$ there is a proper subgraph $\mathcal{G}'$, with at least one arc, such that it has only one vertex v in common with its complement $\overline{\mathcal{G}'}$, then vertex v is said to be a cut vertex of $\mathcal{G}$. A vertex of an unconnected subgraph is a cut vertex if it is a cut vertex of one of its connected components.*

If the removal of a vertex and all incident arcs leads to an increase (of at least one) in the number of connected components, then that vertex is a cut vertex. For multi- and simple graphs, the converse is also true. For pseudographs, vertices which are self-connected and which belong to a connected component with at least another vertex are also cut vertices regardless of whether their removal increases the number of connected components.

Definition 3.38. **(separability** [23]**)** *A disconnected graph is separable. A connected graph is separable if it contains cut vertices.*

Definition 3.39. **(block** [23]**)** *A block of graph $\mathcal{G}$ is a non-separable subgraph of $\mathcal{G}$ such that adding any further arc will make it separable.*

A graph can always be decomposed into its blocks. Notice that isolated vertices (i.e., vertices without incident arcs, not even self-connecting arcs), are blocks, even though they cannot be obtained by decomposition of any graph into blocks. The blocks of a graph are always 2-arc-connected, except for the trivial block with two vertices connected by a single arc, as can be proved easily.

3.3.7 Cuts and cutsets

Definition 3.40. **(cut)** *Given two proper subsets of vertices $\mathbf{V}_1, \mathbf{V}_2 \subset \mathbf{V}$ of a graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ such that $\mathbf{V}_1 \cup \mathbf{V}_2 = \mathbf{V}$ and $\mathbf{V}_1 \cap \mathbf{V}_2 = \emptyset$, the cut $\langle \mathbf{V}_1, \mathbf{V}_2 \rangle$ is the set of arcs $\big\{a \in \mathbf{A} : g(a) = \{u, v\}$ with $u \in \mathbf{V}_1, v \in \mathbf{V}_2\big\}$.*

Definition 3.41. (cutset) *A minimal set of arcs of graph $\mathcal{G}$ such that its removal increases the number of connected components (of exactly one).*

Hence, a cut $\langle \mathbf{V}_1, \mathbf{V}_2 \rangle$ of a connected graph $\mathcal{G}$ is also a cutset if $\mathbf{V}_1$ and $\mathbf{V}_2$ are connected. In general, it can be proved that cuts are either cutsets or unions of arc-disjoint cutsets.

3.3.8 Isomorphism, 2-isomorphism, and homeomorphism

An important concept, which tells when two graphs can be seen as "equivalent", is that of isomorphism:

Definition 3.42. (isomorphic graphs) *Two graphs $\mathcal{G}_1(\mathbf{V}_1, \mathbf{A}_1)$ and $\mathcal{G}_2(\mathbf{V}_2, \mathbf{A}_2)$ are isomorphic if there is a bijective function $g(\cdot) : \mathbf{V}_1 \to \mathbf{V}_2$ such that $d(u,v) = d(g(u), g(v))$ for all $u, v \in \mathbf{V}_1$.*

The concept of homeomorphism will be useful for defining a few concepts related to the borders in 2D maps:

Definition 3.43. (homeomorphic graphs) *Two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are homeomorphic if both can be obtained by successive arc subdivisions performed on a pair of isomorphic originating graphs. Or, if the their maximal reductions are isomorphic.*

Finally, the concept of 2-isomorphism will be of paramount importance when dealing with graph duality:

Definition 3.44. (2-isomorphism [186, 23]**)** *Two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are 2-isomorphic if they can be made isomorphic to one another by performing series of the following operations on any of them:*

1. *Decompose a separable graph into disconnected blocks and reconnect them successively by short-circuiting pairs vertices belonging to different connected components.*
2. *If $\mathcal{G}'$ is a proper subgraph of either $\mathcal{G}_1$ or $\mathcal{G}_2$, with at least one arc, such that it has only two distinct vertices u and v in common with its complement $\overline{\mathcal{G}'}$, decompose the original graph ($\mathcal{G}_1$ or $\mathcal{G}_2$) into $\mathcal{G}'$ and $\overline{\mathcal{G}'}$ by splitting the vertices u and v and then reconnect the same vertices after turning around one of the subgraphs.*

Step 1 above does not introduce nor remove circuits from the original graph: the short-circuited vertices thus become cut vertices, except if one of the components being connected is an isolated vertex (in which case it simply "disappears"). Step 2 also does not introduce nor remove circuits from the original graph. The importance of these facts can be seen from the theorem stating that two graphs are 2-isomorphic iff there is a bijective correspondence between their sets of arcs such that circuits in one graph correspond to circuits in the other [186], though the order of the arcs in the circuits may be different.

Step 1 does not change the number of arcs in the graph, and, when a block is separated by splitting a cut vertex or two connected components united by short-circuiting two vertices, both number of vertices and number of connected components either increase or decrease by one. Step 2 does not change the number of vertices, arcs and connected components of the graph. Hence, neither the rank nor the nullity of the graph change. As a consequence, 2-isomorphic graphs must have the same rank, the same nullity, and the same number of arcs.

3.3.9 Trees and forests

Theorem 3.1. (tree) *The following statements about graph $\mathcal{T}(\mathbf{V}, \mathbf{A})$ are equivalent:*

1. *$\mathcal{T}$ is a tree.*
2. *$\mathcal{T}$ is connected and acyclic.*
3. *$\mathcal{T}$ is connected and $\#\mathbf{A} = \#\mathbf{V} - 1 = \rho(\mathcal{T})$.*
4. *There is a unique path in $\mathcal{T}$ between any pair of distinct vertices.*
5. *Adding a new arc (incident on existing vertices) to $\mathcal{T}$ will create a single circuit.*

Theorem 3.2. (forest) *The following statements about graph $\mathcal{F}(\mathbf{V}, \mathbf{A})$ are equivalent:*

1. *$\mathcal{F}$ is a forest.*
2. *$\mathcal{F}$ is acyclic.*
3. *If there is a path in $\mathcal{F}$ between a pair of distinct vertices, then that path is unique.*
4. *$\mathcal{F}$ has c connected components and $\#\mathbf{A} = \#\mathbf{V} - c = \rho(\mathcal{F})$.*
5. *Adding a new arc (incident on existing vertices) to $\mathcal{F}$ will either reduce the number of connected components by one or create a single circuit.*

Theorem 3.3. (k-tree) *The following statements about graph ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A})$ are equivalent:*

1. *${}^k\mathcal{T}$ is a k-tree.*
2. *${}^k\mathcal{T}$ is a forest with k connected components (trees).*
3. *${}^k\mathcal{T}$ is acyclic and has k connected components.*
4. *${}^k\mathcal{T}$ has $\#\mathbf{A} = \#\mathbf{V} - k = \rho({}^k\mathcal{T})$.*
5. *Adding a new arc (incident on existing vertices) to ${}^k\mathcal{T}$ will either reduce the number of connected components by one or create a single circuit.*

Each connected component of a forest or a k-tree is a tree. A forest with a single connected component is a tree. A 1-tree is a tree.

If a tree $\mathcal{T}$, a k-tree ${}^k\mathcal{T}$ or a forest $\mathcal{F}$ are subgraphs of some graph $\mathcal{G}$, the phrases "$\mathcal{T}$ is a tree of $\mathcal{G}$", "${}^k\mathcal{T}$ is a k-tree of $\mathcal{G}$", "$\mathcal{F}$ is a forest of $\mathcal{G}$", will be used.

For a k-tree ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A})$, ${}^k\mathcal{T}_i(\mathbf{V}_i, \mathbf{A}_i)$, with $i = 1, \ldots, k$, refer to each of its composing trees (connected components). Obviously, $\cup_{i=1}^k \mathbf{V}_i = \mathbf{V}$ while $\mathbf{V}_i \cap \mathbf{V}_j = \emptyset$ for $i \neq j$, and $\cup_{i=1}^k \mathbf{A}_i = \mathbf{A}$ while $\mathbf{A}_i \cap \mathbf{A}_j = \emptyset$ for $i \neq j$.

Spanning trees and forests

Definition 3.45. (spanning tree) *A subgraph $\mathcal{T}(\mathbf{V}', \mathbf{A}')$ of simple graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ is a spanning tree of $\mathcal{G}$ if $\mathcal{T}$ is a tree and it covers $\mathcal{G}$, i.e., $\mathbf{V}' = \mathbf{V}$.*

A connected graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ can be covered by a spanning tree $\mathcal{T}(\mathbf{V}, \mathbf{A}')$ with $\#\mathbf{A}' = \#\mathbf{V} - 1 = \rho(\mathcal{G})$. The converse is true: if $\mathcal{T}(\mathbf{V}, \mathbf{A}')$ is a tree which is a subgraph of $\mathcal{G}(\mathbf{V}, \mathbf{A})$, then $\mathcal{T}$ is a spanning tree iff $\#\mathbf{A}' = \#\mathbf{V} - 1 = \rho(G)$.

Definition 3.46. (spanning forest) *Let $\mathcal{G}$ be a graph with c connected components. Let $\mathcal{F}$ be a forest which is also a subgraph of $\mathcal{G}$. If $\mathcal{F}$ consists of a union of spanning trees for all the c connected components of $\mathcal{G}$, then $\mathcal{F}$ is a spanning forest of $\mathcal{G}$.*

A spanning forest of a connected graph is a spanning tree. All spanning forests $\mathcal{F}(\mathbf{V}, \mathbf{A}')$ of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ with c connected components have $\#\mathbf{A}' = \#\mathbf{V} - c = \rho(\mathcal{G})$. The converse is also true: if $\mathcal{F}(\mathbf{V}, \mathbf{A}')$ is a forest which is a subgraph of $\mathcal{G}(\mathbf{V}, \mathbf{A})$, then $\mathcal{F}$ is a spanning forest iff $\#\mathbf{A}' = \#\mathbf{V} - c = \rho(\mathcal{G})$, where c is the number of connected components of $\mathcal{G}$.

Definition 3.47. (cospanning forest and tree) *Let $\mathcal{F}(\mathbf{V}, \mathbf{A}')$ be a spanning forest of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$. Then the subgraph $\mathcal{F}'(\mathbf{V}, \mathbf{A} \setminus \mathbf{A}')$ will be called the cospanning forest of forest $\mathcal{F}$. If $\mathcal{G}$ is connected, $\mathcal{F}$ is really a spanning tree and $\mathcal{F}'$ will be called its cospanning tree.*

Branches and chords

Definition 3.48. (branch and chord) *The arcs in a spanning forest are called branches of the spanning forest. The arcs in the corresponding cospanning forest are called chords of the spanning forest.*

Notice that self-connecting arcs cannot be part of any spanning forest. Hence, self-connecting arcs are always chords. Also notice that bridges must be part of all spanning forests, and hence they are always branches.

Fundamental circuits and cutsets

Spanning trees have some interesting properties. For instance, removing a branch from a spanning tree results in two connected components (each one a tree on its own), i.e., every branch is a cutset of the tree (trees are 1-arc-connected). The set of arcs in the original graph with one end vertex in each of these two components is a fundamental cutset of the graph:

Definition 3.49. (fundamental cutset) *Let $\mathbf{V}_1$ and $\mathbf{V}_2$ be the two sets of (connected) vertices obtained by removing branch a from the spanning tree $\mathcal{T}$ of graph $\mathcal{G}$. The cutset $\langle \mathbf{V}_1, \mathbf{V}_2 \rangle$ in $\mathcal{G}$ is called a fundamental cutset of graph $\mathcal{G}$ in relation to the spanning tree $\mathcal{T}$.*

Obviously, there is a single branch in every fundamental cutset.

Fundamental cutsets may also be defined for spanning forests, if attention is restricted to the connected component of the removed branch.

If $\mathcal{T}(\mathbf{V}, \mathbf{A}')$ is a spanning tree of a given graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$, then adding a chord to the tree creates a fundamental circuit:

Definition 3.50. (fundamental circuit) *Given a graph $\mathcal{G}$ and a spanning tree $\mathcal{T}$, the circuit that is created by adding a chord to $\mathcal{T}$ is called a fundamental circuit of graph $\mathcal{G}$ relative to the spanning tree $\mathcal{T}$.*

Similarly, there is a single chord in every fundamental circuit.

Fundamental circuits may also be defined with relation to spanning forests. Again, adding a chord to a spanning forest creates a single circuit.

It can be proved that any cutset of a graph $\mathcal{G}$ contains at least one branch of every spanning forest of $\mathcal{G}$. Similarly, a circuit of a graph $\mathcal{G}$ contains at least one chord of every spanning forest of $\mathcal{G}$. For connected graphs, replace "forest" with "tree" in the preceding text.

It can also be proved that the branches in a fundamental circuit are exactly those branches whose fundamental cutsets contain the chord of the circuit. Conversely, the chords in a fundamental cutset are exactly those chords whose fundamental circuits contain the branch of the cutset.

Let $\mathcal{F}$ be a spanning forest of graph $\mathcal{G}$. Let c be a chord of $\mathcal{F}$ and $\mathbf{C}$ its associated fundamental circuit. If a branch b of $\mathcal{F}$ in $\mathbf{C}$ is cut, the corresponding connected component of $\mathcal{F}$ is split into two components. The arcs of $\mathcal{G}$ with end points in each of these two components are the fundamental cutset relative to branch b. c is part of this cutset, by the result in the previous paragraph. Hence c connects two different connected components of $\mathcal{F} - b$, thus it is not part of any circuit. The conclusion is that if in a spanning forest $\mathcal{F}$ a chord c is introduced and any branch b in its fundamental circuit is removed, a new spanning forest $\mathcal{F} + c - b$ will result. It can also be shown, using the results of the previous paragraph, that if in a spanning forest $\mathcal{F}$ a branch b is removed and any chord c in its fundamental cutset is introduced, a new spanning forest $\mathcal{F} + c - b$ will still result.

Of course, there are trivial versions of fundamental circuits and cutsets. In the case of self-connecting arcs, which are always chords, the corresponding fundamental circuit consists of that single self-connecting arc, and thus does not contain any branches. On the other hand, bridges, which are always branches, correspond to fundamental cutsets consisting of that single bridge, and thus contain no chords.

Spanning k-trees

Spanning k-trees are an important theoretical framework for segmentation algorithms, since they partition a graph into k components:

Definition 3.51. (spanning k-tree) *A k-tree ${}^k\mathcal{T}(\mathbf{V}', \mathbf{A}')$ of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ is a spanning k-tree of $\mathcal{G}$ if it covers $\mathcal{G}$, i.e., $\mathbf{V}' = \mathbf{V}$.*

Since a spanning k-tree ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ has the same number of vertices as $\mathcal{G}$ and $\mathbf{A}_s \subseteq \mathbf{A}$, it follows that it has at least as many connected components. Hence, if c is the number of connected components of $\mathcal{G}$, then $k \geq c$. If $k = c$, then ${}^k\mathcal{T} = {}^c\mathcal{T}$ is also a spanning forest of $\mathcal{G}$. Also, by Theorem 3.2, $\#\mathbf{A}_s = \#\mathbf{V} - k$.

Theorem 3.4. *There is a spanning $k-1$-tree of a graph containing any spanning k-tree of the same graph, provided $k > c$, where c is the number of connected components of the graph.*

Proof. Let ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ be a spanning k-tree of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$, such that $k > c$, c being the connected components of $\mathcal{G}$. The set $\mathbf{A} \setminus \mathbf{A}_s$ is not empty, since otherwise ${}^k\mathcal{T} = \mathcal{G}$, and hence $k = c$, which is not true by hypothesis. The set $\mathbf{A} \setminus \mathbf{A}_s$ has at least one arc which connects two different connected components of ${}^k\mathcal{T}$, since otherwise all the arcs of $\mathcal{G}$ not already in ${}^k\mathcal{T}$ could be introduced into ${}^k\mathcal{T}$, effectively transforming it into $\mathcal{G}$, without changing the number of connected components, i.e., $k = c$, which is not true by hypothesis. Let then $a \in \mathbf{A} \setminus \mathbf{A}_s$ connect two connected components of ${}^k\mathcal{T}$. Clearly a cannot introduce any circuit in ${}^k\mathcal{T}$. Hence, the new connected component obtained by adding a is a tree, and the resulting subgraph of $\mathcal{G}$ has $k-1$ trees and still covers $\mathcal{G}$: it is a spanning $k-1$-tree. □

The same way that inserting a connector (see Definition 3.52) to spanning k-tree resulted in a spanning $k-1$-tree, removing a branch from a spanning k-tree results in a spanning $k+1$-tree:

Theorem 3.5. *Any spanning k-tree of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ contains a spanning $k+1$-tree of the same graph, provided that $k < \#\mathbf{V}$.*

Proof. Let ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ be a spanning k-tree of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$, such that $k < \#\mathbf{V}$. The set $\mathbf{A}_s$ is not empty, since $\#\mathbf{A}_s = \#\mathbf{V} - k$. Consider an arc $a \in \mathbf{A}_s$. The removal of a does not affect the number of vertices in ${}^k\mathcal{T}$. Hence, ${}^k\mathcal{T} - a$ is still spanning. Since removal of a cannot introduce any circuit, ${}^k\mathcal{T}$ remains a forest. Hence, the number of connected components of ${}^k\mathcal{T} - a$ is $c = \#\mathbf{V} - \#\mathbf{A}_s + 1 = k + 1$. Hence, ${}^k\mathcal{T}$ is a spanning forest with $k+1$ connected components: it is a spanning $k+1$-tree. □

Branches, chords and connectors

Definition 3.52. (Branch, chord and connector) *Let ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ be a spanning k-tree of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ with c connected components. The arcs of $\mathcal{G}$ will be named branches of ${}^k\mathcal{T}$ if they belong to $\mathbf{A}_s$, chords of ${}^k\mathcal{T}$ if they do not belong to $\mathbf{A}_s$ but their end vertices belong to the same connected component of ${}^k\mathcal{T}$, and connectors if they also do not belong to $\mathbf{A}_s$ but their end vertices belong to different connected components of ${}^k\mathcal{T}$.*

Fundamental circuits and cutsets

A chord of a spanning k-tree ${}^k\mathcal{T}$ of a graph $\mathcal{G}$ is also a chord of one of its components, say ${}^k\mathcal{T}_i$. Hence, introduction of a chord into the k-tree creates a circuit. This circuit is also a fundamental circuit of $\mathcal{G}_i$, the subgraph of $\mathcal{G}$ induced by the vertices of ${}^k\mathcal{T}_i$.

Definition 3.53. (fundamental circuit of a k-tree) *The circuit created in a k-tree by introduction of one of its chords.*

If a chord of a spanning k-tree is inserted into the spanning k-tree and any of the branches in the resulting circuit is removed, a spanning k-tree still results. This stems from a similar result for trees.

Notice that self-connecting arcs are chords of any spanning k-tree, their corresponding fundamental circuit containing only the self-connecting arc.

A branch of a spanning k-tree ${}^k\mathcal{T}$ of a graph $\mathcal{G}$ is also a branch of one of its components, say ${}^k\mathcal{T}_i$. Hence, removal of a branch from a k-tree disconnects the corresponding ${}^k\mathcal{T}_i$. The chords of $\mathcal{G}_i$ relative to ${}^k\mathcal{T}_i$ with end vertices in each of the components thus obtained form a fundamental cutset of $\mathcal{G}_i$ relative to ${}^k\mathcal{T}_i$:

Definition 3.54. (fundamental cutset of a k-tree) *The cutset associated to the two components obtained by removal of a branch of a k-tree from one of its components.*

If a branch of a spanning k-tree is removed from the spanning k-tree and any of the chords in the associated fundamental cutset is inserted, a spanning k-tree still results. This stems from a similar result for trees.

Since removal of any branch in a k-tree increases the number of connected components, inserting a connector to a spanning k-tree and removing one of its branches still results in a spanning k-tree.

Connectors of a k-tree connect distinct trees of the k-tree, and thus their introduction to the k-tree reduces the number of connected components. Since removal of any branch in a k-tree increases the number of connected components, inserting a connector to a k-tree and removing one of its branches still results in a k-tree.

A bridge can either be a branch or a connector of a spanning k-tree, but it can never be a chord. If it is a branch, its corresponding fundamental cutset consists of only itself.

Notice that, after inserting a connector into a k-tree, thereby creating a $k-1$-tree, some of the connectors of the k-tree may become chords of the new $k-1$-tree. This is guaranteed not to happen when the connector is a bridge.

Seeded spanning k-trees

Definition 3.55. (n-seed respecting spanning k-tree) *Given a graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ and a set $\mathbf{S} \subseteq \mathbf{V}$ of n vertex seeds, a spanning k-tree which includes at most one seed in each of its component trees is a n-seed respecting spanning k-tree.*

In the following, the term seeded spanning k-tree will be used often instead of n-seed respecting spanning k-tree.

Obviously, a n-seed respecting spanning k-tree of a graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ must have $n \leq k \leq \#\mathbf{V}$.

Definition 3.56. (smallest n-seed respecting spanning k-tree) *Given a graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ and a set $\mathbf{S} \subseteq \mathbf{V}$ of n vertex seeds, a spanning k-tree which includes at most one seed in each* *of its component trees, where k is as small as possible, is a smallest n-seed respecting spanning k-tree.*

Theorem 3.6. *A smallest n-seed respecting spanning k-tree of a graph with c connected components has $k = n + c'$, where c' is the number of connected components of the graph which do not contain any seed.*

Proof. Let $\mathcal{G}_i$, $i = 1, \ldots, c$ be the components of the graph $\mathcal{G}$ to span. If $\mathcal{G}_i$ contains no seeds, it can be spanned by a single component of the k-tree. If it contains n_i seeds, it can be spanned by n_i components of the k-tree. Hence, $k = c' + \sum n_i = c' + n$. □

Corollary 3.7. *A smallest n-seed respecting spanning k-tree of a graph with c connected components, each containing at least one seed, has $k = n$.*

Branches, chords, connectors and separators

Branches, chords, and connectors are defined for seeded spanning k-trees as for spanning k-trees. However, connectors between seeded trees, which cannot be inserted into seeded spanning trees without violating seed separation, will be named separators:

Definition 3.57. (separator) *Separators are arcs connecting two different components both containing seeds.*

Hence, in seeded spanning k-trees the term connector will be reserved for simple, non-separating connectors, all other connectors being separators.

Theorem 3.8. *A seeded spanning k-tree is smallest iff it has no connectors, only separators.*

Proof. Let ${}^k\mathcal{T}$ be a seeded spanning tree of $\mathcal{G}$. It will be proved first that if ${}^k\mathcal{T}$ is smallest, then it has no connectors. Then it will be proved that if ${}^k\mathcal{T}$ has no connectors, then it must be smallest.

If there is a connector, that is, a connector between a seedless component of the k-tree and any other component, then that connector can be inserted into the k-tree, thereby transforming it into a spanning $k - 1$-tree which is still respects the set of seeds, since at most one of the components connected through that connector has a seed. Hence, the spanning k-tree can not be smallest.

If ${}^k\mathcal{T}$ is not smallest, then, since each seed in $\mathbf{S} = \{s_1, \ldots, s_n\}$ is contained in a single component ${}^k\mathcal{T}_i$ of the spanning k-tree and all such seeded components are separated, there must be at least one seedless component of ${}^k\mathcal{T}$ in a seeded connected component $\mathcal{G}_m$ of $\mathcal{G}$ or two seedless components of ${}^k\mathcal{T}$ in a seedless connected component $\mathcal{G}_m$ of $\mathcal{G}$. Otherwise, by Theorem 3.6, ${}^k\mathcal{T}$ would indeed be smallest. In both cases, there must be a connector between one of the seedless components and another component of the same connected component $\mathcal{G}_m$, since $\mathcal{G}_m$ is connected. Hence, there are connectors. □

Theorem 3.9. *All seeded spanning k-trees which are not smallest are subgraphs of some seeded spanning $k-1$-tree of the same graph for the same set of seeds.*

Proof. Let ${}^k\mathcal{T}$ be a seeded spanning k-tree of some graph $\mathcal{G}$. Assume ${}^k\mathcal{T}$ is not smallest. By Theorem 3.8, ${}^k\mathcal{T}$ must have at least one connector. Take any connector c of ${}^k\mathcal{T}$ and build the graph ${}^k\mathcal{T}+c$. Since c is a connector, it connected two connected components of ${}^k\mathcal{T}$, and hence it introduced no circuits. Since the number of connected components in ${}^k\mathcal{T}+c$ reduced by one, it is obvious that it is a $k-1$-tree. Since at least one of the components connected by c is seedless, ${}^k\mathcal{T}+c$ also respects seed separation. Hence, ${}^k\mathcal{T}+c$ is a seeded spanning $k-1$-tree of which ${}^k\mathcal{T}$ is a subgraph. □

Theorem 3.10. *All seeded spanning k-trees with at least one branch have subgraphs which are seeded spanning $k+1$-trees of the same graph for the same set of seeds.*

Proof. Let ${}^k\mathcal{T}$ be a seeded spanning k-tree of some graph $\mathcal{G}$. Assume ${}^k\mathcal{T}$ has at least one branch. Take any branch b of ${}^k\mathcal{T}$ and build the graph ${}^k\mathcal{T}-b$. Since b is a branch, its removal separates a component of ${}^k\mathcal{T}$ into two components, only one of which may have a seed. Hence, the resulting graph is a seeded spanning $k+1$-tree and a subgraph of ${}^k\mathcal{T}$. □

Fundamental circuits, cutsets, and paths

Fundamental circuits and cutsets are defined for seeded spanning k-trees as for spanning k-trees.

Definition 3.58. (fundamental path) *Let c be a separator of a seeded spanning k-tree ${}^k\mathcal{T}$. Let separator c separate two components ${}^k\mathcal{T}_i$ and ${}^k\mathcal{T}_j$ with seeds s_i and s_j, respectively. The only s_i, s_j-path in the tree ${}^k\mathcal{T}_i \cup {}^k\mathcal{T}_j \cup \{c\}$ is the fundamental path of ${}^k\mathcal{T}$ relative to separator c.*

As happened with spanning k-trees, in seeded spanning k-trees any branch can be exchanged with a chord in its fundamental cutset and any chord can be exchanged with a branch in its fundamental circuit: none of these operations violates the seeds separation, since the set of vertices in each connected component of the k-tree does not change, only the arcs change.

Again as happened with spanning k-trees, in seeded spanning k-trees any connector can be exchanged with any branch. This operation does not violate the seed separation, since inserting a connector (by definition of connector) does not connect two seeds and removing any branch can only result in an extra, seedless component. The final number of components is the same.

A similar result can be proved for separators. In a seeded spanning k-tree any separator can be exchanged with any branch in its fundamental path. This is so because, after insertion of the separator into the k-tree, the two components plus the separator form a new tree and the fundamental path is the only path in this new tree between its two seeds. Hence, if any branch in this path is removed, the seed separation is again valid and the number of components of the k-tree is the same.

Graphs and vector spaces

Definition 3.59. (circuit vector [186]**)** *Circuits and unions of arc-disjoint circuits of a graph are called circuit vectors of a graph.*

Definition 3.60. (cutset vector [186]**)** *Cutsets and unions of arc-disjoint cutsets of a graph are called cutset vectors of a graph.*

Hence, cut and cutset vector are one and the same concept.

The names circuit and cutset vectors stem from the fact that the set of arc induced subgraphs of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ with c connected components, equipped with the ring sum of sets operator,[7] is a vector space of dimension $\#\mathbf{A}$ over the Galois field GF(2) (see [186] for details[8]) where two orthogonal subspaces can be defined: the subspace of all circuits and unions of arc-disjoint circuits and the subspace of all cutsets and unions of arc-disjoint cutsets.

It can be proved that, given a spanning forest $\mathcal{F}(\mathbf{V}, \mathbf{A}_s)$ of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$, the set of its fundamental circuits (dimension $\#\mathbf{A} - \#\mathbf{A}_s = \#\mathbf{A} - \#\mathbf{V} + c = \mu(\mathcal{G})$) and the set of its fundamental cutsets (dimension $\#\mathbf{A}_s = \#\mathbf{V} - c = \rho(\mathcal{G})$) are bases of the subspace of circuits and unions of arc-disjoint circuits and of the subspace of cutsets and unions of arc-disjoint cutsets, respectively. Hence, any circuit vector can be written as a ring sum of fundamental circuits and any cutset vector (or cut) can be written as a ring sum of fundamental cutsets.

Let $\mathcal{F}(\mathbf{V}, \mathbf{A}_s)$ be a spanning forest of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$. Given a circuit consisting of the set of arcs $\mathbf{C}$, decompose it into two disjoint sets $\mathbf{C}_b \subseteq \mathbf{A}_s$ and $\mathbf{C}_c \subseteq \mathbf{A} \setminus \mathbf{A}_s$, containing its branches and its chords, respectively. Then, it can be proved that $\mathbf{C}$ is the ring sum of the fundamental circuits (relative to $\mathcal{F}$) corresponding to the chords in $\mathbf{C}_c$ (and no other combination of fundamental circuits results in $\mathbf{C}$). It is straightforward to prove additionally that the branches in $\mathbf{C}_b$ occur in an odd number of such fundamental circuits, since branches which do occur an even number of times disappear in the ring sum.

The same thing can be proved for cuts relative to fundamental cutsets, if branches and chords are exchanged in the previous paragraph.

Shortest spanning trees and forests

Let $w(\cdot) : \mathbf{A} \to \mathbb{R}$ be a weight function defined on the arcs $\mathbf{A}$ of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$. Let $W(\cdot)$ be defined as

$$W(\mathbf{A}, w) = \sum_{a \in \mathbf{A}} w(a).$$

Definition 3.61. (shortest [or minimal] spanning tree) *A subgraph $\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ of connected graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ is a SST (Shortest Spanning Tree) of $\mathcal{G}$ if $\mathcal{T}$ is a spanning tree and no other spanning tree $\mathcal{T}'(\mathbf{V}, \mathbf{A}'_s)$ exists such that $W(\mathbf{A}'_s, w) < W(\mathbf{A}_s, w)$.*

[7]The ring sum of two sets $\mathbf{A}$ and $\mathbf{B}$ is $\mathbf{A} \oplus \mathbf{B} = (\mathbf{A} \cup \mathbf{B}) \setminus (\mathbf{A} \cap \mathbf{B})$.

[8]Multiplication of a set by 1 results in the same set, multiplication by 0 results in the empty set $\emptyset$.

LST (Longest Spanning Tree), SSF (Shortest Spanning Forest), and LSF (Longest Spanning Forest) are defined similarly.

Theorem 3.11. (chord and branch condition) *Let $\mathcal{F}(\mathbf{V}, \mathbf{A}_s)$ be a spanning forest of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$. The following are equivalent:*

1. *$\mathcal{F}$ is a SSF of $\mathcal{G}$.*
2. *(branch condition) $w(b) \leq w(c)$ for any branch b of $\mathcal{F}$ and for any chord c in the corresponding fundamental cutset relative to $\mathcal{F}$.*
3. *(chord condition) $w(c) \geq w(b)$ for any chord c of $\mathcal{F}$ and for any branch b in the corresponding fundamental circuit relative to $\mathcal{F}$.*

Proof. It will be shown that $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 1$.

$1 \Rightarrow 2$: Assume 2 does not hold for $\mathcal{F}$, i.e., there is a branch b and a chord c in the fundamental cutset of b such that $w(b) > w(c)$. It was shown before that $\mathcal{F} + c - b$ is still a spanning forest. But $W(\mathbf{A}_s \cup \{c\} \setminus \{b\}, w) = W(\mathbf{A}_s, w) + w(c) - w(b) < W(\mathbf{A}_s, w)$. Hence, $\mathcal{F}$ cannot be a SSF of $\mathcal{G}$, that is, $\neg 2 \Rightarrow \neg 1$, which is the same as $1 \Rightarrow 2$.

$2 \Rightarrow 3$: Assume 2 holds for $\mathcal{F}$. Take any chord c of $\mathcal{F}$ and its corresponding fundamental circuit $\mathbf{C}$. From a result before, c belongs to the fundamental cutsets of all branches b in $\mathbf{C}$. Hence, since the branch condition holds, $w(c) \geq w(b)$ for all branches in $\mathbf{C}$ and 3 holds.

$3 \Rightarrow 1$: Assume 3 holds for $\mathcal{F}(\mathbf{V}, \mathbf{A}_s)$. Let $\mathcal{F}'(\mathbf{V}, \mathbf{A}'_s)$ be a SSF of $\mathcal{G}$, for which, as was already proven, 3 holds ($1 \Rightarrow 2$ and $2 \Rightarrow 3$). It will be shown that $\mathcal{F}'$ can be made equal to $\mathcal{F}$ through a series of simple operations which neither increases nor decreases the total weight $W(\mathbf{A}'_s, w)$. This proves that $\mathcal{F}$ is indeed a SSF, since $W(\mathbf{A}_s, w) = W(\mathbf{A}'_s, w)$.

If $\mathcal{F}$ and $\mathcal{F}'$ are equal, then $\mathcal{F}$ is also a SSF. Suppose then that $\mathcal{F}$ and $\mathcal{F}'$ are different. Since $\mathcal{F}$ and $\mathcal{F}'$ are both spanning forest, both contain the same number of arcs of $\mathcal{G}$: $\#\mathbf{A}_s = \#\mathbf{A}'_s$. Hence, there is an equal non-zero number of arcs in $\mathbf{A}_s \setminus \mathbf{A}'_s$ and $\mathbf{A}'_s \setminus \mathbf{A}_s$. The arcs in $\mathbf{A}'_s \setminus \mathbf{A}_s$ are chords of $\mathcal{F}$ and vice versa.

Take a chord c of $\mathcal{F}$ which is also a branch of $\mathcal{F}'$. Let $\mathbf{C}$ be the corresponding fundamental circuit in $\mathcal{F}$. $\mathbf{C}$ can be written as the ring sum of the fundamental circuits of $\mathcal{F}'$ corresponding to the chords of $\mathcal{F}'$ in $\mathbf{C}$. Since $c \in \mathbf{A}'_s$, it must occur in an odd number of these fundamental circuits, corresponding to an odd number of chords of $\mathcal{F}'$ in $\mathbf{C}$. Let then b be a chord of $\mathcal{F}'$ in circuit $\mathbf{C}$ such that its fundamental circuit in $\mathcal{F}'$ includes c. Then $w(c) \geq w(b)$, since b and c are respectively branch and chord of the same fundamental circuit in $\mathcal{F}$ (3 holds for $\mathcal{F}$), and $w(b) \geq w(c)$, since b and c are also respectively chord and branch of the same fundamental circuit in $\mathcal{F}'$ (3 holds for $\mathcal{F}'$). Hence, $w(b) = w(c)$. Since $\mathcal{F}' - c + b$ is also a spanning forest and it has the same weight as $\mathcal{F}'$, the chord c of $\mathcal{F}$ can be substituted by the branch b of $\mathcal{F}$ in $\mathcal{F}'$. Repeating this process while there are branches of $\mathcal{F}'$ which are chords of $\mathcal{F}$, such arcs can be successively eliminated from $\mathcal{F}'$ without changing its global weight $W(\mathbf{A}'_s, w)$. The process terminates when $\mathbf{A}'_s \setminus \mathbf{A}_s$ is empty, i.e., when $\mathbf{A}'_s = \mathbf{A}_s$. Hence, $W(\mathbf{A}_s, w) = W(\mathbf{A}'_s, w)$ and thus $\mathcal{F}$ is also a SSF. □

For LSF, simply invert the weight relations in the chord and branch conditions.

Lemma 3.12. *Let $\mathcal{T}'(\mathbf{V}', \mathbf{A}'_s)$ be a connected subgraph of a spanning forest $\mathcal{F}(\mathbf{V}, \mathbf{A}_s)$ of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$. Let $\mathcal{G}'(\mathbf{V}', \mathbf{A}')$ be the subgraph of $\mathcal{G}$ induced by the vertices $\mathbf{V}'$ in $\mathcal{T}'$. If $\mathcal{T}''(\mathbf{V}', \mathbf{A}''_s)$ is a spanning tree of $\mathcal{G}'$, then the graph $\mathcal{F}''(\mathbf{V}, \mathbf{A}_s \setminus \mathbf{A}'_s \cup \mathbf{A}''_s)$ is also a spanning forest of $\mathcal{G}$.*

Proof. The subgraph $\mathcal{T}'$ is acyclic, since it is a subgraph of the acyclic graph $\mathcal{F}$. Since it is also connected, $\mathcal{T}'$ is a tree. Since it spans the vertices of $\mathcal{G}'$, it is a spanning tree of $\mathcal{G}'$.

Notice that, since $\mathbf{A}'_s \subseteq \mathbf{A}_s$, then $\#(\mathbf{A}_s \setminus \mathbf{A}'_s) = \#\mathbf{A}_s - \#\mathbf{A}'_s$. Also notice that $\#\mathbf{A}'_s = \#\mathbf{A}''_s$, since both $\mathcal{T}'$ and $\mathcal{T}''$ span $\mathcal{G}'$.

Suppose $\mathbf{A}_s \setminus \mathbf{A}'_s$ and $\mathbf{A}''_s$ have a common arc a. This implies, of course, that a belongs to $\mathbf{A}_s$ and $\mathbf{A}''_s$ but not to $\mathbf{A}'_s$. Since $a \in \mathbf{A}''_s$, a is incident on two vertices of $\mathcal{G}'$. These two vertices, by definition of tree, are connected through a unique path in $\mathcal{T}'$. Hence, there are two different paths in $\mathcal{F}$ between these two vertices (viz. a and the path in $\mathcal{T}'$), which is a contradiction, since $\mathcal{F}$ is a forest. Thus, $\mathbf{A}_s \setminus \mathbf{A}'_s$ and $\mathbf{A}''_s$ have no common arcs, i.e., $\#(\mathbf{A}_s \setminus \mathbf{A}'_s \cup \mathbf{A}''_s) = \#\mathbf{A}_s - \#\mathbf{A}'_s + \#\mathbf{A}''_s = \#\mathbf{A}_s$.

Since $\mathcal{F}$ has the same number of connected components c as $\mathcal{G}$, and since $\mathcal{F}''$ has only arcs of $\mathcal{G}$, it is clear that $\mathcal{F}''$ cannot have less connected components than $\mathcal{F}$. It will be shown that it neither can have more connected components.

Consider the path $\mathbf{P}$ between any two connected vertices v_1 and v_2 in $\mathcal{F}$.

If $\mathbf{P}$ does not include any arcs of $\mathbf{A}'_s$, then clearly v_1 and v_2 are also connected in $\mathcal{F}''$. If it contains arcs of $\mathbf{A}'_s$, these arcs connect vertices of $\mathcal{G}'$ which are connected by a path in $\mathcal{T}''$. Hence, all arcs of $\mathbf{A}'_s$ in $\mathbf{P}$ can be substituted by a path in $\mathcal{T}''$, containing only arcs in $\mathbf{A}''_s$. Thus, a walk between the two vertices v_1 and v_2 exists in $\mathcal{F}''$. Hence, v_1 and v_2 are also connected in $\mathcal{F}''$.

Thus, $\mathcal{F}$ and $\mathcal{F}''$ have the same number of connected components c. Since they also have the same number of arcs $\#\mathbf{A}_s = \#\mathbf{A}''_s = \#\mathbf{V} - c$, $\mathcal{F}''$ is indeed a (spanning) forest of $\mathcal{G}$. □

Theorem 3.13. *Let $\mathcal{F}'(\mathbf{V}', \mathbf{A}'_s)$ be a subgraph of a spanning forest $\mathcal{F}(\mathbf{V}, \mathbf{A}_s)$ of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$. Let $\mathcal{F}'_i(\mathbf{V}'_i, \mathbf{A}'_{s_i})$ be a connected component of $\mathcal{F}'$ and $\mathcal{G}_i(\mathbf{V}'_i, \mathbf{A}'_i)$ the subgraph of $\mathcal{G}$ induced by $\mathbf{V}'_i$. If $\mathcal{F}$ is a SSF, then $\mathcal{F}'_i$ is a SST of the connected graph $\mathcal{G}_i$.*

Proof. Suppose there is an i for which $\mathcal{F}'_i$ is not a SST of $\mathcal{G}_i$. Then, there is a lighter way to cover $\mathcal{G}_i$. Let $\mathcal{F}''_i$ be a lighter covering of $\mathcal{G}_i$. Clearly, the branches of $\mathcal{F}$ which are in $\mathcal{F}'_i$ could then be substituted by the branches in $\mathcal{F}''_i$, thus reducing its overall weight. Since this substitution, by Lemma 3.12, still results in a spanning tree, $\mathcal{F}$ cannot be a SSF, which is a contradiction. Thus, $\mathcal{F}'_i$ is indeed a SST of $\mathcal{G}_i$. □

Corollary 3.14. *If $\mathcal{F}$ is a SSF of graph $\mathcal{G}$ and $\mathcal{F}_i$ one of its connected components, then $\mathcal{F}_i$ is a SST of the corresponding connected component $\mathcal{G}_i$ of $\mathcal{G}$.*

Proof. The result is immediate from Theorem 3.13. □

Algorithms

There are two classic algorithms for computing the SST of a connected graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$: Kruskal and Prim [28]. Both work by successively adding to $\mathbf{A}'$ arcs from $\mathbf{A} \setminus \mathbf{A}'$ that are safe for $\mathbf{A}'$. An arc a is safe for $\mathbf{A}'$, where $\mathbf{A}'$ is a subset of the arcs on some SST of $\mathcal{G}$, if $\mathbf{A}' \cup \{a\}$ is also a subset of the arcs of some SST of $\mathcal{G}$. The set $\mathbf{A}'$ is initially empty. Thus $\mathbf{A}'$ is kept always as the subset of the arcs in some SST of $\mathcal{G}$, this being the algorithm's invariant. Hence, the subgraph $\mathcal{F}(\mathbf{V}, \mathbf{A}')$ is an acyclic graph, i.e., a forest. The algorithm finishes after exactly $\#\mathbf{V} - 1 = \rho(\mathcal{G})$ arc insertions, when the forest $\mathcal{F}(\mathbf{V}, \mathbf{A}')$ becomes a SST of $\mathcal{G}(\mathbf{V}, \mathbf{A})$.

Kruskal's algorithm [28] simply adds to $\mathbf{A}'$ one of the lightest of the arcs connecting any two trees in the forest $\mathcal{F}$. It runs, with an appropriate implementation, in $O(\#\mathbf{A} \lg \#\mathbf{A})$. In the case of planar, simple graphs, it runs in $O(\#\mathbf{V} \lg \#\mathbf{V})$ (see Section 3.4.1).

Prim's algorithm [28] also successively adds arcs to $\mathbf{A}'$, though the arc added at each step is chosen as one of the arcs having minimum weight with a single end vertex in the subgraph induced by $\mathbf{A}'$. When $\mathbf{A}'$ is empty, the subgraph consists of an arbitrarily chosen vertex, which is the "seed" of the algorithm. Hence, the subgraph $\mathcal{G}'(\mathbf{V}', \mathbf{A}')$ induced by $\mathbf{A}'$ on $\mathcal{G}$ is a tree at all steps of the algorithm. An implementation using Fibonacci priority queues runs in $O(\#\mathbf{A} + \#\mathbf{V} \lg \#\mathbf{V})$, which, in the case of planar, simple graphs, is asymptotically the same as $O(\#\mathbf{V} \lg \#\mathbf{V})$ (see Section 3.4.1).

Both algorithms can be proven correct through the use of the following theorem from [28, Corolary 24.2], reproduced here without proof:

Theorem 3.15. *Let $\mathcal{G}(\mathbf{V}, \mathbf{A})$ be a connected graph with arc weight function $w(\cdot) : \mathbf{A} \to \mathbb{R}$. Let $\mathbf{A}'$ be a subset of the arcs in some SST of $\mathcal{G}$. Let $\mathcal{F}_i$ be a connected component in the forest $\mathcal{F}(\mathbf{V}, \mathbf{A}')$. If a is one of the lightest arcs connecting $\mathcal{F}_i$ to some other connected component of $\mathcal{F}$, then a is safe for $\mathbf{A}'$, i.e., $\mathbf{A}' \cup \{a\}$ is also a subset of the arcs in some SST of $\mathcal{G}$.*

These algorithms are also applicable to disconnected graphs, and hence also solve the SSF problem. The Kruskal algorithm works without change. As for the Prim algorithm, each run of the basic algorithm builds the SST of a connected component of the graph. Hence, c runs of the basic algorithm are necessary in a graph with c connected components. Since vertices already in some tree may be marked in constant time at each step of the algorithm, thus not changing its asymptotic performance, the seeds may be searched in linear time, by searching the next unmarked vertex in a list of graph vertices. Alternatively, both the Prim and Kruskal algorithms may be applied in parallel to each of the c graph components.

Shortest spanning k-trees

SSkTs are an important framework in which to describe some segmentation algorithms:

Definition 3.62. (shortest [or minimal] spanning k-tree) *A spanning k-tree of a graph is a SSkT (Shortest Spanning k-Tree) of that graph if no other spanning k-tree exists with a smaller weight.*

Theorem 3.16. *The connected components ${}^k\mathcal{T}_i(\mathbf{V}_i, \mathbf{A}_{s_i})$, with $i = 1, \ldots k$, of a SSkT ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$, are SSTs of the subgraphs $\mathcal{G}_i$ induced in $\mathcal{G}$ by the corresponding set of vertices $\mathbf{V}_i$.*

Proof. Let $\mathcal{G}_i(\mathbf{V}_i, \mathbf{A}_i)$ be the subgraph of $\mathcal{G}$ induced by the vertices $\mathbf{V}_i$ of a tree ${}^k\mathcal{T}_i$ of ${}^k\mathcal{T}$. Suppose ${}^k\mathcal{T}_i$ is not a SST of $\mathcal{G}_i$. Then it is possible to chose another tree spanning $\mathcal{G}_i$ with a smaller weight than ${}^k\mathcal{T}_i$ (choose a SST of $\mathcal{G}_i$), without changing the weight associated to the other components of ${}^k\mathcal{T}$ and thereby reducing the weight of ${}^k\mathcal{T}$. But this is a contradiction, since $\mathcal{T}_k$ is a SSkT of $\mathcal{G}$. Hence, ${}^k\mathcal{T}_i$ is indeed a SST of $\mathcal{G}_i$, for $i = 1, \ldots, k$. □

The converse of this theorem is not true. Not all spanning k-trees of a graph $\mathcal{G}$ with the property that each tree is a SST of its corresponding subgraph of $\mathcal{G}$ are SSkT of $\mathcal{G}$. Counter examples are easy to contrive.

Lemma 3.17. *Let ${}^k\mathcal{T}$ be a spanning k-tree of $\mathcal{G}$. If $\mathbf{C}$ is a circuit in $\mathcal{G}$ containing only branches and chords of ${}^k\mathcal{T}$, then $\mathbf{C}$ can be expressed as a ring sum of fundamental circuits of ${}^k\mathcal{T}$, and all branches of ${}^k\mathcal{T}$ in $\mathbf{C}$ occur in an odd number of these fundamental circuits.*

Proof. First it will be proved that circuit $\mathbf{C}$ is contained in the subgraph $\mathcal{G}_i$ induced by the vertices of a component ${}^k\mathcal{T}_i$ of ${}^k\mathcal{T}$. Then, using Theorem 3.16 and the fact that circuits in a graph can be expressed as ring sums of its fundamental circuits relative to some SSF, the result is immediate.

Let circuit $\mathbf{C}$, of length $l > 0$, consist of $v_0, a_1, v_1, \ldots, v_{l-1}, a_l, v_l$. Since the trees ${}^k\mathcal{T}_i$ of the spanning ${}^k\mathcal{T}$ partition the set of vertices into disjoint sets, one for each ${}^k\mathcal{T}_i$, v_0 belongs to some ${}^k\mathcal{T}_i$, with $i \in \{1, \ldots, k\}$. Suppose now that v_n, with $0 \leq n < l$, also belongs to ${}^k\mathcal{T}_i$. The arc a_{n+1} is not a connector. If it is a branch, it connects two vertices from the same component of ${}^k\mathcal{T}$. The same thing happens if a_{n+1} is a chord. Hence, v_{n+1} also belongs to ${}^k\mathcal{T}_i$. Hence, by induction, all vertices in the circuit belong to the same component of ${}^k\mathcal{T}$. Since all the arcs in $\mathcal{G}$ incident on vertices of the same component ${}^k\mathcal{T}_i$ of ${}^k\mathcal{T}$ belong to the corresponding $\mathcal{G}_i$, it is clear that circuit $\mathbf{C}$ is contained on some $\mathcal{G}_i$. □

Theorem 3.18. (branch, chord, and connector conditions) *Let ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ be a spanning k-tree of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$. Consider the following statements:*

1. *${}^k\mathcal{T}$ is a SSkT of $\mathcal{G}$.*
2. *(branch condition) $w(b) \leq w(c)$ for any branch b of ${}^k\mathcal{T}$ and for any chord c in the corresponding fundamental cutset relative to ${}^k\mathcal{T}$.*
3. *(chord condition) $w(b) \leq w(c)$ for any chord c of ${}^k\mathcal{T}$ and for any branch b in the corresponding fundamental circuit relative to ${}^k\mathcal{T}$.*
4. *(connector condition) $w(c) \geq w(b)$ for any branch b and for any connector c of ${}^k\mathcal{T}$.*

The fact that a spanning k-tree is also a SSkT implies that branch, chord, and connector conditions are true for ${}^k\mathcal{T}$, i.e.,

$$1 \Rightarrow 2 \wedge 3 \wedge 4.$$

On the other hand, the connector condition together with either the branch or the chord conditions are sufficient to guarantee that a spanning k-tree is a SSkT, i.e.,

$$2 \wedge 4 \Rightarrow 1, \textit{ and}$$
$$3 \wedge 4 \Rightarrow 1.$$

Proof. It will first be shown that $3 \Leftrightarrow 2$. Then, it suffices to show that $1 \Rightarrow 3$, $1 \Rightarrow 4$, and $3 \wedge 4 \Rightarrow 1$.

Let $\mathcal{G}_i$ be the subgraphs of $\mathcal{G}$ induced by the vertices of the connected components ${}^k\mathcal{T}_i$ of ${}^k\mathcal{T}$.

$3 \Leftrightarrow 2$: Both 3 and 2 apply to chords and branches of the same component ${}^k\mathcal{T}_i$ of ${}^k\mathcal{T}$. Since each component ${}^k\mathcal{T}_i$ of ${}^k\mathcal{T}$ is a spanning tree of $\mathcal{G}_i$, and since, according to Theorem 3.11, chord and branch conditions are equivalent for spanning forests (of which spanning trees are particular cases), 3 and 2 must also be equivalent for the whole spanning k-tree ${}^k\mathcal{T}$.

$1 \Rightarrow 3$: Let c be a chord of ${}^k\mathcal{T}$ and $\mathbf{C}$ its fundamental circuit. If there is any $b \in \mathbf{C}$ such that $w(b) > w(c)$, then ${}^k\mathcal{T} - b + c$, which is clearly still a spanning k-tree of $\mathcal{G}$, has a smaller weight than ${}^k\mathcal{T}$, which is a contradiction, since ${}^k\mathcal{T}$ is a SSkT. Hence, 3 must hold.

$1 \Rightarrow 4$: Let c be a connector of ${}^k\mathcal{T}$. If there is any $b \in \mathbf{A}_s$ such that $w(b) > w(c)$, then ${}^k\mathcal{T} - b + c$, which is clearly still a spanning k-tree of $\mathcal{G}$, has a smaller weight than ${}^k\mathcal{T}$, which is a contradiction, since ${}^k\mathcal{T}$ is a SSkT. Hence, 4 must hold.

$3 \wedge 4 \Rightarrow 1$: Assume 3 and 4 hold for ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$. Let ${}^k\mathcal{T}'(\mathbf{V}, \mathbf{A}'_s)$ be a SSkT of $\mathcal{G}$, for which, as was already proven, 3 and 4 hold. It will be shown that ${}^k\mathcal{T}'$ can be made equal to ${}^k\mathcal{T}$ through a series of simple operations which neither increases nor decreases the total weight $W(\mathbf{A}'_s, w)$. This proves that ${}^k\mathcal{T}$ is indeed a SSkT, since $W(\mathbf{A}_s, w) = W(\mathbf{A}'_s, w)$.

If ${}^k\mathcal{T}$ and ${}^k\mathcal{T}'$ are equal, then ${}^k\mathcal{T}$ is also a SSkT. Suppose then that ${}^k\mathcal{T}$ and ${}^k\mathcal{T}'$ are different. Since ${}^k\mathcal{T}$ and ${}^k\mathcal{T}'$ are both spanning k-trees, both contain the same number of arcs of $\mathcal{G}$: $\#\mathbf{A}_s = \#\mathbf{A}'_s$. Hence, there is an equal non-zero number of arcs in $\mathbf{A}_s \setminus \mathbf{A}'_s$ and $\mathbf{A}'_s \setminus \mathbf{A}_s$. The arcs in $\mathbf{A}'_s \setminus \mathbf{A}_s$ are chords or connectors of ${}^k\mathcal{T}$ and branches of ${}^k\mathcal{T}'$ and vice versa.

Suppose there is a chord c of ${}^k\mathcal{T}$ which is also a branch of ${}^k\mathcal{T}'$. Let $\mathbf{C}$ be the corresponding fundamental circuit in ${}^k\mathcal{T}$. $\mathbf{C}$ cannot consist solely of branches of ${}^k\mathcal{T}'$, since otherwise ${}^k\mathcal{T}'$ would not be a k-tree. Hence, there are arcs of $\mathbf{C}$ which are not branches of ${}^k\mathcal{T}'$.

Suppose that, of these arcs, there is one arc b which is a connector of ${}^k\mathcal{T}'$. Then $w(c) \geq w(b)$, since c is the chord of a fundamental circuit containing b in ${}^k\mathcal{T}$ (3 holds for ${}^k\mathcal{T}$), and $w(c) \leq w(b)$, since c is a branch of ${}^k\mathcal{T}'$ and b is a connector of ${}^k\mathcal{T}'$ (4 holds for ${}^k\mathcal{T}'$). Hence, $w(c) = w(b)$. Since ${}^k\mathcal{T}' - c + b$ is also a spanning k-tree and it has the same weight as ${}^k\mathcal{T}'$, the chord c of ${}^k\mathcal{T}$ can be substituted by the branch b of ${}^k\mathcal{T}$ in ${}^k\mathcal{T}'$.

If there is no connector of ${}^k\mathcal{T}'$ in $\mathbf{C}$, then $\mathbf{C}$ consists solely of branches and chords of ${}^k\mathcal{T}'$. By Lemma 3.17, it can be expressed as the a ring sum of fundamental circuits of ${}^k\mathcal{T}'$. Since $c \in \mathbf{A}'_s$, it must occur in an odd number of these fundamental circuits, corresponding to an odd number of chords of ${}^k\mathcal{T}'$ in $\mathbf{C}$. Let then b be a chord of ${}^k\mathcal{T}'$ in circuit $\mathbf{C}$ such that its fundamental circuit in ${}^k\mathcal{T}'$ includes c. Then $w(c) \geq w(b)$, since b and c are respectively branch and chord of the same fundamental circuit in ${}^k\mathcal{T}$ (3 holds for ${}^k\mathcal{T}$), and $w(b) \geq w(c)$, since b and c are also respectively chord and branch of the same fundamental circuit in ${}^k\mathcal{T}'$ (3 holds for ${}^k\mathcal{T}'$). Hence, $w(b) = w(c)$. Since ${}^k\mathcal{T}' - c + b$ is also a spanning k-tree and it has the same weight as ${}^k\mathcal{T}'$, the chord c of ${}^k\mathcal{T}$ can be substituted by the branch b of ${}^k\mathcal{T}$ in ${}^k\mathcal{T}'$.

In either case, a branch c of ${}^k\mathcal{T}'$ can be substituted by a branch b of ${}^k\mathcal{T}$ in ${}^k\mathcal{T}'$. Repeating this process while there are branches of ${}^k\mathcal{T}'$ which are chords of ${}^k\mathcal{T}$, such arcs are eliminated from ${}^k\mathcal{T}'$ without changing its global weight $W(\mathbf{A}'_s, w)$.

If ${}^k\mathcal{T}' = {}^k\mathcal{T}$ after the above process, then ${}^k\mathcal{T}$ is indeed a SSkT. If not, then there must be some branch b of ${}^k\mathcal{T}$ which is not in ${}^k\mathcal{T}'$.

Suppose that b is a chord of ${}^k\mathcal{T}'$ and $\mathbf{C}'$ is its corresponding fundamental circuit in ${}^k\mathcal{T}'$. $\mathbf{C}'$ cannot consist solely of branches of ${}^k\mathcal{T}$, since otherwise ${}^k\mathcal{T}$ would not be a k-tree. Let then c be an arc of $\mathbf{C}'$ which is not in ${}^k\mathcal{T}$. Since all branches of ${}^k\mathcal{T}'$ which are also chords of ${}^k\mathcal{T}$ have already been removed from ${}^k\mathcal{T}'$, c must be a connector of ${}^k\mathcal{T}$. Then $w(b) \leq w(c)$, since b and c are respectively branch and connector of ${}^k\mathcal{T}$ (4 holds for ${}^k\mathcal{T}$), and $w(b) \geq w(c)$, since b and c are also respectively chord and branch of the same fundamental circuit in ${}^k\mathcal{T}'$ (3 holds for ${}^k\mathcal{T}'$). Hence, $w(b) = w(c)$. Since ${}^k\mathcal{T}' - c + b$ is also a spanning k-tree and it has the same weight as ${}^k\mathcal{T}'$, the chord c of ${}^k\mathcal{T}$ can be substituted by the branch b of ${}^k\mathcal{T}$ in ${}^k\mathcal{T}'$.

Suppose now that b is a connector of ${}^k\mathcal{T}'$. Since there is a branch b of ${}^k\mathcal{T}$ not in ${}^k\mathcal{T}'$, there must be a branch c of ${}^k\mathcal{T}'$ not in ${}^k\mathcal{T}$. Since all chords of ${}^k\mathcal{T}$ which are branches of ${}^k\mathcal{T}'$ have already been removed, c must be a connector of ${}^k\mathcal{T}$. Then $w(b) \leq w(c)$, since b and c are respectively branch and connector of ${}^k\mathcal{T}$ (4 holds for ${}^k\mathcal{T}$), and $w(b) \geq w(c)$, since b and c are also respectively connector and branch of ${}^k\mathcal{T}'$ (4 holds for ${}^k\mathcal{T}'$). Hence, $w(b) = w(c)$. Since ${}^k\mathcal{T}' - c + b$ is also a spanning k-tree and it has the same weight as ${}^k\mathcal{T}'$, the chord c of ${}^k\mathcal{T}$ can be substituted by the branch b of ${}^k\mathcal{T}$ in ${}^k\mathcal{T}'$.

In either case, a branch c of ${}^k\mathcal{T}'$ can be substituted by a branch b of ${}^k\mathcal{T}$ in ${}^k\mathcal{T}'$. Repeating this process while there are branches of ${}^k\mathcal{T}$ which are either chords or connectors of ${}^k\mathcal{T}'$, such arcs are introduced into ${}^k\mathcal{T}'$ without changing its global weight $W(\mathbf{A}'_s, w)$. When the process ends, $\mathbf{A}_s \setminus \mathbf{A}'_s$ is empty. Since $\#\mathbf{A}_s = \#\mathbf{A}'_s$, this implies that $\mathbf{A}_s = \mathbf{A}'_s$. Since the substitutions performed on ${}^k\mathcal{T}'$ did not change its global weight, then one must conclude that ${}^k\mathcal{T}'$ is still a SSkT and ${}^k\mathcal{T} = {}^k\mathcal{T}'$ is indeed a SSkT. □

Theorem 3.19. *If ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ is a SSkT of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ and c is a connector of ${}^k\mathcal{T}$ with minimum weight, then ${}^k\mathcal{T} + c$ is a SSk $- 1$T of $\mathcal{G}$.*

Proof. Let $\mathbf{C}$ be the set of all connectors of ${}^k\mathcal{T}$, which by hypothesis is not empty (otherwise there would be no c). Let $c = \arg\min_{c \in \mathbf{C}} w(c)$. Then $w(c) \leq w(c')\ \forall c' \in \mathbf{C}$.

It is clear, from the proof of Theorem 3.4, that ${}^{k-1}\mathcal{T}' = {}^k\mathcal{T} + c$ is a spanning $k-1$-tree of $\mathcal{G}$. Hence, by Theorem 3.18, it suffices to show that the branch and connector conditions hold for ${}^{k-1}\mathcal{T}'$ in order to prove that ${}^{k-1}\mathcal{T}'$ is indeed a SS$k-1$T.

Since c is a connector of ${}^k\mathcal{T}$, it has end vertices in two different components of ${}^k\mathcal{T}$, say ${}^k\mathcal{T}_i$ and ${}^k\mathcal{T}_j$, with $i \neq j$. Let $\mathbf{C}_{ij}$ be the set of connectors between these two components. Let ${}^{k-1}\mathcal{T}'_{ij}$

be the union of ${}^{k}\mathcal{T}_i$ with ${}^{k}\mathcal{T}_j$ though c in ${}^{k-1}\mathcal{T}'$. The set $\mathbf{C}_{ij}$ is clearly the fundamental cutset corresponding to the branch c in ${}^{k-1}\mathcal{T}'$. Since c is a minimum weight connector of ${}^{k}\mathcal{T}$ and all arcs in $\mathbf{C}_{ij}$ are connectors of ${}^{k}\mathcal{T}$, it is clear that the branch condition is valid for branch c in ${}^{k-1}\mathcal{T}'$.

The arcs in $\mathbf{C}_{ij} \setminus \{c\}$ are clearly all chords of ${}^{k-1}\mathcal{T}'$, in particular of component ${}^{k-1}\mathcal{T}_{ij}$. Since these arcs are also connectors of ${}^{k}\mathcal{T}$, which is a SSkT, then $w(c) \geq w(b)$ $\forall c \in \mathbf{C}_{ij}$ and $\forall b \in \mathbf{A}_s$, and hence also for all branches b in ${}^{k}\mathcal{T}_i$ and ${}^{k}\mathcal{T}_j$. It is clear, then, that the fundamental cutsets of ${}^{k-1}\mathcal{T}'_{ij}$ also fulfill the branch condition, since all the new chords are heavier than all branches of ${}^{k-1}\mathcal{T}'_{ij}$, and the old chords already fulfilled the branch condition in ${}^{k}\mathcal{T}$, given that it is a SSkT. All the other components of ${}^{k}\mathcal{T}$ are unaffected, and hence also fulfill the branch condition.

The connector condition is also fulfilled for ${}^{k-1}\mathcal{T}'$, since there is only one new branch, c, which was chosen a minimum weight connector of ${}^{k}\mathcal{T}$, and the connectors of ${}^{k-1}\mathcal{T}'$ are $\mathbf{C} \setminus \mathbf{C}_{ij}$. □

Theorem 3.20. *Every SSkT of a graph with c connected components is a subgraph of some SS$k-1$T, provided that $k > c$.*

Proof. Since k is larger than the number of connected components of $\mathcal{G}$, there must be at least one connector of ${}^{k}\mathcal{T}$ in $\mathcal{G}$. Hence, it suffices to choose the connector with minimum weight and add it to the original SSkT. The result, by Theorem 3.19, is a SS$k-1$T. □

Theorem 3.21. *Every SSkT ${}^{k}\mathcal{T}$ of a graph $\mathcal{G}$ is a subgraph of some SSF of that graph.*

Proof. Let c be the number of connected components of $\mathcal{G}$. While $k > c$, it is possible, by Theorems 3.20 and 3.19, to construct a sequence of shortest spanning n-trees ${}^{n}\mathcal{T}$, with $n = k, \ldots, c$, where c is the number of connected components of the graph. It is clear that ${}^{n}\mathcal{T}$ is a subgraph of ${}^{m}\mathcal{T}$ whenever $n \geq m$. But ${}^{c}\mathcal{T}$ is a SSF of $\mathcal{G}$ (a spanning c-tree of $\mathcal{G}$ is a spanning forest of $\mathcal{G}$). Hence ${}^{k}\mathcal{T}$ is a subgraph of ${}^{c}\mathcal{T}$. □

Theorem 3.22. *If ${}^{k}\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ is a SSkT of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ and b is a branch of ${}^{k}\mathcal{T}$ with maximum weight, then ${}^{k}\mathcal{T} - b$ is a SS$k+1$T of $\mathcal{G}$.*

Proof. By hypothesis $\mathbf{A}_s$ is not empty (otherwise there would be no b). Since b is maximum, i.e., $b = \arg\max_{a \in \mathbf{A}_s} w(a)$, it is clear that $w(b) \geq w(b')$ $\forall b' \in \mathbf{A}_s$.

It is also clear, from the proof of Theorem 3.5, that ${}^{k+1}\mathcal{T}' = {}^{k}\mathcal{T} - b$ is a spanning $k+1$-tree of $\mathcal{G}$. Hence, by Theorem 3.18, it suffices to show that the branch and connector conditions hold for ${}^{k+1}\mathcal{T}'$ in order to prove that ${}^{k+1}\mathcal{T}'$ is indeed a SS$k+1$T.

Let $\mathbf{C}$ be the fundamental cutset of ${}^{k}\mathcal{T}$ relative to b. It consists of one branch, b, and the remaining arcs $\mathbf{C} \setminus \{b\}$ are chords. When b is removed from ${}^{k}\mathcal{T}$, the arcs in $\mathbf{C}$, including the branch b, change their role to connectors. The branch condition must still be true, since the only change to remaining branches is that they may have lost some chords in their fundamental cutsets.

The connector condition holds for the connectors of ${}^{k}\mathcal{T}$, which are also connectors of ${}^{k+1}\mathcal{T}'$, since the only change was the disappearance of b. Since b, now a connector, was chosen as a maximum weight branch of ${}^{k}\mathcal{T}$, the connector condition holds for b. It remains to be seen that the chords in $\mathbf{C} \setminus \{b\}$ are indeed not lighter than any remaining branch. Since the branch condition holds for ${}^{k}\mathcal{T}$, $w(b) \leq w(c)$ $\forall c \in \mathbf{C} \setminus \{b\}$. But b was chosen such that $w(b) \geq w(b')$ $\forall b' \in \mathbf{A}_s$. Hence, $w(c) \geq w(b)$ $\forall b \in \mathbf{A}_s$ and $\forall c \in \mathbf{C} \setminus \{b\}$, and the connector condition holds for ${}^{k+1}\mathcal{T}'$. □

Theorem 3.23. *Every SSkT of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ has a subgraph which is a SSk+1T, provided that $k < \#\mathbf{V}$.*

Proof. Since k is smaller than the number of vertices of $\mathcal{G}$, there must be at least one branch in ${}^{k}\mathcal{T}$. Hence, it suffices to choose the branch with maximum weight an remove it from the original SSkT. The result, by Theorem 3.22, is a SS$k+1$T. □

Theorem 3.24. *Given a SSF $\mathcal{F}(\mathbf{V}, \mathbf{A}_s) = {}^{c}\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ with c connected components, a SSkT may be obtained, provided that $k \leq \#\mathbf{V}$, by removing from $\mathcal{F}$ a set $\mathbf{B}$ of $k - c$ branches chosen so that $w(b) \geq w(b')$ $\forall b \in \mathbf{B}$ and $\forall b' \in \mathbf{A}_s \setminus \mathbf{B}$.*

Proof. Apply Theorem 3.22 repeatedly. □

Algorithms

Theorem 3.19 proves that, at step n of the Kruskal algorithm, the forest of selected branches is a SS$\#\mathbf{V} - n$T of the graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$.[9] This is so because, in Kruskal's algorithm, the arcs enter the forest in non-decreasing weight order, and only if they are connectors, i.e., if they connect two trees in the forest (chords are discarded). Hence, Theorem 3.19 is applicable at each step. Since the spanning $\#\mathbf{V}$-tree with no arcs is indeed a SS$\#\mathbf{V}$T, it is obvious, by induction, that at each step of the Kruskal algorithm one has a SSkT.

Also, if a SSF $\mathcal{F}$ of a graph $\mathcal{G}$ is available, one can cut forest branches successively, according to Theorem 3.24, and thus obtain a sequence of SSkT, with increasing k. Even though this method is not very efficient, it has a nice parallel with a similar algorithm which can be used to obtain SSSSkTs (to be defined later) of a graph. This type of algorithms will be called destructive, since they achieve the desired result by removing branches from a SSF, i.e., by destroying a SSF. The Kruskal algorithm, on the other hand, will be termed constructive.

Shortest seeded spanning k-trees

Definition 3.63. (shortest seeded spanning k-tree) *A n-seed respecting spanning k-tree of a graph is a SSSkT (Shortest Seeded Spanning k-Tree) of that graph if no other seeded spanning k-tree exists with a smaller weight.*

[9]At step 0, i.e., before the first branch is inserted, the forest has no arcs and hence has $\#\mathbf{V}$ components (trees).

Definition 3.64. (smallest shortest seeded spanning k-tree) *If a SSSkT is also smallest, i.e., k is as small as possible, then it will be called a SSSSkT (Smallest Shortest Seeded Spanning k-Tree).*

Theorem 3.25. *The connected components ${}^k\mathcal{T}_i(\mathbf{V}_i, \mathbf{A}_{s_i})$, with $i = 1, \ldots k$, of a SSSkT ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$, are SSTs of the subgraphs $\mathcal{G}_i$ induced in $\mathcal{G}$ by the corresponding set of vertices $\mathbf{V}_i$.*

Proof. Let $\mathcal{G}_i(\mathbf{V}_i, \mathbf{A}_i)$ be the subgraph of $\mathcal{G}$ induced by the vertices $\mathbf{V}_i$ of a tree ${}^k\mathcal{T}_i$ of ${}^k\mathcal{T}$. Suppose ${}^k\mathcal{T}_i$ is not a SST of $\mathcal{G}_i$. Then it is possible to chose another tree spanning $\mathcal{G}_i$ with a smaller weight than ${}^k\mathcal{T}_i$ (choose a SST of $\mathcal{G}_i$), without changing the weight associated to the other components of ${}^k\mathcal{T}$ and thereby reducing the weight of ${}^k\mathcal{T}$. But this is a contradiction, since $\mathcal{T}_k$ is a SSSkT of $\mathcal{G}$. Hence, ${}^k\mathcal{T}_i$ is indeed a SST of $\mathcal{G}_i$, for $i = 1, \ldots, k$. □

Lemma 3.26. *Consider two (unique) paths $\mathbf{P} = v_0, v_f$-path $\subseteq \mathbf{A}$ and $\mathbf{P}' = v_0', v_f$-path $\subseteq \mathbf{A}$ in a tree $\mathcal{T}(\mathbf{V}, \mathbf{A})$. If $v_0 \neq v_0'$, then the ring sum $\mathbf{P} \oplus \mathbf{P}'$ of $\mathbf{P}$ and $\mathbf{P}'$ is a v_0, v_0'-path in $\mathcal{T}$.*

Proof. Let v be the first vertex in common between $\mathbf{P}$ and $\mathbf{P}'$, starting in v_0 and v_0', respectively. Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be the segments of $\mathbf{P}$ before and after v, respectively, i.e., $\mathbf{P}_1 = v_0, v$-path and $\mathbf{P}_2 = v, v_f$-path. Define $\mathbf{P}_1'$ and $\mathbf{P}_2'$ similarly.[10] Clearly, $\mathbf{P} = \mathbf{P}_1 \cup \mathbf{P}_2$ and $\mathbf{P}' = \mathbf{P}_1' \cup \mathbf{P}_2'$. Since in a tree there is a single path between any two vertices, see Theorem 3.1, it is obvious that $\mathbf{P}_2 = \mathbf{P}_2'$. On the other hand, by construction, $\mathbf{P}_1 \cap \mathbf{P}_1' = \emptyset$. Hence, $\mathbf{P} \oplus \mathbf{P}' = \mathbf{P}_1 \cup \mathbf{P}_1'$, that is, a path between v_0 and v_0'. □

Lemma 3.27. *If $\mathbf{P} \subseteq \mathbf{A}$ is a v_0, v_f-path and $\mathbf{P}' \subseteq \mathbf{A}$ is a v_0, v_f-path, i.e., both are paths between the same end vertices in a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$, then the ring sum $\mathbf{P} \oplus \mathbf{P}'$ of $\mathbf{P}$ and $\mathbf{P}'$ is either empty, a circuit, or a union of arc-disjoint circuits of $\mathcal{G}$. Moreover, $\mathbf{P} \oplus \mathbf{P}'$ is empty only if $\mathbf{P} = \mathbf{P}'$ and all circuits in $\mathbf{P} \oplus \mathbf{P}'$ contain arcs from both paths.*

Proof. Consider the subgraphs $\mathcal{G}_{\mathbf{P}}(\mathbf{V}_{\mathbf{P}}, \mathbf{P})$ and $\mathcal{G}_{\mathbf{P}'}(\mathbf{V}_{\mathbf{P}'}, \mathbf{P}')$ of $\mathcal{G}$ induced by $\mathbf{P}$ and $\mathbf{P}'$, respectively. Let $\mathcal{G}_{\mathbf{P}\oplus\mathbf{P}'}(\mathbf{V}_{\mathbf{P}\oplus\mathbf{P}'}, \mathbf{P} \oplus \mathbf{P}')$ be the subgraph of $\mathcal{G}$ induced by $\mathbf{P} \oplus \mathbf{P}'$. Let $v \in \mathbf{V}_{\mathbf{P}} \cup \mathbf{V}_{\mathbf{P}}'$. If $v = v_0$, then its degree in $\mathcal{G}_{\mathbf{P}\oplus\mathbf{P}'}$ will either be 0, if both paths share the same arc incident on v_0, or 2 otherwise. The same goes for $v = v_f$. If $v \neq v_o, v_f$, then its degree will either be 0, if both paths share the same pair of arcs incident on v, 2 if both paths share a single arc incident on v, or 4, if the arcs on both paths which are incident on v are all different. Clearly, vertices of degree 0 do not belong to $\mathcal{G}_{\mathbf{P}\oplus\mathbf{P}'}$. Hence, graph $\mathcal{G}_{\mathbf{P}\oplus\mathbf{P}'}$ is either empty or it contains only vertices with degrees 2 and 4, i.e., it is either a circuit or a union of arc-disjoint circuits.

That the circuits contain arcs from both paths is obvious, for otherwise one of the paths would contain a circuit, which cannot happen by definition. □

Lemma 3.28. *Let ${}^k\mathcal{T}$ be a seeded spanning k-tree of graph $\mathcal{G}$. Let $\mathbf{C}$ be a circuit in $\mathcal{G}$. If $\mathbf{C}$ contains no connectors, then $\mathbf{C}$ either contains no separators or it contains at least two separators.*

[10]The fact that one of these paths can be empty in no way invalidates the rest of the arguments.

Proof. Let circuit $\mathbf{C}$ in graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ consist of the sequence $v_0, a_1, v_1, \ldots, v_k$, with $v_0 = v_k$. Let v_0 belong to component ${}^k\mathcal{T}_i(\mathbf{V}_i, \mathbf{A}_{s_i})$ of ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$. This circuit consists of branches, chords, and separators of ${}^k\mathcal{T}$. Only the separators connect vertices of different components of ${}^k\mathcal{T}$. Hence, if a_l is the first separator in the sequence above, $v_i \in \mathbf{V}_i$ with $i < l$. The separator a_l connects component ${}^k\mathcal{T}_i$ with some component ${}^k\mathcal{T}_j$, with $i \neq j$. If there is no other separator in the sequence, then one similarly has to conclude that $v_i \in \mathbf{V}_j$ with $i \geq l$. But in that case $v_0 = v_k \in \mathbf{V}_j$, which is a contradiction, since $v_0 \in \mathbf{V}_i$, and $\mathbf{V}_i \cap \mathbf{V}_j = \emptyset$ for $i \neq j$. Hence, there must be another separator in the sequence. □

Lemma 3.29. *Let $\mathbf{C}$ be a circuit in graph $\mathcal{G}$. Let ${}^k\mathcal{T}$ be a seeded k-tree of $\mathcal{G}$. If $b \in \mathbf{C}$ is a branch of ${}^k\mathcal{T}$, then either:*

1. *there is a connector c of ${}^k\mathcal{T}$ in $\mathbf{C}$; or*
2. *b belongs to a fundamental path of some separator c of ${}^k\mathcal{T}$ in $\mathbf{C}$; or*
3. *b belongs to the fundamental circuit of some chord c of ${}^k\mathcal{T}$ in $\mathbf{C}$.*

Proof. The circuit $\mathbf{C}$ cannot consist solely of branches of ${}^k\mathcal{T}$, since a k-tree does not contain any circuits. Hence, $\mathbf{C}$ must contain at least one connector, separator or chord of ${}^k\mathcal{T}$.

If $\mathbf{C}$ contains at least one connector of ${}^k\mathcal{T}$, then 1 holds.

If $\mathbf{C}$ does not contain any connector of ${}^k\mathcal{T}$, then it must contain at least one separator or chord.

If it contains a separator, then, by Lemma 3.28, it contains at least two such separators. Let ${}^k\mathcal{T}_i$ be the component of ${}^k\mathcal{T}$ containing b. Let b belong to a path $\mathbf{P}$ between two separators in $\mathbf{C}$ (it is clear that b must belong to one such paths). Let c_1 and c_2 be the corresponding separators and let v_1 and v_2 be the two vertices in ${}^k\mathcal{T}_i$ which are end vertices of c_1 and c_2, respectively. Clearly, $\mathbf{P}$ is a v_1, v_2-path. If $\mathbf{P}_1$ and $\mathbf{P}_2$ are the restrictions to ${}^k\mathcal{T}_i$ of the fundamental paths associated to c_1 and c_2, then it is clear that $\mathbf{P}_1 = v_1, s_i$-path and $P_2 = v_2, s_i$-path, where s_i is the seed of ${}^k\mathcal{T}_i$ in ${}^k\mathcal{T}$. Hence, by Lemma 3.26, $\mathbf{P}_{12} = \mathbf{P}_1 \oplus \mathbf{P}_2 = v_1, v_2$-path is a path between v_1 and v_2 in ${}^k\mathcal{T}_i$. Now either b belongs to $\mathbf{P}_1$ or $\mathbf{P}_2$, and 2 holds, or it does not. Suppose it does not. Then paths $\mathbf{P}$ and $\mathbf{P}_{12}$ between v_1 and v_2 are different, given that b belongs to the former but not to the latter. By Lemma 3.27, $\mathbf{P} \oplus \mathbf{P}_{12}$ is a circuit or a sum of arc-disjoint circuits of $\mathcal{G}_i$, the subgraph of $\mathcal{G}$ induced by the vertices of ${}^k\mathcal{T}_i$. Moreover, $b \in \mathbf{P} \oplus \mathbf{P}_{12}$, since it does not belong to $\mathbf{P}_{12}$. It is clear that b belongs to one of the arc-disjoint circuits in $\mathbf{P} \oplus \mathbf{P}_{12}$. Let it be circuit $\mathbf{C}'$ in $\mathcal{G}_i$. $\mathbf{C}'$ can be written as the ring sum of the fundamental circuits corresponding to the chords of ${}^k\mathcal{T}_i$ in $\mathbf{C}'$ and b occurs in an odd number of these fundamental circuits. Notice that the chords of these fundamental circuits all belong to $\mathbf{P}$, and hence to $\mathbf{C}$, since $\mathbf{P}_{12}$ contains only branches of ${}^k\mathcal{T}_i$. That is, 3 holds.

If $\mathbf{C}$ does not contain any separator, then it consists solely of branches and chords. Hence, by Lemma 3.17, it can be expressed as a ring sum of the fundamental circuits of ${}^k\mathcal{T}$ corresponding to the chords of ${}^k\mathcal{T}$ in $\mathbf{C}$ and b occurs in an odd number of these fundamental circuits. That is, 3 holds. □

Lemma 3.30. *Let ${}^k\mathcal{T}$ be a seeded k-tree of a graph $\mathcal{G}$. Let $\mathbf{P}$ be a path in $\mathcal{G}$ connecting two seeds s_1 and s_2, i.e., $\mathbf{P}$ is a s_1, s_2-path. If $\mathbf{P}$ contains no connectors of ${}^k\mathcal{T}$, then it must contain at least one separator.*

Proof. Suppose $\mathbf{P}$, which has no connectors of ${}^k\mathcal{T}$, also does not have separators of ${}^k\mathcal{T}$. Then $\mathbf{P}$ consists only of chords and branches of ${}^k\mathcal{T}$. Hence, by a reasoning similar to the one in the proof of Lemma 3.28, one has to conclude that s_2 belongs to the same component of ${}^k\mathcal{T}$ as s_1. But this is a contradiction, since ${}^k\mathcal{T}$ respects seed separation by hypothesis. Hence, there must be separators in $\mathbf{P}$. □

Lemma 3.31. *Let ${}^k\mathcal{T}$ be a seeded k-tree of a graph $\mathcal{G}$. Let $\mathbf{P}$ be a path in $\mathcal{G}$ connecting two seeds s_1 and s_2, i.e., $\mathbf{P}$ is a s_1, s_2-path. If $b \in \mathbf{P}$ is a branch of ${}^k\mathcal{T}$, then either:*

1. *there is a connector c of ${}^k\mathcal{T}$ in $\mathbf{P}$; or*
2. *b belongs to a fundamental path of some separator c of ${}^k\mathcal{T}$ in $\mathbf{P}$; or*
3. *b belongs to the fundamental circuit of some chord c of ${}^k\mathcal{T}$ in $\mathbf{P}$.*

Proof. If $\mathbf{P}$ contains at least one connector of ${}^k\mathcal{T}$, then 1 holds.

If $\mathbf{P}$ does not contain any connectors, then, by Lemma 3.30, it must contain at least one separator of ${}^k\mathcal{T}$. Removal of the separators of ${}^k\mathcal{T}$ from $\mathbf{P}$ segments the path into a series of shorter paths each inside a single component of ${}^k\mathcal{T}$. Branch b belongs to one of these segments.

If branch b belongs to a segment of path $\mathbf{P}_{12}$ between two separators c_1 and c_2 of ${}^k\mathcal{T}$, then, using the same arguments as in the proof of Lemma 3.29, either b belongs to a fundamental path of one of the separators $c_1, c_2 \in \mathbf{P}$, and 2 holds, or b belongs to a fundamental circuit of some chord $c \in \mathbf{P}_{12} \subseteq \mathbf{P}$, and 3 holds.

Otherwise, b belongs to a segment $\mathbf{P}'$ of path $\mathbf{P}$ between a seed s_i and a separator c'. This segment contains only branches or chords of ${}^k\mathcal{T}$. Hence, it is contained in the component ${}^k\mathcal{T}_i$ of ${}^k\mathcal{T}$ to which s_i belongs. Consider the restriction $\mathbf{P}''$ of the fundamental path of ${}^k\mathcal{T}$ corresponding to c' to the component ${}^k\mathcal{T}_i$. Clearly, $\mathbf{P}''$ contains only branches of ${}^k\mathcal{T}$. It is also clear that both paths connect s_i and the end vertex v_i of c' inside ${}^k\mathcal{T}_i$. If b belongs to $\mathbf{P}''$, then 2 holds. Otherwise, b is in $\mathbf{P}' \oplus \mathbf{P}''$, which, according to Lemma 3.27, is a circuit or a sum of circuits. Hence, b must be in the fundamental circuit of some chord c in $\mathbf{P}' \oplus \mathbf{P}''$. Since $\mathbf{P}''$ contains only branches of ${}^k\mathcal{T}$, c must belong to $\mathbf{P}'$ and hence to $\mathbf{P}$. That is, 3 holds. □

Theorem 3.32. (branch, chord, connector, and separator conditions) *Let ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ be a spanning k-tree of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ respecting the set $\mathbf{S} = \{s_1, \ldots, s_n\} \subseteq \mathbf{V}$ of n seed vertices of $\mathcal{G}$. Consider the following statements:*

1. *${}^k\mathcal{T}$ is a SSSkT of $\mathcal{G}$.*
2. *(branch condition) $w(b) \leq w(c)$ for any branch b of ${}^k\mathcal{T}$ and for any chord c in the corresponding fundamental cutset relative to ${}^k\mathcal{T}$.*
3. *(chord condition) $w(b) \leq w(c)$ for any chord c of ${}^k\mathcal{T}$ and for any branch b in the corresponding fundamental circuit relative to ${}^k\mathcal{T}$.*
4. *(connector condition) $w(c) \geq w(b)$ for any branch b and for any connector c of ${}^k\mathcal{T}$.*
5. *(separator condition) $w(c) \geq w(b)$ for any separator c of ${}^k\mathcal{T}$ and for any branch b in the corresponding fundamental path relative to ${}^k\mathcal{T}$.*

If a seeded spanning k-tree ${}^k\mathcal{T}$ is also a SSSkT, then branch, chord, connector, and separator *conditions are true for $^k\mathcal{T}$, i.e.,*

$$1 \Rightarrow 2 \wedge 3 \wedge 4 \wedge 5.$$

On the other hand, the connector and separator conditions together with either the branch or the chord conditions are sufficient to guarantee that a spanning k-tree is a SSSkT, i.e.,

$$\begin{aligned}&2 \wedge 4 \wedge 5 \Rightarrow 1, \textit{ and}\\ &3 \wedge 4 \wedge 5 \Rightarrow 1.\end{aligned}$$

Proof. It will first be shown that $2 \Leftrightarrow 3$. Then, it suffices to show that $1 \Rightarrow 3$, $1 \Rightarrow 4$, $1 \Rightarrow 5$, and $3 \wedge 4 \wedge 5 \Rightarrow 1$.

Let $\mathcal{G}_i$ be the subgraph of $\mathcal{G}$ induced by the vertices of the connected component $^k\mathcal{T}_i$ of $^k\mathcal{T}$.

$2 \Leftrightarrow 3$, $1 \Rightarrow 3$, $1 \Rightarrow 4$: The arguments are similar to the ones used in the proof of Theorem 3.18. Notice that the derived spanning k-tree is still respecting of the set of seeds in all cases.

$1 \Rightarrow 5$ (or $\neg 5 \Rightarrow \neg 1$): Let c be a separator of a seeded spanning k-tree $^k\mathcal{T}$ and $\mathbf{P}$ its corresponding fundamental path. If there is any $b \in \mathbf{P}$ such that $w(b) > w(c)$, then $^k\mathcal{T} - b + c$, which is clearly still a seeded spanning k-tree of $\mathcal{G}$, has a smaller weight than $^k\mathcal{T}$. Hence, $^k\mathcal{T}$ cannot be a SSSkT.

$3 \wedge 4 \wedge 5 \Rightarrow 1$: Assume 3, 4, and 5 hold for $^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$. Let $^k\mathcal{T}'(\mathbf{V}, \mathbf{A}'_s)$ be a SSSkT of $\mathcal{G}$, for which, as was already proven, 3, 4 and 5 hold. It will be shown that $^k\mathcal{T}'$ can be made equal to $^k\mathcal{T}$ through a series of simple operations which neither increases nor decreases the total weight $W(\mathbf{A}'_s, w)$. This proves that $^k\mathcal{T}$ is indeed a SSSkT, since $W(\mathbf{A}_s, w) = W(\mathbf{A}'_s, w)$.

If $^k\mathcal{T}$ and $^k\mathcal{T}'$ are equal, then $^k\mathcal{T}$ is also a SSkT. Suppose then that $^k\mathcal{T}$ and $^k\mathcal{T}'$ are different. Since $^k\mathcal{T}$ and $^k\mathcal{T}'$ are both spanning k-trees, both contain the same number of arcs of $\mathcal{G}$: $\#\mathbf{A}_s = \#\mathbf{A}'_s$. Hence, there is an equal non-zero number of arcs in $\mathbf{A}_s \setminus \mathbf{A}'_s$ and $\mathbf{A}'_s \setminus \mathbf{A}_s$. The arcs in $\mathbf{A}'_s \setminus \mathbf{A}_s$ are chords, connectors or separators of $^k\mathcal{T}$ and branches of $^k\mathcal{T}'$ and vice versa.

i. Suppose there is a chord c of $^k\mathcal{T}$ which is also a branch of $^k\mathcal{T}'$. Let $\mathbf{C}$ be the corresponding fundamental circuit in $^k\mathcal{T}$. Since 3 holds for $^k\mathcal{T}$, $w(c) \geq w(b)$ for all $b \in \mathbf{C}$. By Lemma 3.29, either:

   1. there is a connector b of $^k\mathcal{T}'$ in $\mathbf{C}$, in which case, since 4 holds for $^k\mathcal{T}'$, $w(b) \geq w(c)$;
   2. c belongs to the fundamental path of some separator b of $^k\mathcal{T}'$ in $\mathbf{C}$, in which case, since 5 holds for $^k\mathcal{T}'$, $w(b) \geq w(c)$; or
   3. c belongs to the fundamental circuit of some chord b of $^k\mathcal{T}'$ in $\mathbf{C}$, in which case, since 3 holds for $^k\mathcal{T}'$, $w(b) \geq w(c)$.

   In either case, both $w(c) \geq w(b)$ and $w(c) \leq w(b)$, that is, $w(b) = w(c)$ and $^k\mathcal{T}' - c + b$ is still a seeded spanning tree respecting the same set of seeds. That is, in either case, the branch c of $^k\mathcal{T}'$ can be substituted by the branch b of $^k\mathcal{T}$ in $^k\mathcal{T}'$. Repeating this process while there are branches of $^k\mathcal{T}'$ which are chords of $^k\mathcal{T}$, such arcs are eliminated from $^k\mathcal{T}'$ without changing its global weight $W(\mathbf{A}'_s, w)$.

   If $^k\mathcal{T}' = {}^k\mathcal{T}$ after this process, then $^k\mathcal{T}$ is indeed a SSSkT. If not, then there must be some branch b of $^k\mathcal{T}$ which is not in $^k\mathcal{T}'$ and vice versa.

ii. Suppose there is a branch b of ${}^k\mathcal{T}$ which is also a chord of ${}^k\mathcal{T}'$. Let $\mathbf{C}'$ be its corresponding fundamental circuit in ${}^k\mathcal{T}'$. Since 3 holds for ${}^k\mathcal{T}'$, $w(b) \geq w(c)$ for all $c \in \mathbf{C}'$. By Lemma 3.29, either:

   1. there is a connector c of ${}^k\mathcal{T}$ in $\mathbf{C}'$, in which case, since 4 holds for ${}^k\mathcal{T}$, $w(c) \geq w(b)$;
   2. b belongs to the fundamental path of some separator c of ${}^k\mathcal{T}$ in $\mathbf{C}'$, in which case, since 5 holds for ${}^k\mathcal{T}$, $w(c) \geq w(b)$.

   Notice that case 3 of Lemma 3.29 cannot happen, since all branches of ${}^k\mathcal{T}'$ which are chords of ${}^k\mathcal{T}$ have already been removed (see step i. above).

   In either case, both $w(b) \geq w(c)$ and $w(b) \leq w(c)$, that is, $w(b) = w(c)$ and ${}^k\mathcal{T}' - c + b$ is still a seeded spanning tree respecting the same set of seeds. That is, in either case, the branch c of ${}^k\mathcal{T}'$ can be substituted by the branch b of ${}^k\mathcal{T}$ in ${}^k\mathcal{T}'$. Repeating this process while there are branches of ${}^k\mathcal{T}$ which are chords of ${}^k\mathcal{T}'$, such arcs are introduced into ${}^k\mathcal{T}'$ without changing its global weight $W(\mathbf{A}'_s, w)$.

   If ${}^k\mathcal{T}' = {}^k\mathcal{T}$ after this process, then ${}^k\mathcal{T}$ is indeed a SSSkT. If not, then there must be some branch b of ${}^k\mathcal{T}$ which is not in ${}^k\mathcal{T}'$ and vice versa.

iii. Suppose there is some separator c of ${}^k\mathcal{T}$ which is also a branch of ${}^k\mathcal{T}'$. Let $\mathbf{P}$ be its corresponding fundamental path in ${}^k\mathcal{T}$. Since 5 holds for ${}^k\mathcal{T}$, $w(c) \geq w(b)$ for all $b \in \mathbf{P}$. By Lemma 3.31, either:

   1. there is a connector b of ${}^k\mathcal{T}'$ in $\mathbf{P}$, in which case, since 4 holds for ${}^k\mathcal{T}'$, $w(b) \geq w(c)$;
   2. c belongs to the fundamental path of some separator b of ${}^k\mathcal{T}'$ in $\mathbf{P}$, in which case, since 5 holds for ${}^k\mathcal{T}'$, $w(b) \geq w(c)$.

   Notice that case 3 of Lemma 3.31 cannot happen, since all branches of ${}^k\mathcal{T}$ which are chords of ${}^k\mathcal{T}'$ have already been removed (see step ii. above).

   In either case, both $w(c) \geq w(b)$ and $w(c) \leq w(b)$, that is, $w(b) = w(c)$ and ${}^k\mathcal{T}' - c + b$ is still a seeded spanning tree respecting the same set of seeds. That is, in either case, the branch c of ${}^k\mathcal{T}'$ can be substituted by the branch b of ${}^k\mathcal{T}$ in ${}^k\mathcal{T}'$. Repeating this process while there are branches of ${}^k\mathcal{T}'$ which are separators of ${}^k\mathcal{T}$, such arcs are eliminated from ${}^k\mathcal{T}'$ without changing its global weight $W(\mathbf{A}'_s, w)$.

   If ${}^k\mathcal{T}' = {}^k\mathcal{T}$ after this process, then ${}^k\mathcal{T}$ is indeed a SSSkT. If not, then there must be some branch b of ${}^k\mathcal{T}$ which is not in ${}^k\mathcal{T}'$.

iv. Suppose there is some branch b of ${}^k\mathcal{T}$ which is also a separator of ${}^k\mathcal{T}'$. Let $\mathbf{P}'$ be its corresponding fundamental circuit in ${}^k\mathcal{T}'$. Since 5 holds for ${}^k\mathcal{T}'$, $w(b) \geq w(c)$ for all $c \in \mathbf{P}'$. By Lemma 3.31:

   1. there is a connector c of ${}^k\mathcal{T}$ in $\mathbf{P}'$, in which case, since 4 holds for ${}^k\mathcal{T}$, $w(c) \geq w(b)$;

   Notice that case 3 of Lemma 3.31 cannot happen, since all branches of ${}^k\mathcal{T}'$ which are chords of ${}^k\mathcal{T}$ have already been removed (see step i. above). Also, case 2 of Lemma 3.31 cannot happen, since all branches of ${}^k\mathcal{T}'$ which are separators of ${}^k\mathcal{T}$ have already been removed (see step iii. above).

   In either case, both $w(b) \geq w(c)$ and $w(b) \leq w(c)$, that is, $w(b) = w(c)$ and ${}^k\mathcal{T}' - c + b$ is still a seeded spanning tree respecting the same set of seeds. That is, in either case, the branch c of ${}^k\mathcal{T}'$ can be substituted by the branch b of ${}^k\mathcal{T}$ in ${}^k\mathcal{T}'$. Repeating this process while there are branches of ${}^k\mathcal{T}$ which are separators of ${}^k\mathcal{T}'$, such arcs are eliminated from ${}^k\mathcal{T}'$ without changing its global weight $W(\mathbf{A}'_s, w)$.

If ${}^k\mathcal{T}' = {}^k\mathcal{T}$ after this process, then ${}^k\mathcal{T}$ is indeed a SSSkT. If not, then there must be some branch b of ${}^k\mathcal{T}$ which is not in ${}^k\mathcal{T}'$.

v. At this point, if b is a branch of ${}^k\mathcal{T}$ which is not in ${}^k\mathcal{T}'$, then b is a connector of ${}^k\mathcal{T}'$, since all branches of ${}^k\mathcal{T}$ which were chords and separators of ${}^k\mathcal{T}'$ have been eliminated in steps ii. and iv. Conversely, if c is a branch of ${}^k\mathcal{T}'$ which is not in ${}^k\mathcal{T}$, then c is a connector of ${}^k\mathcal{T}$, since all branches of ${}^k\mathcal{T}'$ which were chords and separators of ${}^k\mathcal{T}$ have been eliminated in steps i. and iii. Moreover, for each b as stated, there is a c (remember that ${}^k\mathcal{T}$ and ${}^k\mathcal{T}'$ have the same number of arcs). Thus, both $w(c) \geq w(b)$ and $w(b) \geq w(c)$, i.e., $w(c) = w(b)$. Also, ${}^k\mathcal{T}' - c + b$ is still a seeded spanning tree respecting the same set of seeds. That is, the branch c of ${}^k\mathcal{T}'$ can be substituted by the branch b of ${}^k\mathcal{T}$ in ${}^k\mathcal{T}'$. Repeating this process while there are branches of ${}^k\mathcal{T}$ which are connectors of ${}^k\mathcal{T}'$, such arcs are eliminated from ${}^k\mathcal{T}'$ without changing its global weight $W(\mathbf{A}'_s, w)$.

After all the above steps, it is obvious that ${}^k\mathcal{T}' = {}^k\mathcal{T}$. Hence, since the weight of ${}^k\mathcal{T}'$ never changed along the process, ${}^k\mathcal{T}$ is indeed a SSSkT of $\mathcal{G}$. □

Corollary 3.33. *A SSSkT is also a SSSSkT iff it has no connectors and the chord (or branch) and separator conditions hold (see Theorem 3.32).*

Proof. The result is immediate from Theorems 3.8 and 3.32. □

Theorem 3.34. *If ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ is a SSSkT of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ and c is a connector of ${}^k\mathcal{T}$ with minimum weight, then ${}^k\mathcal{T} + c$ is a SSSk $-$ 1T of $\mathcal{G}$.*

Proof. Let $\mathbf{C}$ be the set of connectors of ${}^k\mathcal{T}$, which by hypothesis is not empty (otherwise there would be no c). Since c is a minimum weight connector, then $w(c) \leq w(c')$ $\forall c' \in \mathbf{C} \setminus \{c\}$.

It is clear, from the proof of Theorem 3.9, that ${}^{k-1}\mathcal{T}' = {}^k\mathcal{T} + c$ is a seeded spanning $k-1$-tree of $\mathcal{G}$ respecting the same set of seeds. Hence, by Theorem 3.32, it suffices to show that branch, connector, and separator conditions hold for ${}^{k-1}\mathcal{T}'$ in order to prove that ${}^{k-1}\mathcal{T}'$ is indeed a SSS$k-1$T.

Since c is a connector of ${}^k\mathcal{T}$, it has end vertices in two different components of ${}^k\mathcal{T}$, say ${}^k\mathcal{T}_i$ and ${}^k\mathcal{T}_j$, with $i \neq j$. Let $\mathbf{C}_{ij}$ be the set of connectors between these two components. Let ${}^{k-1}\mathcal{T}'_{ij}$ be the union of ${}^k\mathcal{T}_i$ with ${}^k\mathcal{T}_j$ through c in ${}^{k-1}\mathcal{T}'$. The set $\mathbf{C}_{ij}$ is clearly the fundamental cutset corresponding to the branch c in ${}^{k-1}\mathcal{T}'$. Since c is a minimum weight connector of ${}^k\mathcal{T}$ and all arcs in $\mathbf{C}_{ij}$ are connectors of ${}^k\mathcal{T}$, it is clear that the branch condition is valid for branch c in ${}^{k-1}\mathcal{T}'$.

The arcs in $\mathbf{C}_{ij} \setminus \{c\}$ are clearly all chords of ${}^{k-1}\mathcal{T}'$, in particular of component ${}^{k-1}\mathcal{T}_{ij}$. Since these arcs are also connectors of ${}^k\mathcal{T}$, which is a SSSkT, then $w(c) \geq w(b)$ $\forall c \in \mathbf{C}_{ij}$ and $\forall b \in \mathbf{A}_s$, and hence for all branches b in ${}^k\mathcal{T}_i$ and ${}^k\mathcal{T}_j$. It is clear, then, that the fundamental cutsets of ${}^{k-1}\mathcal{T}'_{ij}$ also fulfill the branch condition, since all the new chords are heavier than all branches of

${}^{k-1}\mathcal{T}'_{ij}$, and the old chords already fulfilled the branch condition in ${}^{k}\mathcal{T}$, given that it is a SSSkT. All the other components of ${}^{k}\mathcal{T}$ are unaffected, and hence also fulfill the branch condition.

The connector condition is also fulfilled for ${}^{k-1}\mathcal{T}'$, since there is only one new branch, c, which was chosen a minimum weight connector of ${}^{k}\mathcal{T}$, and the connectors of ${}^{k-1}\mathcal{T}'$ are $\mathbf{C} \setminus \mathbf{C}_{ij}$ minus possibly some new separators (see below).

It remains to check whether the separator condition still holds. It is clear that separators of ${}^{k}\mathcal{T}$ are also separators of ${}^{k-1}\mathcal{T}'$ with the same fundamental paths. Since ${}^{k}\mathcal{T}$ is a SSSkT, the separator condition holds for the separators of ${}^{k-1}\mathcal{T}'$ which already were separators of ${}^{k}\mathcal{T}$. However, the union of ${}^{k}\mathcal{T}_i$ with ${}^{k}\mathcal{T}_j$ through c may have created some new separators.

If both ${}^{k}\mathcal{T}_i$ and ${}^{k}\mathcal{T}_j$ are seedless in ${}^{k}\mathcal{T}$, then no new separators were created, and hence the separator condition holds for ${}^{k-1}\mathcal{T}'$.

Otherwise, only one of ${}^{k}\mathcal{T}_i$ and ${}^{k}\mathcal{T}_j$ may have contained a seed. Without loss of generality, suppose it is ${}^{k}\mathcal{T}_i$. Then all connectors of ${}^{k}\mathcal{T}_j$, which is seedless in ${}^{k}\mathcal{T}$, to some seeded component of ${}^{k}\mathcal{T}$ will become separators. But, since these connectors of ${}^{k}\mathcal{T}$ fulfill the connector condition, they are heavier than any branch in ${}^{k}\mathcal{T}$. Hence, they are also heavier than any branch in their corresponding fundamental paths in ${}^{k-1}\mathcal{T}'$. Hence, the separator condition holds for ${}^{k-1}\mathcal{T}'$. □

Corollary 3.35. *All SSSkT which are not smallest are subgraphs of some SSS$k-1$T.*

Proof. By Theorem 3.8, there must be at least one connector in ${}^{k}\mathcal{T}$. The result is immediate from Theorem 3.34. □

Theorem 3.36. *If ${}^{k}\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ is a SSSkT of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ and b is a branch of ${}^{k}\mathcal{T}$ with maximum weight, then ${}^{k}\mathcal{T} - b$ is a SSS$k+1$T of $\mathcal{G}$ for the same set of seeds.*

Proof. By hypothesis $\mathbf{A}_s$ is not empty (otherwise there would be no b). Since b is maximum, i.e., $b = \arg\max_{a \in \mathbf{A}_s} w(a)$, it is clear that $w(b) \geq w(b')\ \forall b' \in \mathbf{A}_s$.

It is also clear, from Theorem 3.10, that ${}^{k+1}\mathcal{T}' = {}^{k}\mathcal{T} - b$ is a seeded spanning $k+1$-tree of $\mathcal{G}$ for the same set of seeds. Hence, by Theorem 3.32, it suffices to show that the branch, connector, and separator conditions hold for ${}^{k+1}\mathcal{T}'$ in order to prove that ${}^{k+1}\mathcal{T}'$ is indeed a SSS$k+1$T.

Let $\mathbf{C}$ be the fundamental cutset of ${}^{k}\mathcal{T}$ relative to b. It consists of one branch, b, and the remaining arcs $\mathbf{C} \setminus \{b\}$ are chords. When b is removed from ${}^{k}\mathcal{T}$, the arcs in $\mathbf{C}$, including the branch b, change its role to connectors, since at most one of the resulting connected components is seeded. The branch condition must still be true, since the only change to remaining branches is that they may have lost some chords in their fundamental cutsets.

The connector condition holds for the connectors of ${}^{k}\mathcal{T}$, which are also connectors of ${}^{k+1}\mathcal{T}'$, since the only change was the disappearance of b. Since b, now a connector, was chosen as a maximum weight branch of ${}^{k}\mathcal{T}$, the connector condition holds for b. It remains to be seen that the chords in $\mathbf{C} \setminus \{c\}$ are indeed not lighter than any remaining branch. Since the branch condition holds for ${}^{k}\mathcal{T}$, $w(b) \leq w(c)\ \forall c \in \mathbf{C} \setminus \{b\}$. But b was chosen such that $w(b) \geq w(b')$ $\forall b' \in \mathbf{A}_s$. Hence, $w(c) \geq w(b)\ \forall b \in \mathbf{A}_s$ and $\forall c \in \mathbf{C} \setminus \{b\}$.

The only further change to the class of arcs is that some separators in ${}^{k}\mathcal{T}$ may have changed to connectors in ${}^{k+1}\mathcal{T}'$. Let ${}^{k+1}\mathcal{T}_i'$ and ${}^{k+1}\mathcal{T}_j'$ be the two components which resulted from removing b from its component ${}^{k}\mathcal{T}_l$ in ${}^{k}\mathcal{T}$.

If ${}^{k}\mathcal{T}_l$ is seedless, so are ${}^{k+1}\mathcal{T}_i'$ and ${}^{k+1}\mathcal{T}_j'$, and no separators changed to connectors.

Otherwise, suppose, without loss of generality, that ${}^{k+1}\mathcal{T}_i'$ contains the seed of ${}^{k}\mathcal{T}_l$. Then, separators connecting to vertices in ${}^{k+1}\mathcal{T}_j'$ are now connectors. The fundamental paths of these separators all included the branch b, since there is only one path between two vertices in a tree and the seed is on ${}^{k+1}\mathcal{T}_j'$. Since these separators fulfilled the separator condition, and since b is a maximum weight branch of ${}^{k}\mathcal{T}$, it is obvious that the new connectors are heavier than any branch in ${}^{k+1}\mathcal{T}'$. Hence, the connector condition holds for ${}^{k+1}\mathcal{T}'$.

The separators of ${}^{k}\mathcal{T}$ which are also separators of ${}^{k+1}\mathcal{T}'$ maintain their fundamental paths. Hence, the separator condition holds for ${}^{k+1}\mathcal{T}'$. □

Corollary 3.37. *Every SSSkT of a graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ has a subgraph which is a SSSk + 1T of $\mathcal{G}$ with the same set of seeds, provided that $k < \#\mathbf{V}$.*

Proof. Since k is smaller than the number of vertices of $\mathcal{G}$, there must be at least one branch in ${}^{k}\mathcal{T}$. Hence, it suffices to choose the branch with maximum weight and remove it from the original SSSkT. The result, by Theorem 3.36, is a SSS$k+1$T of $\mathcal{G}$ with the same set of seeds. □

Theorem 3.38. *All SSSkT of a graph are subgraphs of some SSSSk′T of the same graph with the same set of seeds.*

Proof. Let ${}^{k}\mathcal{T}$ be a SSSkT of a graph $\mathcal{G}$ with c connected components and c' seedless connected components. Let ${}^{k}\mathcal{T}$ respect a set of n seeds. According to Theorem 3.6, any SSSSk'T of $\mathcal{G}$ has $k' = n+c'$. Clearly, $k \geq k'$. If Theorem 3.34 is applied successively to ${}^{k}\mathcal{T}$, thereby constructing a sequence ${}^{k}\mathcal{T}, {}^{k-1}\mathcal{T}, \ldots, {}^{n+c'}\mathcal{T}$ by insertion at each step of a lightest connector, the first element of the sequence, ${}^{k}\mathcal{T}$, is a subgraph of the last element of the sequence, ${}^{n+c'}\mathcal{T} = {}^{k'}\mathcal{T}$, which is a SSSS$k'$T. □

Theorem 3.39. *All SSSSkT of a graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ have subgraphs which are SSSk′T of $\mathcal{G}$ with the same set of n seeds, provided that $k \leq k' \leq \#\mathbf{V}$.*

Proof. From a SSSSkT ${}^{k}\mathcal{T}$, it is obvious that, by successive removal of a heaviest branch, one can build a sequence ${}^{k}\mathcal{T}, {}^{k+1}\mathcal{T}, \ldots, {}^{k'}\mathcal{T}$ whose elements are all subgraphs of ${}^{k}\mathcal{T}$, and all shortest, according to Theorem 3.36. Hence, element ${}^{k'}\mathcal{T}$ is a SSSk'T as required. □

Lemma 3.40. *Let $\mathcal{F}$ be a spanning forest of a graph $\mathcal{G}$. Let v_1 and v_2 be two different but connected vertices of $\mathcal{G}$. Let $\mathbf{P}$ be the v_1, v_2-path in $\mathcal{F}$. If b is a branch of $\mathcal{F}$ which is also in $\mathbf{P}$ and c a chord in its fundamental cutset, then the ring sum of the arcs in $\mathbf{P}$ with the arcs in the fundamental circuit $\mathbf{C}$ associated with c is the unique v_1, v_2-path in the spanning forest $\mathcal{F} - b + c$.*

Proof. First notice that b belongs to both $\mathbf{P}$ and $\mathbf{C}$, and that c belongs to $\mathbf{C}$ but not to $\mathbf{P}$, since $\mathbf{P}$ contains only branches of $\mathcal{F}$. Also notice that $\mathcal{F} - b + c$ is still a spanning forest of $\mathcal{G}$.

Let v_a be the first vertex in $\mathbf{C}$ found when scanning all vertices of $\mathbf{P}$ starting from v_1. Similarly, let v_b be the first vertex in $\mathbf{C}$ found when scanning all vertices of $\mathbf{P}$ starting from v_2.

Suppose $v_a = v_b$. Since paths have no repeated vertices, this implies that $\mathbf{P}$ and $\mathbf{C}$ touch only at a single vertex, and hence do not have common arcs, which cannot happen by hypothesis, since b is a common arc. Hence, $v_a \neq v_b$.

The two vertices v_a and v_b divide circuit $\mathbf{C}$ into two segments $\mathbf{C}_1$ and $\mathbf{C}_2$, each a disjoint v_a, v_b-path. The chord c either belongs to $\mathbf{C}_1$ or $\mathbf{C}_2$. Suppose, without loss of generality, that it belongs to $\mathbf{C}_2$. Then, $\mathbf{C}_1$ is composed solely of branches of $\mathcal{F}$: it is the v_a, v_b-path in $\mathcal{F}$.

Similarly, the two vertices v_a and v_b divide path $\mathbf{P}$ into three segments, all arc-disjoint paths in $\mathcal{F}$: the v_1, v_a-path $\mathbf{P}_{1a}$, the v_a, v_b-path $\mathbf{P}_{ab}$, and the v_b, v_2-path $\mathbf{P}_{b2}$. It is obvious, then, that $\mathbf{P}_{ab} = \mathbf{C}_1$, since paths are unique in forests, by Theorem 3.2. It is also clear that $\mathbf{P}_{1a} \cap \mathbf{C}_2 = \mathbf{P}_{2b} \cap \mathbf{C}_2 = \emptyset$, by selection of v_1 and v_2. Thus, $\mathbf{P} \cap \mathbf{C} = \mathbf{C}_1$ and $b \in \mathbf{C}_1$.

The ring sum of $\mathbf{P}$ and $\mathbf{C}$ is thus,

$$\begin{aligned}\mathbf{P} \oplus \mathbf{C} = & \\ &= (\mathbf{P} \cup \mathbf{C}) \setminus (\mathbf{P} \cap \mathbf{C}) \\ &= (\mathbf{P}_{1a} \cup \mathbf{P}_{ab} \cup \mathbf{P}_{b2} \cup \mathbf{C}_2) \setminus (\mathbf{P}_{ab}) \\ &= \mathbf{P}_{1a} \cup \mathbf{P}_{b2} \cup \mathbf{C}_2,\end{aligned}$$

which is obviously a v_1, v_2-path. This path is composed solely of branches of $\mathcal{F}$ except for one chord, $c \in \mathbf{C}_2$, and it does not contain branch $b \in \mathbf{C}_1$. Hence, this path belongs to the spanning forest $\mathcal{F} - b + c$. □

Theorem 3.41. *Let $\mathcal{F}(\mathbf{V}, \mathbf{A}_s) = {}^c\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ be the SSF of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ with c connected components, and let $\mathbf{S}$ be a set of n seed vertices. Suppose there are c' seedless and c'' seeded connected components of $\mathcal{G}$, i.e., $c = c' + c''$. A SSSSkT, with $k = n + c'$, may be obtained by successively removing from $\mathcal{F}$ a set of $k - c = n + c' - c = n - c''$ branches chosen so that each is, at its step, the heaviest branch on all possible paths in the k-tree between pairs of different seeds.*

Proof. It will be proved that the chosen branches lead to a set of separators which fulfill the separator condition.

If $c'' = n$, then the seeds are already separated, and the SSF $\mathcal{F}$ is indeed a SSSSkT of $\mathcal{G}$, since there is no spanning k'-tree with a smaller k' nor a spanning k tree with a smaller weight.

Assume that the set of seeds $\mathbf{S}$ is initially partitioned into subsets $\mathbf{S} = \mathbf{S}_1 \cup \cdots \cup \mathbf{S}_c$, each containing the seeds in connected component $\mathcal{G}_i$ of $\mathcal{G}$. Clearly $\mathbf{S}_i = \emptyset$ if $\mathcal{G}_i$ is a seedless component of $\mathcal{G}$, and hence there are c' empty sets $\mathbf{S}_i$. Let $\mathcal{F}_i$ be the connected component of $\mathcal{F}$ covering $\mathcal{G}_i$. When the heaviest branch in any path between two seeds is removed from $\mathcal{F}$, it is removed from one of its components, say $\mathcal{F}_i$. Hence, $\mathcal{F}_i$ is separated into two connected components, each containing a non-empty set of seeds. The subset $\mathbf{S}_i$ of $\mathbf{S}$ can be split into the corresponding seeds. This process ends when all sets in the partition of $\mathbf{S}$, each corresponding to a tree, contain either zero or one seed. The number of seedless $\mathbf{S}_i$ does not change while branches are removed. Hence, the final number of trees is $c' + n$, as required. Since initially there were c connected components and a connected component is added for each removed branch, the total number of removed branches is $c' + n - c = n - c''$. For each component $\mathcal{F}_i$ of $\mathcal{F}$ with n_i seeds, exactly $n_i - 1$ branches are removed so as to separate its seeds. Hence, attention can be concentrated on each component at a time. The removal of the branches as stated indeed leads to a n-seed respecting spanning $c' + n$-tree of $\mathcal{G}$. It remains to be seen whether it is shortest.

Let $\mathcal{F}_i$ be a component of $\mathcal{F}$ containing $n_i > 1$ seeds. Clearly, $n_i - 1$ branches will be removed from $\mathcal{F}_i$. Since $\mathcal{F}_i$ is a SST of the corresponding connected component $\mathcal{G}_i$ of $\mathcal{G}$, see Corollary 3.14, the chords of $\mathcal{F}_i$ in $\mathcal{G}_i$ fulfill the chord condition. When the heaviest branch b in any path between seeds in $\mathcal{F}_i$ is removed, it becomes a connector of the spanning 2-tree obtained, the same thing happening with all the chords in the corresponding fundamental cutset. All these connectors connect seeded trees, and hence, even though each of the resulting trees may have more than one seed, they are separators. Actually, they will be separators in the final n_i-tree covering $\mathcal{F}_i$. Since b is the heaviest branch in any path between two seeds in $\mathcal{F}_i$, it fulfills the separator condition for whatever resulting final partition of seeds among the trees.

Consider now a chord c in the fundamental cutset of b, $\mathbf{C}$ its fundamental circuit, and consider also any path $\mathbf{P}$ between two seeds of $\mathcal{F}_i$ which contains b. By Lemma 3.40, $\mathbf{C} \oplus \mathbf{P}$ is the path between the same two seeds in $\mathcal{F} - b + c$. Since $w(c) \geq w(b')$ for all $b' \in \mathbf{C}$ (chord condition is fulfilled for $\mathcal{F}_i$), $w(b) \geq w(b'')$ for all $b'' \in \mathbf{P}$ (by selection of b), and b is common to $\mathbf{C}$ and $\mathbf{P}$, then $w(c) \geq w(b'')$ for all $b'' \in \mathbf{P}$. Since this happens for all paths, c fulfills the separator condition for whatever resulting final partition of seeds among the trees.

Hence, by repeating the above arguments for any component trees with more than two seeds, it is clear that the removal of the selected branches leads to a n-seed respecting spanning $c' + n$-tree which fulfills the chord, connector and separator conditions (the connector condition is fulfilled trivially since there are no connectors in the final seeded $c' + n$-tree), and hence, by Corollary 3.33, it is a SSSS$c' + n$T. □

Theorem 3.42. *Let ${}^k\mathcal{T}(\mathbf{V}, \mathbf{A}_s)$ be the SSSSkT of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ with c connected components, respecting the set $\mathbf{S}$ of n seed vertices. Suppose there are c' seedless and c'' seeded connected components of $\mathcal{G}$, i.e., $c = c' + c''$. A SSF of $\mathcal{G}$ may be obtained by successively adding to ${}^k\mathcal{T}$ a set of $k - c = n + c' - c = n - c''$ separators chosen so that each is, at its step, the lightest connector of any two trees in the current forest.*

Proof. It is clear that after $k - c$ insertions of new branches into the forest which initially is ${}^k\mathcal{T}$, with k components, the resulting forest has $k - k + c = c$ components as required. That this is possible is evident, since $\mathcal{G}$ has c components and thus is spannable by a forest with c components.

It remains to be seen whether the final spanning forest $\mathcal{F}$ fulfills the chord condition.

The chords of ${}^k\mathcal{T}$ are clearly also chords of $\mathcal{F}$ with the same fundamental circuit. Since, by Theorem 3.33, they fulfill the chord condition in ${}^k\mathcal{T}$, they also fulfill it in $\mathcal{F}$.

Each time a separator c is introduced to the forest, it becomes a branch whose fundamental cutset corresponds to the other separators c' connecting the two trees connected by c. By selection, $w(c') \geq w(c)$. Let $\mathbf{P}$ and $\mathbf{P}'$ be the fundamental paths associated to c and c' in ${}^k\mathcal{T}$. Then, $w(c) \geq w(b)$ for all $b \in \mathbf{P}$ and $w(c') \geq w(b)$ for all $b \in \mathbf{P}'$. By Lemma 3.27, $\mathbf{P} \oplus \mathbf{P}'$, which cannot be empty since $c' \notin \mathbf{P}$ and $c \notin \mathbf{P}'$, is a circuit or a sum of arc-disjoint circuits. Since $\mathbf{P} \oplus \mathbf{P}'$ contains a single chord, it can only contain one circuit, since otherwise there would be a chordless circuit in the forest. This circuit is the fundamental circuit of c' and, from the relations above, $w(c') \geq w(b)$ for all $b \in \mathbf{P} \oplus \mathbf{P}'$. Hence, when a new branch is introduced as specified, the corresponding new chords fulfill the chord condition. Hence, by Theorem 3.11, the final spanning forest is indeed a SSF of $\mathcal{G}$. □

Algorithms

There are essentially two types of algorithms for finding a SSS*k*T of a graph, as was hinted before: constructive algorithms and destructive algorithms. The problem of finding a SSSS*k*T of a graph is a particular case where $k = n + c'$, c' being the number of seedless components of the graph and n the number of seeds.

Constructive algorithms

Two basic constructive algorithms can be developed for finding the SSSS*k*T of a graph. Both are based on Kruskal and Prim, and can be proven correct using the following theorem:

Theorem 3.43. *Let $\mathcal{G}'(\mathbf{V}', \mathbf{A}')$ be the extension of a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ with an associated set of $n > 0$ seed vertices $\mathbf{S} = \{s_1, \ldots, s_n\}$, such that $\mathbf{V}' = \mathbf{V} \cup \{v_e\}$ and $\mathbf{A}' = \mathbf{A} \cup \mathbf{A}_e$, with $\mathbf{A}_e = \bigcup_{i=1}^{n}\{a_{e_i}\}$, where v_e is an extra, external vertex, and a_{e_i} are extra arcs, connecting the extra vertex to each seed vertex, i.e., $g(a_{e_i}) = \{v_e, s_i\}$. Let the weight of the extra arcs be strictly smaller than any other arc in the graph, i.e., $w(a_{e_i}) < w(a)$ for all $a \in \mathbf{A}$ with $i = 1, \ldots, n$. If the forest $\mathcal{F}'(\mathbf{V}', \mathbf{A}'_s)$ is a SSF of $\mathcal{G}'$, then $\mathcal{F}' - v_e$ is a SSSSkT of $\mathcal{G}$ for the given set of seeds.*

Proof. The arcs of $\mathcal{G}'$ in $\mathbf{A}_e$, i.e., the extra arcs, must be branches of $\mathcal{F}'$. Suppose that there is an arc a_{e_i} which is not a branch of $\mathcal{F}'$, i.e., it is a chord of $\mathcal{F}'$. Let $\mathbf{C}$ be its fundamental circuit. $\mathbf{C}$ cannot consist only of arcs in $\mathbf{A}_e$, since by construction all arcs in $\mathbf{A}_e$ connect to a different seed. Let then b be a branch of $\mathcal{F}'$ in $\mathbf{C}$ which is an arc of $\mathcal{G}$. By hypothesis $w(b) > w(a_{e_i})$, and, by the chord condition for SSF in Theorem 3.11, $w(b) \leq w(a_{e_i})$. This is a contradiction. Hence, a_{e_i} is a branch of $\mathcal{F}'$.

The removal of v_e from $\mathcal{F}'$ also removes all branches of $\mathcal{F}'$ that are incident on v_e, i.e., the n arcs in $\mathbf{A}_e$. Hence, $\mathcal{F}' - v_e$ contains only vertices and branches of $\mathcal{G}$. Since $\mathbf{V}' = \mathbf{V} \cup v_e$, $\mathcal{F}' - v_e$ contains all vertices of $\mathcal{G}$, hence, $\mathcal{F}' - v_e$ spans $\mathcal{G}$. Since $\mathcal{F}'$ is a forest, and hence acyclic, $\mathcal{F}' - v_e$ must also be acyclic. Hence, $\mathcal{F}' - v_e$ is a spanning forest of $\mathcal{G}$.

It is clear that, if $\mathcal{G}$ has $c = c' + c''$ connected components, of which c' are seedless and c'' are seeded, $\mathcal{G}'$ has $c' + 1$, since all seeded connected components are connected through the extra arcs in $\mathbf{A}_e$ to one another. By definition of spanning forest, $\mathcal{F}'$ has also $c' + 1$ connected components.

The removal of each arc in $\mathbf{A}_e$ from $\mathcal{F}'$ increases the number of connected components by one, hence $\mathcal{F}' - a_{e_1} - \cdots - a_{e_n}$ has $c + 1 + n$ connected components. Finally, removal of v_e reduces the number of connected components by one, that is, $\mathcal{F} - v_e$ has $c' + n$ connected components, and spans the graph $\mathcal{G}$. Hence, it is a spanning $n + c'$-tree of $\mathcal{G}$.

Suppose there is a s_i, s_j-path $\mathbf{P}$ between to seeds s_i and s_j of $\mathbf{S}$ in $\mathcal{F}'$ containing only arcs of $\mathcal{G}$. Then, $\mathbf{P}, a_{e_j}, v_e, a_{e_i}, s_i$ is clearly a circuit in $\mathcal{F}'$, which is a contradiction, since $\mathcal{F}'$ is a forest and hence acyclic. This proves that all paths between seeds must pass though the vertex v_e and two extra arcs. That is, $\mathcal{F} - v_e$ respects seed separation. Hence, $\mathcal{F} - v_e$ is a n-seed respecting spanning $n + c'$-tree of graph $\mathcal{G}$ with seeds $\mathbf{S}$. It is also smallest, by Theorem 3.6, since it has the right number of connected components, and thus, by Theorem 3.8, it has no connectors.

It remains to be seen whether it is a SSSSkT. By Corollary 3.33, if $\mathcal{F}' - v_e$ fulfills the chord and separator conditions, then indeed it is a SSSSkT.

Let c be a chord of $\mathcal{F}'$. If its fundamental circuit includes only arcs of $\mathcal{G}$, this circuit will remain unaltered in $\mathcal{F} - v_e$. Hence, c is also a chord of $\mathcal{F}' - v_e$. Since, by Theorem 3.11 the chord condition holds for all chords of $\mathcal{F}'$, it also holds for c.

Let c be a chord of $\mathcal{F}'$. If its fundamental circuit includes any of the extra arcs, then it cannot be a chord of $\mathcal{F}' - v_e$. Hence, it must be a separator. Let $\mathbf{C}$ be the fundamental circuit of c in $\mathcal{F}'$. It is straightforward to see that circuit $\mathbf{C}$ includes exactly two arcs of $\mathbf{A}_e$, which besides are in succession. Hence, circuit $\mathbf{C}$ corresponds in $\mathcal{F}' - v_e$ to the fundamental path of c, connecting two different seeds. Since the chord condition holds for $\mathcal{F}'$, again by Theorem 3.11, $w(c) \geq w(a)$ for all $a \in \mathbf{C}$ and hence also in the fundamental path of c in $\mathcal{F}' - v_e$.

Since both chord and separator conditions hold for $\mathcal{F}' - v_e$, it is indeed a SSSSkT. □

Suppose this theorem is used to develop a simple algorithm: build the SSF of the extended graph, using Kruskal's or Prim's algorithms, and then remove the extra vertex v_e. Assume the initial vertex in Prim's algorithm is the extra vertex v_e. In both cases, it is straightforward to see that the first n arcs introduced into the forest are the extra arcs. Since all seeds, after the first n steps of each algorithm, are connected through v_e, branches which would connect two connected components with different seeds in $\mathcal{F} - v_e$ are automatically forbidden, since they are chords of $\mathcal{F}'$, i.e., they would introduce circuits in $\mathcal{F}'$.

In the case of the Kruskal algorithm, this amounts to extending the basic algorithm so that only connectors, and not separators, are considered as candidate branches. At each step, the forest obtained is a SSSkT of the graph, by Theorem 3.34. This, of course, requires that separators are somehow distinguished from connectors. This can be done by labeling the vertices according to the seed of the component they belong to. It simply requires running a DFS (linear in the number of vertices) to label the unlabeled component of two components connected through a new branch. If the two components connected are unlabeled, nothing needs to be done. The overall time spent in labeling is asymptotically linear on the number of branches, and hence on the number of vertices, i.e., $O(\#\mathbf{V})$. In total, since only unlabeled vertices are labeled, $\#\mathbf{V}$ vertices are labeled. The running time of the algorithm thus does not change asymptotically.

In the case of the Prim algorithm, this amounts to extending the basic algorithm so that initially there are n seeds instead of a single one. Since the forest grows from seeds one vertex at a time, it is a simple matter to propagate seed labels along the vertices of each connected component (a different label for each seed) so as to eliminate from consideration arcs connecting vertices with different labels. The reason for the use of the name "seed" for the vertices that must be kept separate should now be clear. In this extension of the Prim algorithm, the set of selected branches at each step constitutes a forest with n trees. Hence, $c' + 1$ iterations of the algorithm are required, c' using the simple version of the algorithm for each seedless component of the graph and another using the n seeds which cover the remaining seeded components. The same result may be obtained in a single iteration by inserting one fictitious seed in each seedless component of the graph. Unlike what happens with the Kruskal extension for seeded k-trees, this extension of the Prim algorithm, though leading to a SSSSkT of the graph, does not have the nice property of all intermediate forests being SSSkTs.

Both algorithms are important because of their uses in segmentation. Even though the issue of segmentation will be dealt with within Chapter 4, it may be advanced here that Prim's algorithm is used mostly in region growing segmentation algorithms and Kruskal' algorithm is used mostly in region merging with seeds segmentation algorithms, and that, although both algorithms lead to SSSSkTs, their extensions by globalizing information have vastly different properties.

Destructive algorithms

Destructive algorithms start by building a SSF of the graph. Then, following Theorem 3.41, a SSSSkT can be obtained by cutting appropriately chosen branches of the forest. Curiously enough, the same method can be used, according to Theorem 3.24, to obtain a SSkT of the graph. The only difference between destructive algorithms for finding SSSSkTs or SSkTs is which branch to cut at each step of the algorithm: for unseeded trees the heaviest among all forest branches is chosen, while for seeded trees the heaviest of those branches whose removal separates seeds is chosen. Since destructive algorithms successively remove branches from a forest, each time separating a connected into two new connected components, these algorithms, in the framework of image segmentation, are some times called region splitting algorithms, see Chapter 4.

If the Kruskal algorithm is used to build the SSF, since it inserts forest branches of non-decreasing weight, a list of the branches of the forest sorted by non-increasing order may be built without changing the asymptotic behavior of the algorithm. Hence, it will run in $O(\#\mathbf{A} \lg \#\mathbf{A})$. Selection of the heaviest branches then takes linear time, that is $O(k - c)$, since $k - c$ branches must be removed from the sorted list, each removal taking a constant time.

If the Prim algorithm is used, a priority queue can be used to insert the branches of the forest. If Fibonacci priority queues are used, then each insertion requires $O(1)$ amortized time, so that the queue takes linear time, on $\#A_s = \#\mathbf{V} - c$, to build (see [28] for details). Hence, running Prim and building the queue still takes $O(\#\mathbf{A} + \#\mathbf{V} \lg \#\mathbf{V})$. Selection of the heaviest branches then takes $O\big((k - c) \lg(\#\mathbf{V} - c)\big)$, since $k - c$ branches must be removed, one at a time, from the queue, each removal taking $O\big(\lg(\#\mathbf{V} - c)\big)$, $\#\mathbf{V} - c$ being the size of the queue.

This was in the case of the SSkT. The selection of the branches to remove in the case of the SSSSkT is more involved, since only those branches which stand in the path between two seeds should be taken into account. The following theorem will be used to build an algorithm which solves the problem in linear time, provided a list of branches sorted by non-decreasing weight is available, as in the case of the Kruskal algorithm for the destructive construction of a SSkT from a SSF above. Even though there may be more efficient algorithms, this proves that linear time algorithms do exist, which means that, if several different sets of seeds are to be experimented in building SSSSkTs, the SSF can be built once, followed by successive applications of an asymptotically linear time algorithm. If the number of experiments with sets of seeds is high enough, the overall efficiency approximates linear time asymptotically. That is, the algorithm runs in asymptotically linear amortized time.

Theorem 3.44. *Let $\mathcal{F}$ be the SSF of a graph $\mathcal{G}$. If ${}^k\mathcal{T}$ is a SSSSkT of $\mathcal{F}$, then ${}^k\mathcal{T}$ is also a SSSSkT of $\mathcal{G}$.*

Proof. Clearly, ${}^k\mathcal{T}$ has no chords in $\mathcal{F}$, since $\mathcal{F}$ is acyclic. But ${}^k\mathcal{T}$ may have separators (not connectors, since it is smallest). Suppose $\mathcal{F}$ has $\#\mathbf{V}$ vertices and c connected components. Suppose c' of them are seedless. It is known that $k = n + c'$, where n is the number of seeds. Also, the number of arcs in ${}^k\mathcal{T}$ is $\#\mathbf{V} - n - c'$ and the number of arcs in $\mathcal{F}$ is $\#\mathbf{V} - c$. Hence, there are $n + c' - c = n - c''$ separators of ${}^k\mathcal{T}$ in $\mathcal{F}$, where c'' is the number of seeded components of $\mathcal{F}$.

It is also clear that ${}^k\mathcal{T}$ spans $\mathcal{G}$, is acyclic, does not violate seed separation, and has the right number of connected components to be smallest.

The separators of ${}^k\mathcal{T}$ in $\mathcal{F}$ are also separators of ${}^k\mathcal{T}$ in $\mathcal{G}$. Hence, they fulfill the separator condition. The arcs of $\mathcal{G}$ which are not branches of $\mathcal{F}$, i.e., chords of $\mathcal{F}$, are either chords or separators of ${}^k\mathcal{T}$.

If their fundamental circuit $\mathbf{C}$ in $\mathcal{F}$ includes a branch of $\mathcal{F}$ which is a separator of ${}^k\mathcal{T}$, then they are themselves separators, since they introduce no circuit in ${}^k\mathcal{T}$.

Take one such separator c of ${}^k\mathcal{T}$. Since it is also a chord of $\mathcal{F}$, it must fulfill the chord condition for SSFs, that is $w(c) \geq w(b)$ for all branches b of $\mathcal{F}$ in $\mathbf{C}$, including the separators of ${}^k\mathcal{T}$. The fundamental path of c in ${}^k\mathcal{T}$ is, by Lemma 3.40 (remember that c is in the fundamental cutsets of all branches of its circuit), the ring sum of the path $\mathbf{P}$ between the two seeds in $\mathcal{F}$ and the circuit $\mathbf{C}$. Since the fundamental path includes a single separator, c, all branches of $\mathcal{F}$ which are separators of ${}^k\mathcal{T}$ in $\mathbf{C}$ must also be in $\mathbf{P}$. Hence, using Lemma 3.31, it must be concluded that all arcs in $\mathbf{P}$ are lighter than the heaviest separator in $\mathbf{P}$, and hence c is heavier than any branch of ${}^k\mathcal{T}$ in its fundamental path. Hence, ${}^k\mathcal{T}$ fulfills the separator condition.

Let c now be a chord of $\mathcal{F}$ such that its fundamental circuit does not include any separator of ${}^k\mathcal{T}$. Then c is also a chord of ${}^k\mathcal{T}$ and obviously fulfills the chord condition.

Hence, ${}^k\mathcal{T}$ is indeed a SSSSkT of $\mathcal{G}$. □

The Kruskal algorithm, with the seed extensions described previously, can thus be used to calculate the SSSSkT of the SSF, and consequently of the graph. However, notice that:

1. there is no need to initially sort the branches of $\mathcal{F}$, since, by hypothesis, a sorted list is already available.
2. there is no need to test whether an arc introduces a circuit or not, since $\mathcal{F}$ has no circuits.

The analysis of the Kruskal algorithm with the above simplifications reveals that it runs indeed in linear time, i.e., $O(\#\mathbf{V})$.

Further notes on trees and algorithms

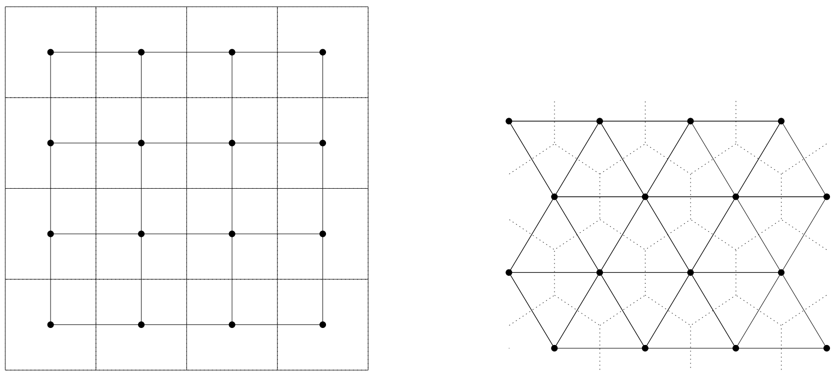
There are some issues regarding shortest spanning (k-) trees and forests which should be mentioned here. The first has to do with the uniqueness of the solutions. In general, none of the problems discussed so far (SSF, SSkT, SSSkT, SSSSkT) has a unique solution. It can be proved that, if all arc weights are different, then solutions are always unique, but this is a rather conservative condition. Less conservative conditions of uniqueness can be developed in each case, but this will not be attempted here. The consequences of non-uniqueness and ways of dealing with the issue will be discussed in Chapter 4, in the framework of a particular application: segmentation for image analysis.

Secondly, it must be pointed out that in all constructive algorithms the addition of a branch to the growing forest can be followed by a contraction of that branch in the original graph, followed possibly by removal of self-connecting arcs (which in the uncontracted graph are chords). This results in contraction of the vertices connected by a tree in the evolving forest to a single representative vertex. The remaining arcs of the graph retain their properties, and hence the algorithms run exactly in the same manner with and without contraction. The proof of this fact is simple and will not be given here. Such contractions may be very useful, since they allow each component of the evolving forest, a segment or class or region in the framework of segmentation, to be represented by a single vertex, hence simplifying possible region globalization processes. In the case of 2D maps, as will be seen in Section 3.5, such simplifications of the map graph can be accompanied by dual changes in the border pseudograph, which may be useful for globalization of border information and even for contour coding.

3.3.10 Graphs and lattices

The coordinates v_k of any site of a lattice $\mathcal{L}$ in $\mathbb{R}^m$ belong to $\mathbb{Z}^m$. A grid can thus be associated with a 2D lattice: each vertex in the grid corresponds to a site in the lattice and vice versa. Lattice sites can be associated with grid vertices through a site function: all sites s of a lattice can be written $s = s[v] = \sum_{j=0}^{m-1} v_j u_j$ where $v = [v_0 \dots v_{m-1}]^T \in \mathbf{V} = \mathbb{Z}^m$. Figure 3.3 shows the 2D rectangular and hexagonal $\mathbb{R}^2$ lattices with the corresponding $\mathbb{Z}^2$ grids superimposed.

Hence, a grid and a lattice are usually associated with a digital image: the latter specifies the positions of each pixel in the original continuous image, while the former introduces a structure, or neighborhood system, into the set of image pixels. The type of grid selected usually depends on the spatial arrangement of the sampling lattice sites: the hexagonal grid in $\mathbb{Z}^2$ is normally associated with a hexagonal lattice in $\mathbb{R}^2$, i.e., when digitalization is performed through a


(a) Rectangular grid over rectangular lattice.

(b) Hexagonal grid over hexagonal lattice.

Figure 3.3: Examples of 2D lattices with superimposed grids. Grid vertices and lattice sites are represented by dots and grid edges are represented by lines. The Voronoi tessellation corresponding to the lattice sites is shown using dotted lines.

hexagonal sampling lattice, it is natural to use a hexagonal grid for representing the relations between the pixels.

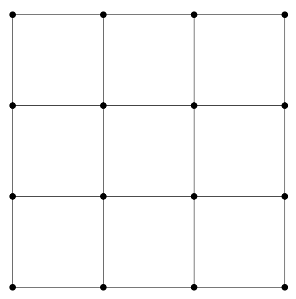
When digitizing, $\mathbf{Z}$ must be chosen such that $s[v] \in \mathbf{R}$. Since $\mathbf{R}$ is usually a bounded region (mostly a rectangle), $\mathbf{Z}$ will also be bounded. The definition of grid given previously precludes the use of a bounded set of vertices, since grids must be t-invariant. However, it is possible to restrict the simple graph corresponding to an unbounded grid to the vertices $\mathbf{Z}$ of interest. Hence, the graphs associated with digital images are usually not only simple, but also limited. From here on, the stress will be put on graphs, rather than grids, and thus $\mathcal{G}(\mathbf{Z}, \mathbf{A})$ will always refer to a graph, usually a simple graph associated to either an image or a sequence of images defined either on $\mathbf{Z} \subset \mathbb{Z}^2$ or on $\mathbf{N} \times \mathbf{Z} \subset \mathbb{Z}^3$.

Definition 3.65. (image graph) *A graph where the vertices correspond to the image pixels and there are arcs between pixels which are geometrical neighbors in the implicit sampling lattice. Often a single extra external vertex is added to represent the outside of the image which is adjacent to all pixels in the boundary of the image. It may be necessary to allow the image graph to be a multigraph. This happens when it is important to establish a correspondence between its arcs and the edges (to be defined later) between pixels. In this case, the corner pixels in a rectangular lattice on a rectangular domain may have two arcs connecting them to the extra external pixel.*

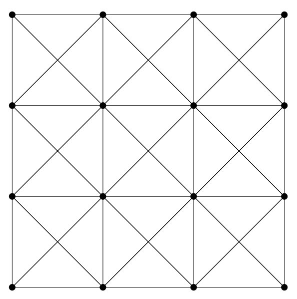
In image graphs there is an implicit neighborhood system. The rectangular 2D graph, obtained from the rectangular grid, defines a N_4 neighborhood system,[11] where each vertex has four neighbors, as shown in Figure 3.4(a). Similarly a N_6 neighborhood system is associated with the

[11] A neighborhood system is N_n if each vertex in the graph has n neighbors, except possibly at the limits of the graph.

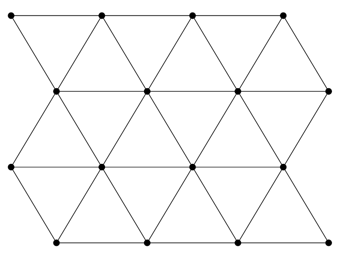
hexagonal grid, see Figure 3.4(c). Another type of neighborhood system, N_8 in Figure 3.4(b), can be associated with images digitized using a rectangular 2D lattice. This type of neighborhood system can be very useful, even though it cannot correspond to any possible grid, since it fails the non-crossing condition for grids (this neighborhood system is associated with a non-planar graph).



(a) N_4 on a rectangular lattice.



(b) N_8 on a rectangular lattice.



(c) N_6 on a hexagonal lattice.

Figure 3.4: Examples of 2D image graphs with different neighborhood systems. Graph vertices are represented by dots and graph arcs are represented by lines.

In the case of a progressive (3D) sampling matrix, the natural neighborhood system, and hence graph, to associate with the digital pixel sequence is N_6, where each vertex has six neighbors, as shown in Figure 3.5.

Pixel neighborhoods and connectivity

On a rectangularly sampled digital image $f[\cdot]$ defined on $\mathbf{Z} \subset \mathbb{Z}^2$, the terms $N_4(v)$ (or 4-neighbors of v), $N_d(v)$ (or d-neighbors of v) and $N_8(v)$ (or 8-neighbors of v), v being a 2D pixel $v = [i, j]$, will be taken to mean

$$N_4(v) = \{[i-1, j], [i, j-1], [i, j+1], [i+1, j]\} \cap \mathbf{Z}, \tag{3.2}$$

$$N_d(v) = \{[i-1, j-1], [i-1, j+1], [i+1, j-1], [i+1, j+1]\} \cap \mathbf{Z}, \text{ and} \tag{3.3}$$

$$N_8(v) = N_4(v) \cup N_d(v). \tag{3.4}$$

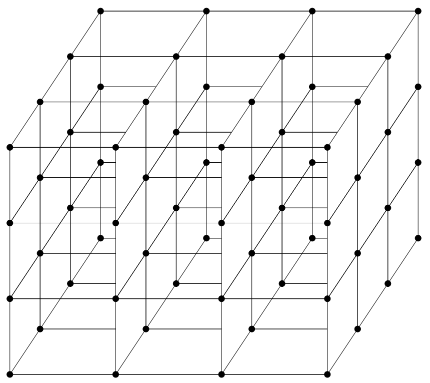


Figure 3.5: The 3D progressive sampling lattice with the N_6 neighborhood system.

Two pixels v_1 and v_2 are 4-connected if $v_2 \in N_4(v_1)$, d-connected if $v_2 \in N_d(v_1)$, and 8-connected if $v_2 \in N_8(v_1)$.

Mixed connectivity is defined with respect to a given property $P(.)$ of the values of the pixels of an image f. Two pixels v_1 and v_2 such that $P(f[v_1]) = P(f[v_2])$ are m-connected if $v_2 \in N_4(v_1)$ or if $v_2 \in N_d(v_1)$ and $\forall v \in N_4(v_1) \cap N_4(v_2), P(f[v]) \neq P(f[v_1])$. Mixed connectivity is a modification of 8-connectivity which eliminates multiple path connections in sets of pixels when the N_8 image graph is used [56].

3.4 Planar graphs and duality

Planar graphs have a series of interesting properties. If a graph is planar, so is any of its contractions, any of its reductions, and, in general, any homeomorphic graph. Notice, however, that if the contraction of a graph is planar, nothing can be said about the planarity of the original.

3.4.1 Euler theorem

When a planar graph is drawn without crossing arcs, its arcs divide the 2D plane into regions (or faces), all but one of which are bounded. Drawing a graph on a plane can be seen to be equivalent to drawing it on a sphere (see [186]), and, on a sphere, all regions are bounded. Let r be the number of regions in a planar embedding of a connected planar graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$. Then, by Euler's theorem [44], $\#\mathbf{V} + r - \#\mathbf{A} = 2$. This formula can be extended to encompass also disconnected graphs. Let c be the number of connected components in the planar graph. Then, $\#\mathbf{V} + r - \#\mathbf{A} = 1 + c$. Or, which is the same, $\mu(\mathcal{G}) = r - 1$ or even $\rho(G) = 1 + \#\mathbf{A} - r$.

An arc subdivision performed on a planar graph removes one arc, adds other two arcs, and an extra non-isolated vertex: the number of connected components and the number of regions do not change. The same thing happens for an arc reduction, where the net result is one less vertex and one less arc.

A nice corollary of Euler's theorem is that in any planar, *simple* graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ with $\#\mathbf{V} > 2$, the relation $\#\mathbf{A} \leq 3\#\mathbf{V} - 6$ always holds [44]. The importance of this relation, in the framework of this thesis, stems from the fact that graph algorithms whose efficiency is bounded above by some polynomial function $f(\cdot)$ of $\mathbf{A}$, that is, algorithms which run in $O(f(\#\mathbf{A}))$, are also $O(f(\#\mathbf{V}))$ in the case of planar, simple graphs. The result is also valid if some of the terms in $f(\cdot)$ are logarithms. Notice that, for simple graphs in general, one can only say that $\#\mathbf{A} \leq \#\mathbf{V}(\#\mathbf{V}-1)/2$, which is an equality for a complete simple graph.

No simple relations exist between the number of arcs and the number of vertices in the general case of pseudo- or multigraphs, even if planar.

3.4.2 Duality

Definition 3.66. (duality [186]**)** *A graph $\mathcal{G}_2(\mathbf{V}_2,\mathbf{A}_2)$ is a dual of a graph $\mathcal{G}_1(\mathbf{V}_1,\mathbf{A}_1)$ if there is a bijective mapping between $\mathbf{A}_2$ and $\mathbf{A}_1$ such that a set of arcs in $\mathbf{A}_2$ is a circuit vector of $\mathcal{G}_2$ iff the corresponding set of arcs in $\mathbf{A}_1$ is a cutset vector of $\mathcal{G}_1$.*

For questions of economy of notation, it will be assumed that dual graphs share the same set of arcs, i.e., the bijective mapping is the identity function. In this case, it is the arc functions of the dual graphs which are different and which map the same set of arcs to pairs of vertices from the two different graphs.

It can be proved that if graph $\mathcal{G}_2$ is a dual of graph $\mathcal{G}_1$, then graph $\mathcal{G}_1$ is a dual of graph $\mathcal{G}_2$. Hence, it may be said that two graphs are dual. If two graphs are dual, then circuits in one correspond to cutsets in the other. Two dual graphs always have the same number of arcs, by definition.

But perhaps the most important fact about duals is that a graph has a dual iff it is planar. Hence, dual graphs are always planar. Notice that duals, in general, are not unique. However, it can be proved that all duals of a graph $\mathcal{G}$ are 2-isomorphic, and that every graph 2-isomorphic to a dual of $\mathcal{G}$ is also a dual of $\mathcal{G}$.

Given an arbitrary disconnected planar graph, by the definition of 2-isomorphism, it is always possible to find a connected 2-isomorphic graph. Hence, any planar graph, disconnected or not, has a connected dual.

Given a 2D embedding of a planar graph, a dual graph can be derived as follows:

Definition 3.67. (geometrical dual of a planar embedding) *Let $\mathcal{G}(\mathbf{V},\mathbf{A})$ be a planar graph with a given graphical representation without crossing arcs (a planar embedding). Let $\mathbf{F}$ be the set of the regions in its graphical representation (including the outer, unbounded region). Then, the planar graph $\mathcal{G}_d(\mathbf{F},\mathbf{A})$ with each arc $a \in \mathbf{A}$ such that $g_d(a) = \{r_1, r_2\}$, where r_1 and r_2 are the regions which arc a separates in the original graph, is the geometrical dual of the given planar embedding of graph $\mathcal{G}(\mathbf{V},\mathbf{A})$.*

It can be proved that the geometrical dual of a planar embedding of a planar graph is indeed a dual of the planar graph. Also, by construction, all geometrical duals are connected.

Let $\mathcal{G}_d(\mathbf{V}_d, \mathbf{A})$ be the geometric dual of a planar embedding of the planar connected graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$. Clearly, by construction of geometric duals, $\#\mathbf{V}_d = r$, where r is the number of regions of $\mathcal{G}$. Then, by Euler's theorem, $\#\mathbf{V} = r_d$, where r_d is the number of regions in $\mathcal{G}_d$. Hence, given two connected dual graphs, the number of vertices in one is equal to the number of regions in the other.

Since 2-isomorphic graphs have the same rank and the same number of arcs, then, by Euler's formula, all duals of a planar graph have the same number of regions.

Theorem 3.45. *If two graphs are dual, the nullity of one is equal to the rank of the other.*

Proof. Let $\mathcal{G}(\mathbf{V}, \mathbf{A})$ and $\mathcal{G}_d(\mathbf{V}_d, \mathbf{A})$ be two planar graphs. Let $\mathcal{G}'(\mathbf{V}', \mathbf{A})$ and $\mathcal{G}'_d(\mathbf{V}'_d, \mathbf{A})$ be 2-isomorphic to $\mathcal{G}$ and $\mathcal{G}_d$, respectively, but connected. Hence, $\mathcal{G}'$ and $\mathcal{G}'_d$ are duals. If two graphs are 2-isomorphic, they have the same rank. Hence, $\rho(\mathcal{G}) = \rho(\mathcal{G}')$ and $\rho(\mathcal{G}_d) = \rho(\mathcal{G}'_d)$. By Euler's formula, $\rho(\mathcal{G}) = \rho(\mathcal{G}') = 1 + \#\mathbf{A} - r'$ and $\rho(\mathcal{G}_d) = \rho(\mathcal{G}'_d) = 1 + \#\mathbf{A} - r'_d$, where $r' = r$ is the number of faces of $\mathcal{G}$ and $\mathcal{G}'$ and any 2-isomorphic graphs, and $r'_d = r_d$ is the number of faces of $\mathcal{G}_d$ and $\mathcal{G}'_d$ and any 2-isomorphic graphs. But $r' = \#\mathbf{V}'_d$. Hence $\rho(\mathcal{G}) = 1 + \#\mathbf{A} - \#\mathbf{V}'_d = \#\mathbf{A} - \rho(\mathcal{G}'_d) = \#\mathbf{A} - \rho(\mathcal{G}_d) = \mu(\mathcal{G}_d)$. □

It will be seen in the following that duality can be used to relate two types of information found in 2D maps: information about adjacency between regions and information about the borders between regions.

Dual operations

Given two dual graphs $\mathcal{G}$ and $\mathcal{G}_d$, it is possible to define an algebra of dual operations on both graphs which still result in dual graphs.

Let a be an arc in the dual graphs $\mathcal{G}$ and $\mathcal{G}_d$. Let $\mathcal{G}'$ be the graph obtained by contracting arc a in $\mathcal{G}$, and $\mathcal{G}'_d$ the graph obtained by removing a from $\mathcal{G}_d$. Then $\mathcal{G}'$ and $\mathcal{G}'_d$ are also duals with the same correspondence between the arcs. Hence, contraction and removal of arcs are dual operations.

Let a be an arc in the dual graphs $\mathcal{G}$ and $\mathcal{G}_d$. If a is self-connecting in $\mathcal{G}$, then it is a circuit vector in $\mathcal{G}$. By duality, it is also cutset vector in $\mathcal{G}_d$, i.e., it is a bridge in $\mathcal{G}_d$. Hence, removal of a self-connecting arc and removal of the corresponding bridge are dual operations.

Consider a vertex v with $d(v) = 2$ and two different arcs a_1 and a_2 of $\mathcal{G}$ incident on v. Performing an arc reduction of v is the same as contracting one of its arcs, say a_1. The two arcs are clearly a cutset vector of $\mathcal{G}$. This cutset vector either consists of a single cutset or it consists of two cutsets. The first case occurs only if a_1 and a_2 are circuit arcs. The second case occurs when both a_1 and a_2 are bridges. In the dual graph the first case corresponds to a_1 and a_2 forming a circuit, that is, to a_1 and a_2 being parallel or multiple arcs. The second one, to a_1 and a_2

being self-connecting arcs. Hence, performing an arc reduction on two circuit arcs is the same as merging (or simplifying) the two corresponding parallel arcs in the dual.

Spanning trees and forests

One of the most important results relating duality and spanning trees is the following theorem:

Theorem 3.46. *The chords of a spanning forest of a graph induce a spanning forest in the dual graph, assuming it exists.*

Proof. Let $\mathcal{G}(\mathbf{V},\mathbf{A})$ and $\mathcal{G}_d(\mathbf{V}_d,\mathbf{A})$ be two dual (planar) graphs. Let $\mathcal{F}(\mathbf{V},\mathbf{A}_s)$ be a spanning forest. We want to prove that $\mathcal{F}_d(\mathbf{V}_d,\mathbf{A}_{s_d})$, with $\mathbf{A}_{s_d} = \mathbf{A} \setminus \mathbf{A}_s$, is a spanning forest of $\mathcal{G}_d$. First we prove that $\mathcal{F}_d$ is acyclic. Then we prove that it has exactly $\#\mathbf{A}_{s_d} = \#\mathbf{A} - \#\mathbf{A}_s = \#\mathbf{V}_d - c_d = \rho(\mathcal{G}_d)$, where c_d is the number of connected components of $\mathcal{G}_d$. It is also clear that $\mathcal{F}_d$ is a subgraph of $\mathcal{G}_d$. Together, these facts prove that $\mathcal{F}_d$ is indeed a forest of $\mathcal{G}_d$.

$\mathcal{F}_d$ is acyclic: Suppose that $\mathcal{F}_d$ has a circuit $\mathbf{C}$. Circuit $\mathbf{C}$ is also a circuit of $\mathcal{G}_d$, obviously. The arcs in $\mathbf{C}$ are a cutset in $\mathcal{G}$. This cutset must contain at least one branch b of $\mathcal{F}$. But then $b \in \mathbf{A}_s$ and $b \in \mathbf{A}_{s_d} = \mathbf{A} \setminus \mathbf{A}_s$, which is a contradiction. Hence, $\mathcal{F}_d$ is acyclic.

$\#\mathbf{A}_{s_d} = \rho(\mathcal{G}_d)$: By Theorem 3.45 $\rho(\mathcal{G}) = \mu(\mathcal{G}_d)$, i.e., $\rho(\mathcal{G}) = \#\mathbf{A} - \rho(\mathcal{G}_d)$. But, since $\mathcal{F}$ is a spanning forest, $\#\mathbf{A}_s = \rho(\mathcal{G})$. Hence, $\#\mathbf{A}_{s_d} = \#\mathbf{A} - \#\mathbf{A}_s = \rho(\mathcal{G}_d)$. □

Definition 3.68. (dual spanning forest) *Given a spanning forest $\mathcal{F}(\mathbf{V},\mathbf{A}_s)$ of graph $\mathcal{G}(\mathbf{V},\mathbf{A})$ with dual $\mathcal{G}(\mathbf{V}_d,\mathbf{A})$, the subgraph $\mathcal{F}_d(\mathbf{V}_d,\mathbf{A} \setminus \mathbf{A}_s)$ is the dual spanning forest of $\mathcal{F}$.*[12]

Corollary 3.47. *Given a spanning forest in a graph and its dual spanning forest in the dual graph, the fundamental circuits in one graph correspond to the fundamental cutsets one the other.*

Proof. Let $\mathcal{F}(\mathbf{V},\mathbf{A}_s)$ be a spanning forest of $\mathcal{G}(\mathbf{V},\mathbf{A})$ with dual $\mathcal{G}_d(\mathbf{V}_d,\mathbf{A})$, and $\mathcal{F}_d(\mathbf{V}_d,\mathbf{A}\setminus\mathbf{A}_s)$ its dual spanning forest. Let b be a branch of $\mathcal{F}$. Let $\mathbf{C}$ be its corresponding cutset in $\mathcal{G}$. Since fundamental cutsets contain only one branch, $\mathbf{C} \setminus \{b\}$ consists solely of chords. But $\mathbf{C}$, by duality, is a circuit in $\mathcal{G}_d$. The chords of $\mathcal{F}$ are the branches of $\mathcal{F}_d$, and vice versa. Hence, $\mathbf{C}$ is a circuit in $\mathcal{G}_d$ which contains only one chord of $\mathcal{F}_d$. Hence, it is a fundamental circuit. The proof that fundamental circuits correspond to fundamental cutsets follows the same reasoning. □

Given a spanning forest $\mathcal{F}$ of a connected graph $\mathcal{G}$ with geometrical dual $\mathcal{G}_d$, let $\mathcal{F}_d$ be the dual spanning forest of $\mathcal{F}$. If a branch is removed from $\mathcal{F}$, the number of trees, or connected components, in the forest will increase by one. If the same branch of $\mathcal{F}$ is inserted into $\mathcal{F}_d$, a circuit is created, or, using Euler's theorem, a region is introduced by splitting an existing region in two. This process can be continued as long as there are branches to remove from $\mathcal{F}$ and insert into $\mathcal{F}_d$, increasing successively the connected components of $\mathcal{F}$ and the number of regions in $\mathcal{F}_d$. When a region is split in two in $\mathcal{F}_d$, those two regions are limited by a circuit

[12]Note the subtlety: a dual spanning forest is not the dual of a spanning forest!

which, in the geometrical representation, envelops the vertices of the two connected components obtained in $\mathcal{F}$.

It is clear that the process of removing $k - c$ branches from a spanning forest with c connected components leads to a spanning k-tree of the same graph. The corresponding operation in the dual spanning forest is the creation of circuits. Hence, the dual of a k-tree is a pseudo-forest in which some arcs are allowed to be circuit arcs.

Shortest spanning trees and forests

It was shown in the previous section that the chords of a spanning forest of a graph induce a spanning forest in the dual graph. A stronger result is proved here:

Theorem 3.48. *The dual spanning forest of a SSF is a LSF (and vice versa).*

Proof. Let $\mathcal{F}(\mathbf{V}, \mathbf{A}_s)$ be an SSF of graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ with dual $\mathcal{G}_d(\mathbf{V}_d, \mathbf{A})$. Let $\mathcal{F}_d(\mathbf{V}_d, \mathbf{A} \setminus \mathbf{A}_s)$ be the dual spanning forest of $\mathcal{F}$. Take any branch b of $\mathcal{F}$ and its corresponding fundamental cutset $\mathbf{C}$ in $\mathcal{G}$ relative to forest $\mathcal{F}$. According to Theorem 3.11, $w(b) \leq w(c)$ with $c \in \mathbf{C}$. By Corollary 3.47, $\mathbf{C}$ is also a fundamental circuit of $\mathcal{F}_d$ with chord b. Hence, given that $w(b) \leq w(c)$ with $c \in \mathbf{C}$, and again by Theorem 3.11, $\mathcal{F}_d$ is a LSF. $\square$

3.4.3 Four-color theorem

In mathematical terms, a graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ is k colorable if there is a color function $c(\cdot) : \mathbf{V} \to \{1, \ldots, k\}$ such that $c(v_1) \neq c(v_2)$ whenever $\{v_1, v_2\} \in \mathbf{A}$. Notice that, while multi- and simple graphs $\mathcal{G}(\mathbf{V}, \mathbf{A})$ are always $\#\mathbf{V}$ colorable, pseudographs are only colorable if they possess no self-connecting arcs, i.e., if they are multi- or simple graphs.

The problem of asserting whether a general graph is $k \geq 3$ colorable is NP-complete [51]. That is, there is no deterministic algorithm able to solve the problem in polynomial time [51].[13]

An interesting property of multi- or simple planar graphs is that they can be colored using four colors. That is, it is possible to assign one of four different colors to each vertex of the graph such that vertices which are end vertices of the same arc have different colors. This was conjectured in 1852 by Francis Guthrie, a student of De Morgan, and remained a conjecture until 1976, when Appel and Haken published the proof of the FCT (Four-Color Theorem) which involved thorough computer assisted proofs, impossible to verify by hand. In 1996, Robertson et al. [170] published a simpler proof, even though using similar methods, and, in passing, also obtained a quadratic algorithm for coloring a planar graph with four colors, i.e., a $O(\#\mathbf{A}^2)$ algorithm, $\mathbf{A}$ being the set of arcs of the planar graph to be colored.

[13]If P$\neq$NP, see [51].

3.5 Maps

What are maps? How can the relations between their regions be represented? What is peculiar in discrete maps? Maps can be defined over continuous or discrete spaces, of any dimension, just like images. All functions from such a space into a finite set can be though of as a (function) map. The elements in this finite set can be interpreted as labels of a certain class. In a sense, thus, maps and partitions, to be defined in Section 3.6.1, are one and the same concept. However, the term partition will often be more specifically associated to a map which results from a segmentation process (see Definition 3.71).

A map was defined above as a function from a space to a set of labels. It can also be seen as a partition of the space into disjoint subsets of the space (classes), each corresponding to a label. The study of the spatial relations between these disjoint subsets is the subject of topology. In the case of a discrete and finite space, which is of paramount interest in the field of automatic analysis of images, finite topology can be used. Kovalevsky, in two excellent papers on finite topology [91, 92], demonstrated that cellular complexes, a finite topology construct, allow to unambiguously represent neighborhood relations between the subsets of a map, and this independently of the space dimension. More than that, his results demonstrate that pixel relationships are insufficient for this purpose. The edges and vertices of the pixels (see Definition 3.79), in the case of a 2D discrete space, are fundamental. This had already been recognized intuitively by several generations of image analysis theorists, though it had never before been demonstrated formally.

This thesis uses a related but not equivalent concept. By the use of duality, the relationships between the classes, or better, between the regions in a map are represented simultaneously by two graphs: the RAMG (Region Adjacency MultiGraph) and the RBPG (Region Border PseudoGraph) (see Definitions 3.85 and 3.92).[14] Even though this representation works well for 2D maps, for 3D maps the notions must be extended: the borders no longer form a graph, and duality must be redefined. Also, the proposed method of representation does not fully solve the ambiguity problems of which cellular complexes are free. The solution of both problems, through a reformulation of the results on graphs for the broader theory of cell complexes, has been left for future work on the subject. However, unlike the RAG [154], this pair of graphs, RAMG and RBPG, retains information about the continuity of the borders.

3.5.1 Operations on the dual RAMG and RBPG graphs

Consider a 2D image and the corresponding 2D embedding of its corresponding planar image graph. Consider also the geometrical dual graph of this embedding. These graphs together represent the spatial relationships between the pixels, regarded as individual regions. The first graph is the RAMG, and the second is the RBPG, if all classes in the map have a single pixel.

What happens when two adjacent regions, that is, adjacent in the RAMG, are merged together?

[14]RAMG is used even though the word graph, in this thesis, refers by default to a pseudograph (and hence also to multigraphs). This was done to distinguish it from the RAG (Region Adjacency Graph), which is a simple graph. For reasons of coherence, the border graph was named accordingly.

Clearly, the corresponding vertices of the RAMG are short-circuited. But this results in at least one self-connecting arc. Such self-connecting arcs have no role to play in the RAMG, since they say nothing about relations between regions. Hence, they must be eliminated. What are the corresponding operations in the dual graph? Since short-circuiting of two adjacent vertices and removal of one of the arcs between them is actually a contraction of this arc, its dual operation is simply the removal of the corresponding arc from the RBPG. The removal of the other now self-connecting arcs of the RAMG can also be seen as special cases of contraction, its dual operations also being removal from the RBPG. But after removing such arcs, the RBPG may have been left redundant, in the sense that it may contain some redundant vertex, that is, a vertex of degree two which has not a self-connecting arc. If this happens, arc reduction can be performed on this vertex. Since arc reduction is the same as contraction of one of the arcs incident on the vertex of degree two, the corresponding operation on the RAMG is removal of that arc.

Contraction and removal of a self-connecting arc have the same result for the RAMG, but altogether different results in the case of the dual. Contraction corresponds to removal in the dual and vice-versa. The appropriate operation is thus contraction in the RAMG and removal in the RBPG, since otherwise an artificial connection of two disconnected border sets would be introduced. The result would still be a valid map, actually it would be a 2-isomorphism of the result obtained as suggested. However, it would not have a correspondence in the real map, defined as a partition of the space.

3.5.2 Definition of map

It is important to realize two facts about the RAMG. First, it must be a connected graph: even in the unlikely event that the 2D image is defined on several non-contiguous subsets of the space, one can always add a background region to the map and thus render the RAMG connected. If the 2D image is defined on a subset of the space whose pixels are connected in the corresponding image graph, the result is obvious. Secondly, there cannot be any bridges in the RBPG, since otherwise that arc would not separate two regions. Equivalently, the RAMG cannot have any self-connecting arcs (which are the duals of bridges). As an immediate consequence, the RBPG is 2-arc-connected.

To sum up:

1. Self-connecting arcs play no role in the RAMG. This explains why the RAMG is a multi-graph.
2. All vertices of degree two in the RBPG are connected to themselves by a self-connecting arc.
3. The operation of merging two regions in a map[15] corresponds to short-circuiting the two corresponding vertices in the RAMG, removing the self-connecting arcs created in the process, and finally arc-reducing the vertices of degree two in the RBPG which are not isolated, if any.

[15]Annexation or invasion, in geopolitics parlance.

Hence, the operations on the dual adjacency and border graphs can be though of as the merging itself, followed by the dual operations necessary to render this pair of graphs proper. It is now possible to define the concept of a map and a proper map:

Definition 3.69. (map) *A pair of dual graphs, the RAMG and the RBPG, such that the RAMG is the geometrical dual of a given embedding of the RBPG (and hence is connected).*

Definition 3.70. (proper map) *A pair of dual graphs, the RAMG and the RBPG, forming a map, such that the RAMG has no self-connecting arcs (redundant adjacency information) and the RBPG has no vertices of degree two, except if isolated (that is, there is no homeomorphic graph of the RBPG which is smaller than the RBPG).*

In maps, region inclusion relations correspond to cut vertices in the RAMG. These vertices correspond to regions with, if eliminated, lead to two disconnected maps. A cut vertex in a planar graph without self-connecting arcs defines a cut (the arcs which incident on it), composed of cutsets (each cutset is composed of the arcs with connect the vertex to a different block). These cutsets are disjoint. In the dual they correspond each to arc-disjoint circuits. But the vertices correspond, in the dual, to a region or face (by construction, there is a single vertex of the RAMG in each region of the RBPG). Hence, that region is limited by more than one circuit, that is, it has "holes". If there are n cutsets in the cut, there are $n-1$ holes in the region, which is limited, in the planar embedding, by the remaining circuit $(1+n-1=n)$.

For vertices which are not cut vertices, the set of their arcs is a cutset, and hence corresponds, in the dual, to a single circuit. Hence, such regions have no "holes".

3.5.3 Algorithms

Given a function map, the corresponding map can be obtained by building first a fictitious map where each pixel corresponds to a different region. As seen above, this fictitious map is simply the image graph and its geometrical dual. The map can be obtained by merging successively adjacent regions with the same label, using the operations defined above. Alternatively, one might start with a single region, encompassing all pixels, and successively split non-uniform regions, i.e., regions containing different labels.

Often the function map is defined implicitly by the values of the pixels of an image. This is the case before a segmentation process is performed. In this case, the map can be built on the fly, while the segmentation proceeds. Actually, most segmentation algorithms rely on the map structure to store information about regions and borders. Regions can contain the set of corresponding pixels and statistics of their values, just as borders can contain sets pixel borders and statistics of their values.

3.6 Partitions and contours

In this section the notions of segmentation as the process leading to a partition, and the notions of class, region and borders are defined.

3.6.1 Partitions and segmentation

Definition 3.71. (segmentation) *Process of classifying each pixel in a digital image or sequence as belonging to a certain class with certain properties. The class properties are assumed to be representable by vectors of parameters (or statistics). Hence, after segmenting an image $f[\cdot]$ one obtains:*

1. *the number l of classes that were found (this value may be fixed a priori);*
2. *the partition, i.e., a function $p[\cdot] : \mathbf{Z} \to \mathbf{L}$ which classifies each pixel (see definition below); and*
3. *possibly a sequence p_i, with $i = 0, \ldots, l-1$, of parameter vectors.*

This definition of segmentation is generic. Stricter definitions will be given as needed in Chapter 4.

Definition 3.72. (partition) *A digital image $p[\cdot] : \mathbf{Z} \to \mathbf{L}$ (or $p[\cdot] : \mathbf{N} \times \mathbf{Z} \to \mathbf{L}$ in the case of 3D partitions) taking values in $\mathbf{L} = \{0, \ldots, l-1\}$, where the value of each pixel is a label identifying the class to which the pixel belongs.*

Definition 3.73. (binary partition) *A partition with $l = 2$ is said to be a binary partition, since it is a binary image taking only values 0 and 1.*

Definition 3.74. (mosaic partition) *A partition with $l > 2$ is a mosaic partition.*

3.6.2 Classes and regions

Definition 3.75. (class) *The set of all pixels in a partition, or vertices of the associated image graph, having a specific label. Class c, i.e., the set $\mathbf{V}_c$, of a partition defined in $\mathbf{Z}$ is given by: $\mathbf{V}_c = \{v \in \mathbf{Z} : p[v] = c\}$ (or $\mathbf{V}_c = \{v \in \mathbf{N} \times \mathbf{Z} : p[v] = c\}$ in the case of a 3D partition).*

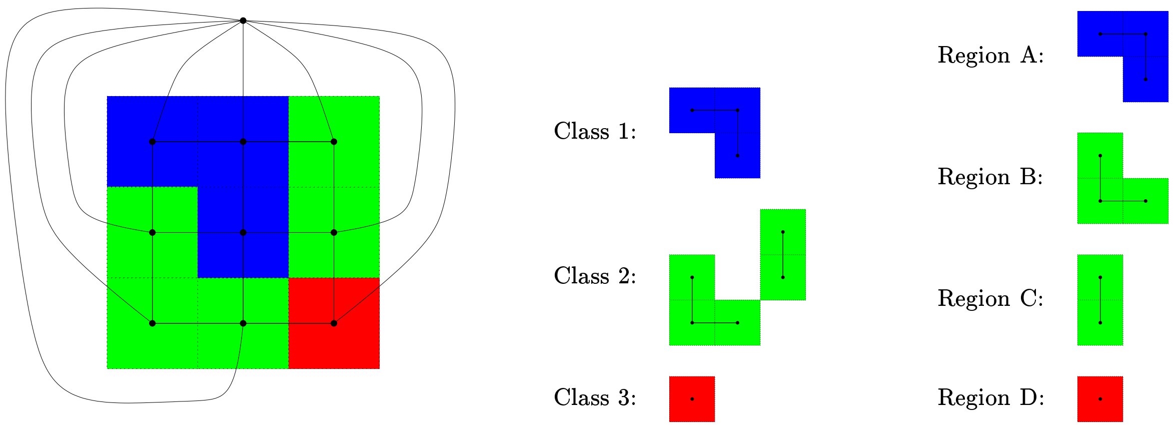
Definition 3.76. (class graph) *The maximal subgraph of the image graph $\mathcal{G}(\mathbf{V}, \mathbf{A})$ (where $\mathbf{V} = \mathbf{Z}$ for 2D partitions and $\mathbf{V} = \mathbf{N} \times \mathbf{Z}$ for 3D partitions) induced by a class c, i.e., by the set of vertices $\mathbf{V}_c \subseteq \mathbf{V}$.*

Definition 3.77. (region graph) *A connected component of a class graph.*

Definition 3.78. (region) *A set of pixels in a partition which are the vertices of a region graph. Regions are thus components of classes.*

The terms class and class graph (and also region and region graph) will be used interchangeably. The meaning should be deducible from the context.

The main difference between classes and regions, both containing pixels with the same label, is that the latter are always connected while the former can be disconnected, see Figure 3.6. If a class is connected, then it consists of a single region. Notice that no connectivity restrictions were imposed to the classes in the definition of segmentation. Several stricter definitions of segmentation impose connected classes. If such is the case, and if this is clear from the context, the term region will be used instead of class.



(a) Partition and the corresponding image graph. (b) Class graphs. (c) Region graphs.

Figure 3.6: Example of a partition on a rectangular lattice with an associated N_4 graph. The partition has three classes (and hence three class graphs) and four regions (and thus four region graphs).

Edges, borders, boundaries, and contours

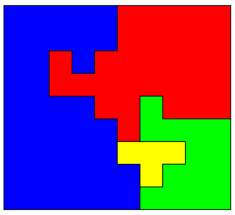
The trivial way to represent a partition is by specifying the labels of each of its pixels: this is actually what Definition 3.72 says. However, it is often more natural to represent a partition by specifying the boundaries of its regions. Such a representation is sufficient if region equivalence is sufficient (see Section 3.6.4). If class equivalence is desired, then, in the case of partitions with disconnected classes, further information is required, namely which regions belong to each class.

Definition 3.79. (border, edge and face) *A border is a continuous line between two adjacent regions, in the case of 2D partitions. The borders do not contain points of departure of any other borders (see Figure 3.7). If both regions correspond to a single pixel, the border is an edge. In the case of 3D partitions, a border is a contiguous surface between two adjacent regions. The 3D concept corresponding to the 2D edge is the face.*

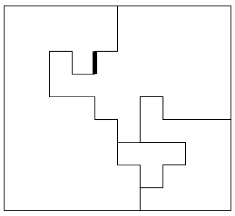
Edges have a (relative) length which depends on their orientation and on the shape of the pixel in the associated sampling lattice, if there is one. In the case of rectangular sampling lattices, this dependence can be written in terms of the pixel aspect ratio. The measure associated with borders in the 3D case is an area. However, since one of the dimensions of the 3D partition is usually time, this area does not have an immediate physical interpretation.

Definition 3.80. (boundary) *The union of the borders of a class or region. The length of the boundary is a length proper (actually a perimeter, since boundaries are always closed) for 2D partitions and an area for 3D partitions.*

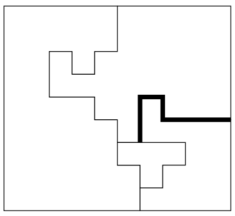
Definition 3.81. (contour) *The union of all class boundaries in a partition. Notice that the*



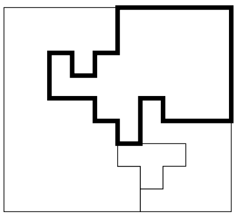
(a) Partition.



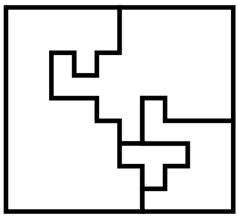
(b) An edge.



(c) A border.



(d) A boundary.



(e) The contour.

Figure 3.7: A partition, its contour, and examples of an edge, a border, and a boundary.

length of the contour of a partition is half the sum of the boundary lengths of all classes, since adjacent classes share, by definition, the common borders.

A partition can thus be (partially) represented by specifying the boundaries of its regions.

3.6.3 Region and class graphs

Two types of graphs, besides image graphs, can be defined over partitions: the RAG and the CAG (Class Adjacency Graph). The definition of both makes use of the concept of adjacency:

Definition 3.82. (adjacency) *Two sets of vertices* $\mathbf{V}', \mathbf{V}'' \subseteq \mathbf{V}$ *of graph* $\mathcal{G}(\mathbf{V}, \mathbf{A})$ *such that* $\mathbf{V}' \cap \mathbf{V}'' = \emptyset$ *are said to be adjacent if there is at least one arc* $\{v', v''\} \in \mathbf{A}$ *such that* $v' \in \mathbf{V}'$ *and* $v'' \in \mathbf{V}''$*.*

Definition 3.83. (RAG) *A simple graph with as many vertices as regions in a given partition, plus an extra region representing the outside of the partition domain* $\mathbf{Z} \subset \mathbb{Z}^2$ *(or* $\mathbf{N} \times \mathbf{Z} \subset \mathbb{Z}^3$*). There is a single arc between any pair of vertices corresponding to adjacent regions in the partition, i.e., there is a single arc between any two regions sharing at least one border in the partition.*

The RAGs of partitions associated with N_4 and N_6 graphs are always planar. A RAG can be obtained from the image graph of a partition by successively performing arc contractions on arcs incident on vertices belonging to the same class.

Definition 3.84. (CAG) *A simple graph with as many vertices as classes in a given partition, plus an extra class representing the outside of the partition (as above). There is a single arc between any pair of vertices corresponding to adjacent classes in the partition, i.e., there is a (single) arc between any two classes sharing at least one border in the partition.*

The CAGs of partitions associated with N_4 or N_6 graphs may be non-planar. If classes are connected, the CAG is equal to the RAG (and hence planar). The CAG can be obtained from either the image graph of the partition or from the RAG by successively short-circuiting pairs of vertices belonging to the same class and removing the resulting self-connecting arcs.

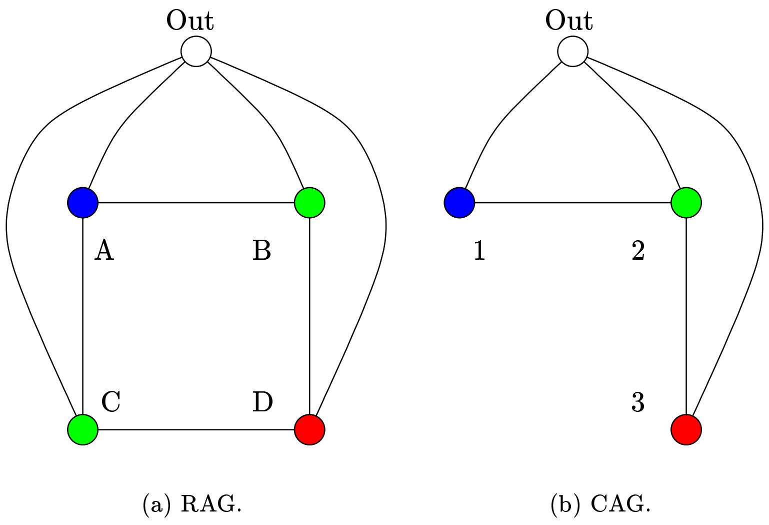
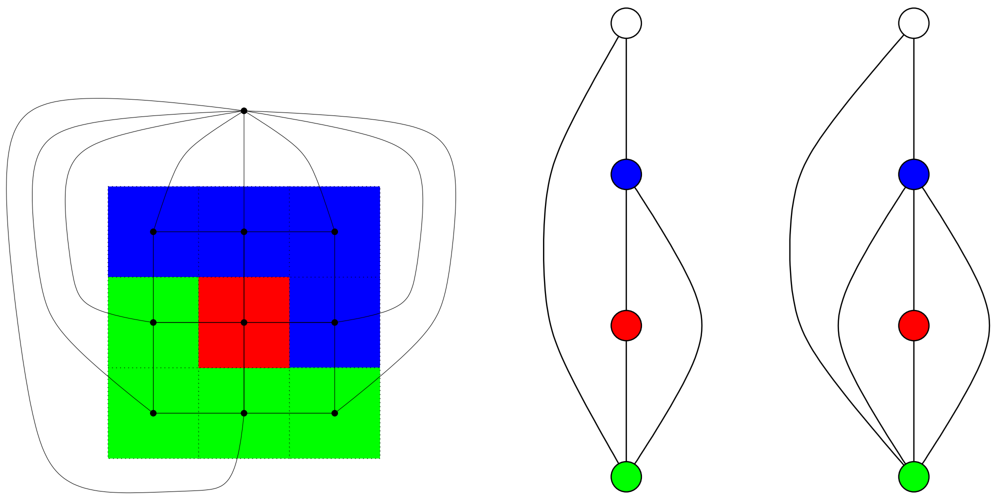


Figure 3.8: RAG and CAG corresponding to the partition in Figure 3.6(a). The circles are the graph vertices and correspond to regions and classes, respectively. The circle labeled "Out" is the external region or class.

A more powerful graph, which, together with the RBPG, defines the map of a partition, is the RAMG:

Definition 3.85. (RAMG) *A multigraph with as many vertices as regions in a given partition, plus an extra region corresponding to the outside of the partition domain. There is an arc for each border between regions in the partition. The arcs connect the two regions which are adjacent through the corresponding border. Hence, two regions can be connected by more than one arc, if they are adjacent through more than one border.*

Unlike the case of the RAGs, RAMGs cannot be built ignoring the topology of the boundaries. They can, though, be built as indicated in Section 3.5.3.



(a) Partition and the corresponding image graph. (b) RAG. (c) RAMG.

Figure 3.9: RAG and RAMG corresponding to a given partition. The white vertex is the external region.

3.6.4 Equivalence and equality of partitions

An important concept when dealing with partition coding is that of equivalence between partitions:

Definition 3.86. (class and region topological equivalence of partitions) *Two partitions $p_1[\cdot]$ and $p_2[\cdot]$ with the same labels are class (region) topologically equivalent if their corresponding CAGs (RAGs) are isomorphic through the identity function on labels.*

A stricter form of topological equivalence can be used in which the RBPG graphs[16] of the proper maps of the two partitions are required to be isomorphic.

Definition 3.87. (class and region equivalence of partitions) *Two partitions are class (region) equivalent if they divide an image into equal classes (regions). Mathematically, partitions $p_1[\cdot] : \mathbf{Z} \to \mathbf{L}_1$ and $p_2[\cdot] : \mathbf{Z} \to \mathbf{L}_2$, defined in $\mathbf{Z}$, are said to be class equivalent if it is possible to find a function $f[\cdot] : \mathbf{L}_1 \to \mathbf{L}_2$ which is bijective between the classes used in partitions $p_1[\cdot]$ and $p_2[\cdot]$. That is:*

$$f(p_1[v]) = p_2[v] \ \forall v \in \mathbf{Z}$$
$$f^{-1}(p_2[v]) = p_1[v] \ \forall v \in \mathbf{Z}$$

A class is used in a partition if there is at least one pixel in the partition with the corresponding label.

[16]Or the RAMG, for that matter.

Equality is defined trivially:

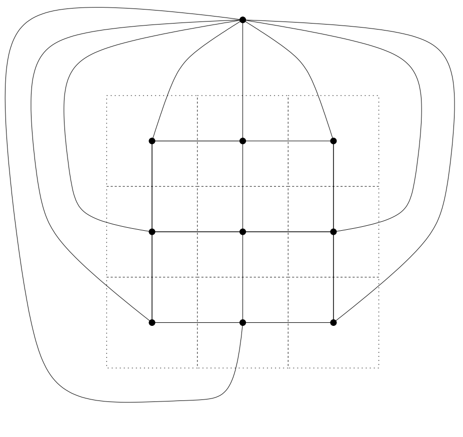
Definition 3.88. (equality of partitions) *Two partitions are said to be equal if, apart from being class equivalent, the labels of each class are equal in both partitions. Or, which is the same, if the corresponding digital partition images are equal.*

Line, edge, and border graphs

In 2D, contours can be conveniently defined over a line graph, which is the dual of a planar image graph. For 3D partitions more complicated structures are required. This issue will not be discussed here, since often the 3D partitions are taken as sequences of 2D partitions, which is even more natural in the case of partitions of moving images.

Definition 3.89. (line graph) *Planar simple graph obtained by geometrical duality from the (natural) embedding of the (connected) planar image graph (with an extra external pixel).*

Figure 3.10 shows an N_4 image graph (on a rectangular lattice) and the corresponding line graph, which is also N_4. The line graph corresponding to the N_6 image graph, e.g., on a hexagonal lattice, is N_3, as can be easily verified.


(a) Image graph.

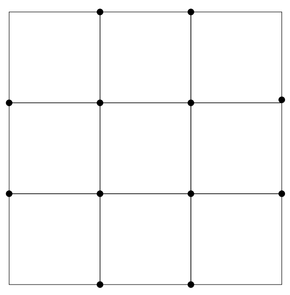

(b) Line graph.

Figure 3.10: A N_4 image graph and its dual line graph, also N_4.

The line graph of a 2D partition is thus obtained by duality of its planar image graph. The contour of a partition can be represented by the subgraph of the line graph containing all vertices and arcs standing between pixels with different labels (i.e., belonging to different classes). This is the edge contour graph:

Definition 3.90. (edge contour graph) *A subgraph (planar and simple) of the line graph corresponding to the boundaries of the classes in the partition. An arc in the line graph belongs to the edge contour subgraph if its corresponding arc in the dual partition image graph connects pixels with different labels, i.e., which belong to different classes.*

All edge contour graphs are 2-arc-connected, since bridges cannot stand between two classes.

An edge contour graph, i.e., a contour defined on the edges, can thus be constructed as follows:

1. mark the arcs of the (planar) image graph which stand between pixels belonging to different classes (the exterior extra pixel can be assumed to belong to a non-existent class);

2. mark the corresponding arcs in the dual line graph, also mark the end vertices of these arcs;

3. the edge contour graph is composed of the marked arcs and vertices in the line graph.

Definition 3.91. (contour) *A function* $c[\cdot] : \mathbf{A} \to 0, 1$ *marking the arcs of a line graph* $\mathcal{G}(\mathbf{F}, \mathbf{A})$ *as belonging or not to the edge contour graph.*

Notice that the edge contour graph, and hence the partition (up to region equivalence), can be obtained from $c[\cdot]$. The same observation would not be true for a function marking the edge contour graph vertices, since ambiguity might occur, as shown in Figure 3.11.

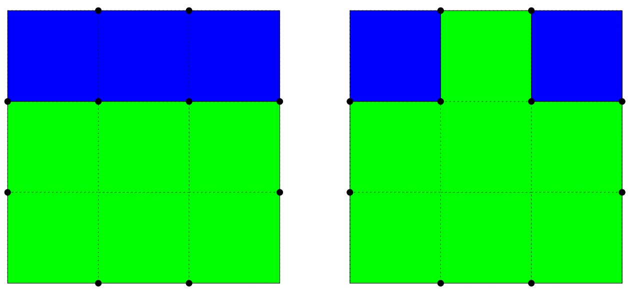


Figure 3.11: Example of ambiguity for vertex based contour definitions on edge contour graphs. Two partitions with the same edge contour graph vertices.

Contours may also be defined in the image graph itself, that is, with its vertices corresponding to the pixels of the image. In this case, a contour might consist of a function marking all those pixels with neighbors belonging to a different class. However, this leads to thick contours, since pixels at both sides of a border between two regions are marked. This problem may be solved by marking only one side of each border.

In the case of contours defined on pixels, a graph can also be associated with the contour. This graph will be called the pixel contour graph, to distinguish it from the edge contour graph. The pixel contour graph corresponds to the maximal subgraph of the image graph whose vertices have been deemed to belong to a border, i.e., to belong to the contour. Notice, however, that if a contour is defined on pixels over a N_8 graph, it may be necessary to purge some of the contour pixels, obtained using the criterion above, and also some of the contour graph arcs (see [133]). Contours on pixels are plagued by many inconsistencies, which are not discussed in this thesis [181, 91, 92, 31].

Generally, the contour information allows only for a representation of partitions up to region equivalence. If class equivalence, or equality, is required, then information about which regions belong to which classes (region-class information) is necessary.

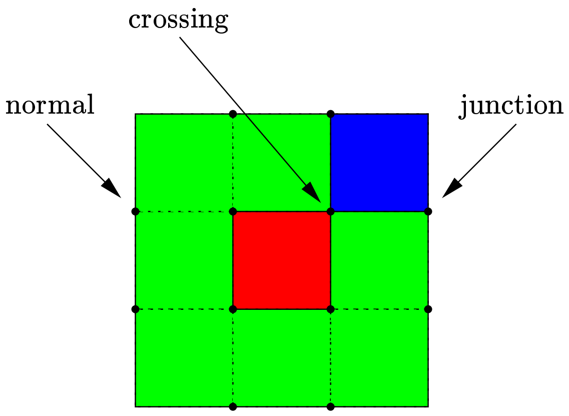
The contour graphs, edge- or pixel-based, can contain several types of contour vertices, according their degree (see Figure 3.12):

Degree 1
: Dead end vertex; these vertices exist only for contours defined on the pixels.[17]

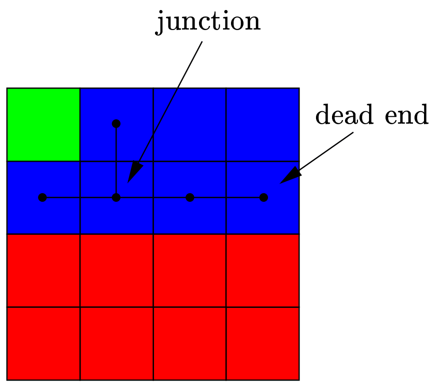
Degree 2
: Normal vertices.

Degree 3
: Junction vertices.

Degree 4
: Crossing vertices.



(a) Edge contour graph.



(b) Pixel contour graph.

Figure 3.12: Types of vertices on contour graphs.

The maximal reduction of the edge contour graph is the RBPG (of which the RAMG is the geometrical dual):

Definition 3.92. **(RBPG)** *A graph having has many vertices as there are vertices of degree larger than two in the edge contour graph plus as many vertices as there are components of the edge contour graph consisting of a single circuit. Each arc corresponds to a path (possibly closed) in the edge contour graph containing only vertices of degree two (i.e., to sets of contiguous edges forming borders).*

[17]However, in a more general definition of contours, where contours are not the dual of some partition, these vertices do occur even for contours defined on the line graph. Such techniques which use a more general definition of contour may be used for the "edge-based description of color images," see [37, 19, 58].

Since the RBPG is the maximal reduction of the edge contour graph, it obviously contains no vertices of degree two, except possibly isolated vertices with a self-connecting arc.

3.7 Conclusions

The graph theoretical foundations of image analysis were presented. A thorough discussion of seeded SST concepts and algorithms, namely for finding the SSF, the SSkT, the SSSkT, or the SSSSkT of graph, has been done. From this work resulted a new asymptotically linear amortized time algorithm for finding multiple SSSSkTs, for different sets of seeds, of the same graph. The relation between the SST and dual graphs, which plays an important role in proving that basic region merging and basic contour closing are one and the same algorithm, solving the same problem (see Chapter 4), has been established.

Chapter 4

Spatial analysis

The best way of finding out the difficulties of doing something is to try to do it.

David Marr

This chapter deals with spatial analysis, which, strictly speaking, is the analysis of still images. However, moving images will also be taken into account. Time analysis proper, or motion analysis, is the subject of the next chapter.

Even though there has been intense research in this area of knowledge, results are still far from the final goal: the construction of a model of reality and its full understanding. The goal attainable, for the time being, is to extract mid-level vision primitives. Most of the work presented here, and most of the contributions of this thesis, can be classified as pertaining to second-generation video coding.

Segmentation is a very important step in analyzing a scene, i.e., in obtaining a structured description for it. Section 4.1 introduces briefly the subject of segmentation and Section 4.2 presents a hierarchy of the tools involved in segmentation process and an overview of some specific segmentation tools, especially those related to contour-oriented segmentation. Section 4.3 then deals with several classes of region-oriented segmentation algorithms and attempts to frame them within the same theoretical framework. The dual relation between region- and contour-oriented segmentation is also explained within the same framework.

The evolution path in spatial analysis, in the framework of video coding, has passed through transition zones, when going from first-generation, low-level analysis techniques, to second-generation, mid-level analysis techniques. Section 4.4 presents a knowledge-based segmentation technique for videotelephony applications, capable of dealing with mobile environments. It attempts to make a coarse segmentation of head-and-shoulders mobile videotelephony sequences into three disjoint regions: head, body, and background. Different qualities, and thus different bitrate assignments, can be attributed to each region. This may improve subjective quality of first-generation encoders by incorporating simple second-generation, mid-level analysis techniques. These revamped encoders maintain compatibility with existing decoders, though providing better subjective quality. Such encoders can be said to belong to the transition layer between first- and second-generation.

Section 4.5 presents contributions in the area of generic color (or texture) segmentation. These contributions belong to the area of mid-level analysis, second-generation video coding. One of the segmentation algorithms proposed, which is related both to split & merge and to RSST segmentation, is then extended in Section 4.6 to support supervised segmentation. The importance of supervision stems from the fact that, as stated before, supervision can be seen as a first, pragmatic step towards third-generation, high-level analysis.

Finally, in Section 4.7, the RSST techniques presented before are extended to allow segmentation of sequences of (moving) images in a recursive way, so as to maintain the time coherence of the attained segmentation. The resulting technique has been coined TR-RSST. It can be seen as a step in the direction of the integration of time and space analysis.

4.1 Introduction to segmentation

The identification of regions (or objects) within an image or sequence of images, i.e., image segmentation, is one of the most important steps in second-generation (object- or region-based) video coding, and hence in mid-level analysis.

If a partition of a set **S** is defined as a set **R** of subsets of **S** such that the union of all the elements of **R** is **S** and such that, for all $r_1 \neq r_2 \in \mathbf{R}$, $r_1 \cap r_2 = \emptyset$, then the definition of segmentation is apparently simple: produce a partition of the set of image pixels so that each set (class or region) in the partition is uniform according to a certain criterion and so that the union of any two sets in the partition is non-uniform.

In the words of Pavlidis [156] "segmentation identifies areas of an image that appear uniform to an observer, and subdivides the image into regions of uniform appearance," where uniformity can be defined in terms of grey level (or color) or texture. One can, however, envisage another kind of segmentation where one expects to identify certain known objects in an image. According to Haralick [68], "image segmentation is the partition of an image into a set of non-overlapping regions whose union is the entire image (...) that are meaningful with respect to a particular application."

These definitions are more or less equivalent, though quite vague. Even if an appropriate uniformity criterion is given, they establish no constraint as to the number or connectivity of the regions. Hence, another, possibly more useful, definition may be: produce a partition of the image into a minimum set of connected regions such that a certain global uniformity measure is above a given threshold. Or: produce a partition of the image into a certain number of connected regions such that a certain global uniformity measure is maximized. But the exact definition and the homogeneity criteria are still dependent on the application.

Instead of using uniformity criteria, dissimilarity criteria may also be used. They are in the origin of the edge detection methods, leading to contour-oriented segmentation, but they can also be used in region-oriented segmentation. Segmentation, considering dissimilarity instead of uniformity criteria, requires that all pairs of sets (regions) in the image partition are dissimilar. This type of segmentation is conceptually dual to region-oriented segmentation. Furthermore, it will be shown that there are region-oriented segmentation algorithms which are in fact formally dual to contour-oriented algorithms, in the sense that both produce the same segmentation.

The existence of a wide variety of natural image features (e.g., shadows, texture, small contrast zones, noise, object overlap) makes it very difficult to define robust and generic homogeneity or similarity criteria. A large number of different criteria appears in the literature. The choice of appropriate uniformity criteria depends on the task at hand. If the segmentation aims at identifying the real life objects in the image automatically, e.g., if the image is to be easily manipulated or edited by a human, then criteria will have to be related to the semantic content of the represented scene. Developing such criteria is a daunting task that implies modeling with detail all the levels of the human visual system. It is a high-level vision problem. However, by using simpler criteria, one may render the problem tractable and still hope the results to be of some use for human manipulation. Also, one may envisage mid-level tools attaining high-level results with appropriate supervision. As a first approach, the supervision may be performed by a human, but evolution may render it possible to build automatic supervision tools.

Once an appropriate uniformity criterion or measure has been established, there are many different algorithms for achieving the desired segmentation.

4.2 Hierarchizing the segmentation process

This section intends to hierarchize the possible actors in a segmentation process, from low-level operators to high-level algorithms, and to describe some of the main tools used at the various hierarchical levels. The segmentation methods approached here will be restricted to the mid-level vision level, and hence with little or no semantical ambitions.

In a segmentation process three hierarchical levels may be considered (although the division is more or less arbitrary):

1. The lower level is the operator level. Usually the segmentation operators have one or two images of a sequence as input. The result usually maps each pixel into one of several categories, serving has a basis for the segmentation of the current image into non-overlapping regions. The result is either a primary division of the image into several non-overlapping regions (for region segmentation operators) or a primary classification of each pixel or of each edge as belonging to a boundary or not (for edge detection segmentation operators). In [108] and [18] examples of the later class of operators can be found.

   Notice that in this context edge means the physical edge of some object in the represented scene, and not an edge in the sense of Definition 3.79. Furthermore, more often than not edge detectors aim at detecting strong transitions in image color, even if not corresponding to physical edges. The wording edge detection is thus used mainly for historical reasons.

2. The middle level is the technique level. Each segmentation technique uses one or several of the segmentation operators to produce an intermediate step of the segmentation process. Usually, e.g., in edge detection segmentation techniques, firstly one or more of the low-level segmentation operators are applied, and finally some processing is performed using topological considerations (like contour closing and clearing of isolated contour pixels).

3. The higher level is the algorithm level. At this level, segmentation algorithms integrate one or more segmentation techniques (and eventually also segmentation operators) to achieve the final segmentation. In the case of algorithms using edge detection techniques, connected component labeling may be used to identify the regions corresponding to the detected contours (if the physical edges detected correspond to closed contours). It also tries to assess and control the overall segmentation quality attained.

Notice that this hierarchical division of segmentation into levels is not related to the levels of vision, and analysis, already mentioned: the operator level, in the case of edge detection operators, is clearly a low-level vision mechanism, while region operators are clearly related to mid-level vision concepts. Also notice that this hierarchizing is not always clearcut. In the case of region-oriented segmentation, for example, only two levels, or even only a single level, are often used.

4.2.1 Operator level

The lower level in the segmentation process is the operator level. There are a wealth of image operators available in literature for use in contour-oriented segmentation techniques. For computational efficiency reasons, these operators usually have a limited region of support, i.e., they correspond, if linear, to 2D FIR (Finite Impulse Response) filters. If G is an operator, and f is the original image, this means that $g = G(f)$, the result of the operator, is such that $g[n, m]$ can be written as a function of the values of $f[\cdot]$ in a limited region, the region of support, centered in $f[n, m]$.

In general, large regions of support correspond to a large computational effort. However, as [179] points out, some IIR (Infinite Impulse Response) filters can be implemented recursively, with a consequent reduction of the associated computational effort.

Spatial features

Segmentation operators usually have one or two images of a sequence as input and produce as output a mapping of each pixel into one of several categories. These categories usually correspond to:

1. a primary division of the image into several non-overlapping regions, for region segmentation operators (the framework is region-oriented segmentation); or
2. a primary classification of each pixel or edge as belonging to a boundary, for edge detection segmentation operators (the framework, in this case, is contour-oriented segmentation).

Vectorial vs. scalar

Segmentation operators can be classified according to the number of image components they operate on:

1. scalar operators operate on a single color component; and
2. vectorial operators operate on more than one color component.

By far the most common operators in the literature are of the scalar type. However, several authors proposed vectorial operators as a good way to cope with error and to detect some features which may be impossible to detect using a single component [35]. Lee and Cok [98] have shown that, if (physical) step edges are highly correlated in all the color components of an image and if noise in each component is uncorrelated (which seems a plausible assumption), then vectorial edge detection operators are less sensitive to noise than the scalar ones. If, on the other hand, the color components of an image are less correlated, some important physical edges may appear in some of the components whilst missing in others. The use of vectorial operators allows the detection of physical edges that would otherwise be missed by scalar operators.

2D vs. 3D operators

Segmentation operators which have only one image as input are 2D operators. On the other hand, image operators that have two or more successive images of a sequence as operands are 3D operators. They operate, thus, on more than one image, and hence may make use of time and motion information in the image sequence.

2D edge detection operators

Most edge detection operators attempt to detect the pixels or edges where the image gradient has a local maximum in at least one direction or where some second derivative of the image has a zero crossing [108]. Most of these operators were developed in order to detect a particular type of transition optimally, such as step, roof or ridge transitions, in the hope that they also detect reasonably well other types of transitions, hopefully corresponding to physical edges. Also, most of them were developed initially for detection of transitions on pixels. However, most of them can be easily adapted to detect transitions on edges.

Since digital images usually possess noise, most of the operators use thresholding techniques and/or filtering in order to condition the derivative estimation problem (Torre and Poggio in [187] show that numerical differentiation is an ill-posed problem in the sense of Hadamard[1]).

Canny [18] proposed that the design of an edge detection segmentation operator should attempt to optimize a functional involving three criteria:

[1]A problem is well posed in the sense of Hadamard if its solution: exists, is unique, and depends continuously on the initial data.

Good detection
Low probability of detecting false physical edges and of not detecting true physical edges. Both decrease monotonically with the image signal to noise ratio.

Good localization
The estimated physical edges should be as spatially close as possible to the true (projected) physical edges.

Single response
There should be only one response to a single physical edge.

Canny [18] minimized numerically the product of the first two criteria with the single response as a constraint. This was only done for one dimensional physical edges, resulting in a filter very similar to the first derivative of a Gaussian. The expansion to two dimensions is done by convolving the one-dimensional edge detector with an appropriate perpendicular projection function. The proposed projection function is a Gaussian with the same standard deviation σ. This operator should be oriented such that the one-dimensional edge detector is normal to the estimated physical edge direction, i.e., parallel to the Gaussian smoothed gradient direction.

Sarkar and Boyer, in [179], extended Canny's optimization to an unlimited region of support filter, and proposed the implementation of such filter using a recursive approach (i.e., using IIR filters instead of FIR filters). The main advantage of this scheme is that the computational effort is the same regardless of the size of the operator (i.e., the standard deviation σ).

Marr and Hildreth [108] proposed to use the zero crossings of the LoG (Laplacian of Gaussian), described below. Their method, as opposed to the one proposed by Canny [18], is not directional. Besides, as pointed out in [200], zero crossing operators basically divide the image pixels into three classes ($+$, $-$, and 0) which can be thought to color the image plane. However, it is know that, in general, four colors are required to represent arbitrary 2D partitions. So, zero crossing operators have an inherent difficulty in segmenting arbitrary images.

Haralick, in [66], proposed a technique which is similar to Canny's, the main differences being that the localization uses the zero crossings of the second derivative in the gradient direction, and that the derivatives are calculated using interpolation.

In the literature one can rarely find precise descriptions of the various operators (e.g., in [108, 18]). This has led to considerable difficulty in repeating the results presented by the authors, as can be seen by the debate aroused in the mid eighties by [66] (see [60] and [67]).

Several issues have systematically been lacking precise descriptions:

- Since most of the edge detection theories were first built assuming analog images, how should digital filters be build from the corresponding analog ones?
- How to recondition the digital coefficients of the obtained filter so that inconsistencies introduced by the digitalization process are eliminated?
- How exactly are zero crossings detected in second derivative edge detection methods?

- How exactly are gradient local maxima detected in first derivative edge detection methods?
- Where are transitions detected? On the pixels, on the edges, or on both?

These issues are very important and can change quite dramatically the results obtained by applying the same operator to the same images. The mentioned ambiguities and imprecisions have been partially addressed by two review papers on edge detection methods: [8] and [47].

A good review of the most common edge detector operators can be found in [8], where a method is proposed for the fair comparison of several operators (this method is an improvement of the one proposed by Haralick in [66]). Another good review, which also dwells on the ill-posedness character of edge detection, can be found in [187].

Operator components

Bernsen, in [8], proposes the division of the edge detection operators into three components:

Transition strength
The basis for the thresholding which attempts to eliminate the false estimation of physical edges caused by noise.

Edge localization
Attempts to estimate the exact localization of the physical edges detected by thresholding the transition strength.

Derivative computation
Estimates the derivatives.

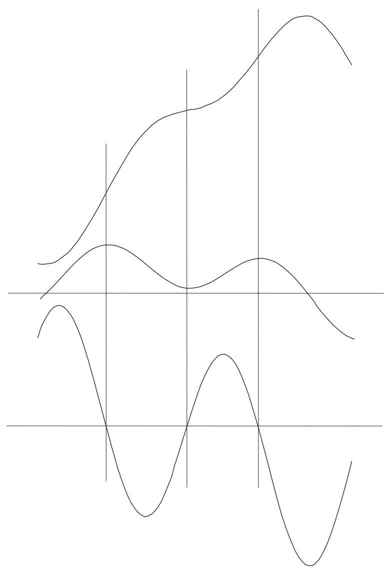
Transition strength

Transition strength is used to separate between real physical edges and image color transitions due to noise. It is usually computed either as the magnitude of the gradient or as the slope of the zero crossings of the second derivative. The latter, however, is much more sensitive to noise than the former, and hence less reliable [8]. The reason for this is that the slope of the second derivative is an approximation of a third order derivative, as opposed to the gradient, which is a first order derivative. Since numerical differentiation is an ill-posed problem, third order derivative estimation is much more sensitive to noise.

Edge localization

Many solutions have been proposed for edge localization. Usually pixels or edges whose transition strength is above a given threshold are considered good candidates for estimated physical edges. However, Canny [18] proposed the use of adaptive thresholding with hysteresis. This method reduces the chances of breaking a contour (a contiguous set of detected pixels or edges), if the threshold of transition strength is set too high, and of estimating wrong physical edges at strong transitions caused by noise, if the threshold is set too low. Two thresholds are used: T_l and T_h, where $T_l < T_h$. Pixels or edges with transition strength above T_l are considered tentative physical edge elements. If a set of connected tentative physical edge elements has at least one element whose transition strength is above T_h, then all the elements of the set will be considered good estimates of physical edges. Otherwise all the elements of the set will be considered *not* to belong to a physical edge.

The threshold schemes have several problems. The first is the possibility of estimation of physical edges several pixels thick, the second is the use of a threshold which is often adjusted by hand. Other schemes, which hopefully avoid these problems, classify as belonging to a physical edge all pixels or edges at a zero crossing of a second order derivative. Marr and Hildreth [108] proposed the use of the Laplacian, which may be said to be an isotropic second order derivative. Haralick [66] proposed the use of the second derivative in the direction of the gradient, but detecting only zero crossings that have negative slope in the gradient direction, so as to avoid detection of false physical edges corresponding to minima, instead of maxima, of the slope. The figure below shows an example. The upper line is a hypothetical luma profile, the middle line and bottom lines correspond to its first and second order derivatives, respectively. The second zero crossing occurs at a point of minimum slope.



How the zero crossings should be detected is not a trivial, especially in the case of detection on pixels, and is rarely described precisely in the literature, albeit considerably different techniques may be used:

1. classify as physical edges those pixels with at least one neighbor pixel having a different signal in the estimated second derivative (this method yields two pixels thick estimated physical edges);
2. classify as physical edges the pixels under the same conditions as before but only those having a specific sign (e.g., positive second derivative); or

3. use a set of predicates for the classification.

This latter solution has been proposed by Huertas and Medioni [75]. A set of predicates is used for the detection of zero crossings in second derivative edge detection methods. They also propose a method to localize physical edges with subpixel accuracy with little extra computational effort.

However, note that the above techniques are not completely specified:

1. Which type of neighborhood should be used?
2. How should "different sign" be interpreted? Should some thresholding be used so that small values of the second derivative are considered as zeros?
3. How should zeros be dealt with?

Since the Laplacian is independent of the coordinate axis chosen, it turns out that its value is equal to the second derivative in the direction of the gradient plus the second derivative perpendicular to the direction of the gradient. For linear physical edges, the later derivative contributes only with noise, and, in the case of a curved physical edge, introduces an offset into the second derivative. This results in a higher sensitivity to noise and larger biases in the estimated physical edge position for the Laplacian methods.

Another method is the so-called "non-maximum suppression in the gradient direction." This method checks whether the magnitude of the gradient is a local maximum in the direction of the gradient.

A further possibility would be to classify tentatively as physical edges all pixels where the magnitude of the gradient is high enough, i.e., the simple thresholding mentioned before, and then to use a thinning technique in order to obtain one pixel thick estimated physical edges.

The chosen solution must take into account the relative merits of each technique both in terms of physical edge localization and in terms of the associated computational effort. Specifically, when efficiency is at a premium, simple solutions should always be considered as good candidates.

Derivative computation

According to Bernsen [8], there are at least four types of methods to compute the derivative:

1. convolve the image with a kernel obtained by sampling the derivatives of the 2D Gaussian function (e.g., LoG [108]);
2. use instead the sampled derivatives of the 2D symmetric exponential function (this method is similar to the first one, the only difference being the kind of smoothing applied to the image prior to derivative calculation);
3. use the derivatives of a tilted-plane approximation to the image in a $n \times n$ window around the given location—up to first order derivatives only; or
4. use instead the derivatives of a third order bivariate polynomial approximation in a $n \times n$ window around the given location—up to third order derivatives only (e.g., [66]).

Bernsen [8] showed that the last method is very similar to the first one for least squares polynomial approximations with Gaussian weights.

Selection of components

According to Bernsen's evaluation [8], the best operators are those that use a gradient magnitude transition strength computation, the zero crossings of the second derivative in the direction of the gradient for edge localization and the sampled derivatives of the 2D Gaussian function.

Examples

In order to illustrate the division into components proposed in [8], two simple and well known 2D scalar edge detection operators are presented in the next section.

Sobel operator

One of the most common 2D scalar edge detection segmentation operators is the Sobel operator [55]. This operator calculates transition strength of an image f from an estimate $G = \mathrm{Sobel}(f)$ of the magnitude of the gradient in each pixel. The gradient is estimated using averaged first order differences in the horizontal and vertical directions, which can be proved to be equivalent to using the derivatives of an appropriately weighted tilted-plane least squares approximation of the image in a 3×3 window around each pixel

$$\nabla f[i,j] = \begin{bmatrix} \frac{\delta f[i,j]}{\delta x} \\ \\ \frac{\delta f[i,j]}{\delta y} \end{bmatrix},$$

with

$$\begin{aligned}\frac{\delta f[i,j]}{\delta x} \approx \frac{1}{8a} f_x[i,j] = \frac{1}{8a}\big(& f[i-1,j+1] + 2f[i,j+1] + f[i+1,j+1] \\ & - f[i-1,j-1] - 2f[i,j-1] - f[i+1,j-1]\big)\end{aligned} \tag{4.1}$$

and

$$\begin{aligned}\frac{\delta f[i,j]}{\delta y} \approx \frac{1}{8b} f_y[i,j] = \frac{1}{8b}\big(& f[i-1,j-1] + 2f[i-1,j] + f[i-1,j+1] \\ & - f[i+1,j-1] - 2f[i+1,j] - f[i+1,j+1]\big),\end{aligned} \tag{4.2}$$

where a and b are the horizontal and vertical dimensions of the rectangular pixels, i.e., $\alpha = \frac{a}{b}$ is the pixel aspect ratio.

The magnitude of the gradient is estimated by

$$\|\nabla f[i,j]\| \approx \frac{1}{8a} G[i,j] = \frac{1}{8a}\sqrt{|f_x[i,j]|^2 + \alpha^2 |f_y[i,j]|^2}, \tag{4.3}$$

where the factor $\frac{1}{8a}$ is dropped from G because the resulting values will later be compared to a threshold, which may be adjusted accordingly. Often the pixel aspect ratio α is also dropped from the expression, since it is usually close to the unity.

Pixels for which $G[i,j] > t$, where t is a given threshold, are considered candidate physical edge pixels.

Edge localization is based on a simple thinning algorithm [100]: only candidate pixels which are local maxima in terms of the estimated gradient in the horizontal (i.e., $G[i,j] > G[i,j-1]$ and $G[i,j] > G[i,j+1]$) or vertical ($G[i,j] > G[i-1,j]$ and $G[i,j] > G[i+1,j]$) directions are considered physical edge pixels. Besides that, in order to avoid "minor edge lines in the vicinity of strong edge lines," the following additional constraints are imposed:

1. If $G[i,j]$ is a local maximum in the horizontal direction, but not in the vertical direction, $[i,j]$ is a physical edge pixel when:
$$|f_x[i,j]| > k|f_y[i,j]| \tag{4.4}$$
2. If $G[i,j]$ is a local maximum in the vertical direction, but not in the horizontal direction, $[i,j]$ is a physical edge pixel when:
$$|f_y[i,j]| > k|f_x[i,j]| \tag{4.5}$$

The value of k is usually set around 2.

An example of application of the Sobel operator to the first image of the "Carphone" sequence (see Appendix A) can be seen in Figure 4.1.

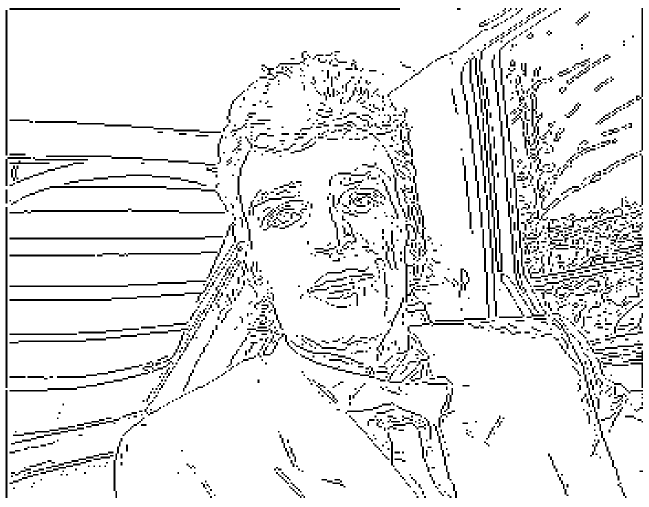


Figure 4.1: "Carphone": application of the Sobel operator, with thresholding and thinning, to the luma of the first image (using $t = 40$, $k = 2$, and $\alpha = 1$).

Laplacian of the Gaussian operator

Another common 2D scalar edge detection segmentation operator is the LoG operator [108, 100]. This name is usually given to any operator using the LoG for edge localization purposes. Some freedom remains about:

1. transition strength;
2. digitalization of the LoG; and
3. zero crossing detection method.

The operator herewith presented calculates the transition strength using the estimate of the gradient magnitude as given by (4.3). Thus, this operator has two different derivative computation methods: sampled derivatives of the two dimensional Gaussian function (for edge localization), and tilted plane approximation (for transition strength computation).

Pixels for which $G[i,j] > t$, where t is a given threshold, are considered candidate physical edge pixels.

Edge localization uses the zero crossings of the Laplacian (of the image smoothed by a Gaussian). A zero crossing is considered at each candidate physical edge pixel having positive second derivative and for which any of its 4-neighbors has negative second derivative. However, after calculating the LoG and before localization, pixels with second derivative inferior to a given threshold t_2 are set to zero.

The filter $w[\cdot]$ for the computation of LoG has a $2n+1 \times 2n+1$ region of support, where $n = \text{round}(4.5\sigma)$, and is calculated by

$$w[i,j] = \text{round}\left(K\left(2 - \frac{i^2+j^2}{\sigma^2}\right)\exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)\right) \quad \text{with } i,j = -n,\ldots,n,$$

where K is a scaling constant. It can be chosen to provide appropriate approximation or such that $\sum_{i,j=-n}^{n} w[i,j] = 0$. If the sum, for a given K, does not yield zero, the values of $w[\cdot]$ can be manipulated "by small amounts" [60].

The scale of the operator is given by σ, which is the standard deviation of the Gaussian. In [60], n is calculated so that all non-zero sampled and quantized coefficients of $w[\cdot]$ are included in the filter's region of support. The presented formula provides a decent approximation.

The result of applying this operator to the first image of the "Carphone" sequence can be seen in Figure 4.2.

3D operators

3D operators, which operate on more than one image, do not really aim at detecting physical edges. They can, however, help to detect regions that changed from one image to another in an image sequence, and the largest changes are usually located near the physical edges of scene objects. Thus, this information can be used to restrict the search area of more precise 2D edge detection operators applied afterwards. This can be useful when detecting the boundaries of moving objects over a static background [99].

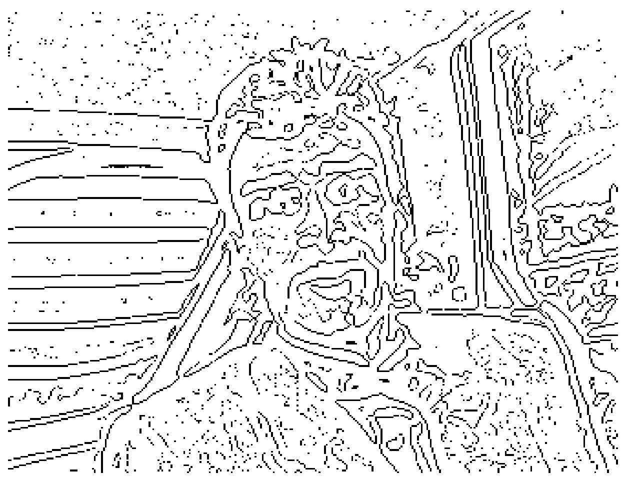


Figure 4.2: "Carphone": application of the LoG operator to the luma of the first image ($t = 10$, $\sigma = 2$ and hence $n = 9$, $K = 3208$ and $t_2 = 5000$).

Image differences

The simplest of the 3D operators is the image differences. Given two successive images f_n and f_{n-1}, it calculates the difference image D_n

$$D_n = \mathrm{Diff}[f_n, f_{n-1}]$$

such that

$$D_n[i,j] = \|f_n[i,j] - f_{n-1}[i,j]\|. \tag{4.6}$$

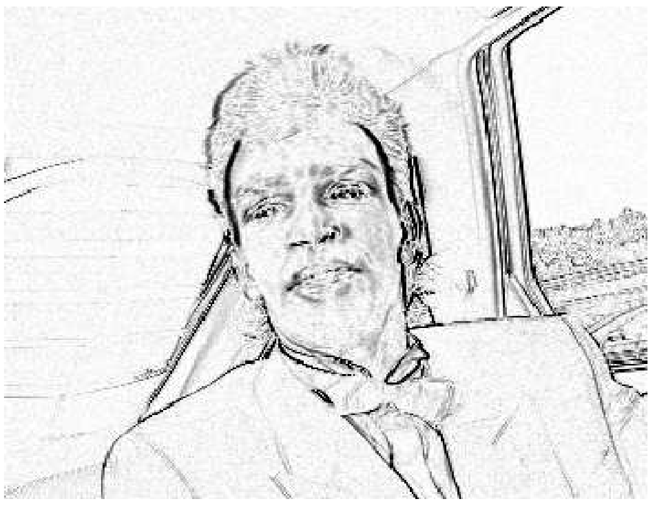
The difference operator is usually followed by some type of thresholding.

This operator is useful as a first step in an edge detection segmentation technique because it can be used to detect the zones that have changed significantly from one image to the next. This assumes that the objects of interest move in front of a static background. If the background also moves, then global motion, usually corresponding to camera movements, can be canceled out of f_{n-1} in order to stabilize the image before applying the difference operator. See Section 5.5 for a discussion of image stabilization methods.

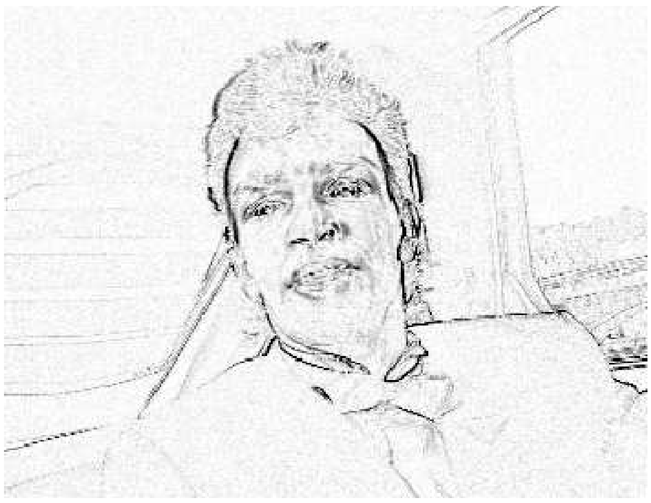
The result of the application of the image difference operator to the "Carphone" sequence, with and without image stabilization, can be seen in Figure 4.3.

Different motion

Another interesting scalar 3D operator was proposed by Cafforio and Rocca in [17]. This operator classifies each pixel of an image as belonging to one of "n moving areas with different displacements and a [fixed] background area" using "a Viterbi algorithm with $n+1$ states" [48].



(a) Without image stabilization.



(b) With image stabilization.

Figure 4.3: "Carphone": application of the image differences operator (without thresholding) to the luma of images 26 and 27 (differences multiplied by 5 and inverted for display purposes).

It can also be useful when trying to obtain the boundaries of moving objects in front of a fixed background.

4.2.2 Technique level

The middle level in a segmentation process is the technique level. Each segmentation technique may use several low-level segmentation operators and integrate their results into a coherent partition of the image. Topological considerations are used often at this level: boundary detection techniques, for example, can use edge detection operators followed by contour closing, clearing of isolated edge pixels, etc.

Segmentation techniques can be classified according to several attributes, which will be presented in the next sections.

Spatial primitives

The first attribute considered is the kind of primitives addressed by the technique. There are basically three types of segmentation techniques:

1. techniques aiming at boundary detection, i.e., techniques attempting to detect object or region boundaries from the places where there are sudden changes of illumination, texture, etc. (framework is contour-oriented segmentation);
2. techniques aiming at region detection, i.e., techniques detecting regions with uniform characteristics, such as intensity, color, texture, etc. (framework is region-oriented segmentation); and
3. techniques which aim at detecting both boundaries and regions (see for instance [65]).

Memory

Segmentation techniques may also be divided according to the use of memory:

1. techniques with memory use information from previous images; and
2. memoryless techniques do *not* use information from previous images in the image sequence.

Techniques with memory typically use 3D operators, i.e., operating on more than one image, possibly together with 2D operators. Memoryless techniques use only 2D operators.

Features used

If techniques with memory are being used and the previous image is available, then it is possible to estimate which parts of the image suffered different movements (see [17] for an early example). Strictly speaking, this is motion-based segmentation, and thus could also be classified as a tool towards time analysis of image sequences. This kind of techniques is associated with a different type of segmentation which was not mentioned in the introduction: segmentation into regions of uniform motion.

Thus, there are:

1. motion-based techniques, when segmentation is based on motion; and
2. color-based techniques, when segmentation is based on spatial features such as color or texture.

Vectorial vs. scalar

Segmentation techniques will be designated according to the number of color components they work with:

1. scalar techniques make use of a single color component, regardless of whether more are available; and
2. vectorial techniques use several color components.

Knowledge-based

Another feature of segmentation at technique level is the availability of a priori knowledge about the images to be segmented, that is, information about the image model that can be used:

1. if a priori knowledge is available, segmentation techniques are knowledge-based;
2. otherwise segmentation techniques are generic.

4.2.3 Algorithm level

The highest level in a segmentation process is the algorithm level. Segmentation algorithms integrate the results obtained by the lower level segmentation techniques (and possibly also operators) and attempt to assess and control the attained segmentation quality. There are several features distinguishing the different segmentation algorithms. A few are described in the next sections.

Segmentation quality

Segmentation quality is an important feature of segmentation algorithms. There are three main issues related to quality:

Quality estimation
How is the attained segmentation quality estimated?

Quality estimation objectives
What will the estimated segmentation quality be used for?

Quality control
How is the segmentation quality controlled?

Quality estimation

The estimation of the quality attained depends on the techniques used, and is still, to a certain extent, an unsolved issue, at least if done in an automatic way. Of course, it is possible to estimate the quality of a given segmentation if the structure of the image is known beforehand. This is what is done in the papers which attempt to evaluate segmentation algorithms, or even segmentation operators such as edge detectors: see [187, 8, 47]. This is not possible, however, when the correct or desired segmentation is not known before hand (otherwise why should one waste time reproducing a known result?).

Quality estimation objectives

The estimation of the segmentation quality can be used:

1. for changing the parameters of the segmentation so that a desired segmentation quality is attained, i.e., quality estimation for (feedback) control; or
2. for deciding whether the segmentation results should be accepted or rejected.

Quality control

The measure of segmentation quality may be used to adjust segmentation parameters (e.g., operator thresholds) in order to improve, through feedback, the segmentation quality of:

1. the next image, in the case of delayed segmentation quality control; or
2. the current image, in the case of immediate quality control.

Delayed segmentation quality control has a delayed response to changes in the sequence to segment. Hence, it can only be applied if these changes are expected to be slow. On the other hand, immediate segmentation quality control can only be used if the segmentation algorithm being used is not too computationally demanding.

Scale and resolution

One of the features of segmentation algorithms considered is the scale or scales at which they operate. That is, the scale at which differences in the properties that characterize the segmented regions are detected. A small scale segmentation algorithm will be able to detect and segment detailed regions, e.g., the details of a textured surface, while a large scale segmentation algorithm will be able to detect only less detailed changes in the properties of the regions.

The scale of an algorithm is usually controlled by adjusting scale parameters in the low-level operators used. For instance σ in the LoG operator of Marr and Hildreth [108]. An algorithm may use techniques at different scales and integrate them into a many-scale description of the scene [18]. Jeong and Kim [85] proposed a method for adaptively selecting the scale along the image.

Resolution is another feature of segmentation algorithms. It is related to the resolution of the segmentation of the image. The resolution usually depends on the application at hand. Segmentation can be done at pixel resolution or, for instance, at MB (MacroBlock) resolution (16×16 pixels).

Even though segmentation scale and resolution are related, there are a few differences between the two, the most important being that scale is concerned with the level of detail taken into account *during* the segmentation while resolution is concerned with the level of detail necessary *after* the segmentation. The difference can be made clear by means of an example. Suppose that segmentation is to be used in a H.261 encoder merely by changing the quantization step (which is fixed for each MB). Then, a MB resolution for the segmentation is clearly enough. However, in order to clearly determine the speaker's position against the background, a segmentation scale much smaller than MB will obviously be needed (see Section 4.4).

Memory

A segmentation algorithm may also be classified according to its use of the temporal information between adjacent images in an image sequence. This use can be done at the technique or even operator level, for instance by using 3D operators, or only at algorithm level, for instance by restricting the search of the edges of physical objects to a small region around their previous positions, possibly by using motion information extracted from the image sequence. Other uses of memory will be seen in Section 4.7.1, where a region-oriented segmentation algorithm making use of memory is proposed. The segmentation algorithm can thus be:

1. memoryless if temporal information is *not* used; and
2. with memory if temporal information is used.

Knowledge-based

Segmentation at algorithm level, as happened at technique level, can be classified according to the availability of knowledge:

1. if a priori knowledge is available, segmentation algorithms are knowledge-based;
2. otherwise segmentation algorithms are generic.

4.2.4 Conclusions

In summary, the various levels in the segmentation process can be classified according to:

Operator level

1. Spatial features detected (edge or region detection or both).
2. Number of components used (vectorial or scalar).
3. Number of images operated on (2D or 3D).

Technique level

1. Segmentation operators used.
2. Spatial features detected (boundary or region detection or both).
3. Use of temporal information (with memory or memoryless).
4. Features used (motion or color).
5. Number of components used (vectorial or scalar).
6. Use of a priori information (knowledge-based or generic).

Algorithm level

1. Segmentation techniques (and eventually segmentation operators) used.
2. Quality estimation objectives (control or acceptance/rejection decision).
3. Quality estimation method.
4. Type of quality control (immediate or delayed or none).
5. Scale of the segmented features.
6. Resolution of the resulting classification.
7. Use of temporal redundancy (with memory or memoryless).
8. Use of a priori knowledge (knowledge-based or generic).

A diagram with the proposed segmentation process hierarchy and the classification criteria for each of its levels is show in Figure 4.4.

4.2.5 Pre-processing

Pre-processing can be an important step in a video encoder, where it is often seen as a part of image analysis. It can occur in two different positions:

1. before analysis proper, pre-processing can be used used to emphasize important image features and to eliminate details which are irrelevant to the subsequent analysis; and

2. before encoding (after analysis[2]), pre-processing can be used to change the input sequence's characteristics, according to the analysis results, so that the coding can be made more efficient (e.g., in the framework of videotelephony, by low-pass filtering the background of the images after having detected the speaker's position through knowledge-based segmentation, or by manipulating coding parameters such as the DCT quantization step, in the case of classic codecs).

[2]It is actually post-processing relative to analysis.

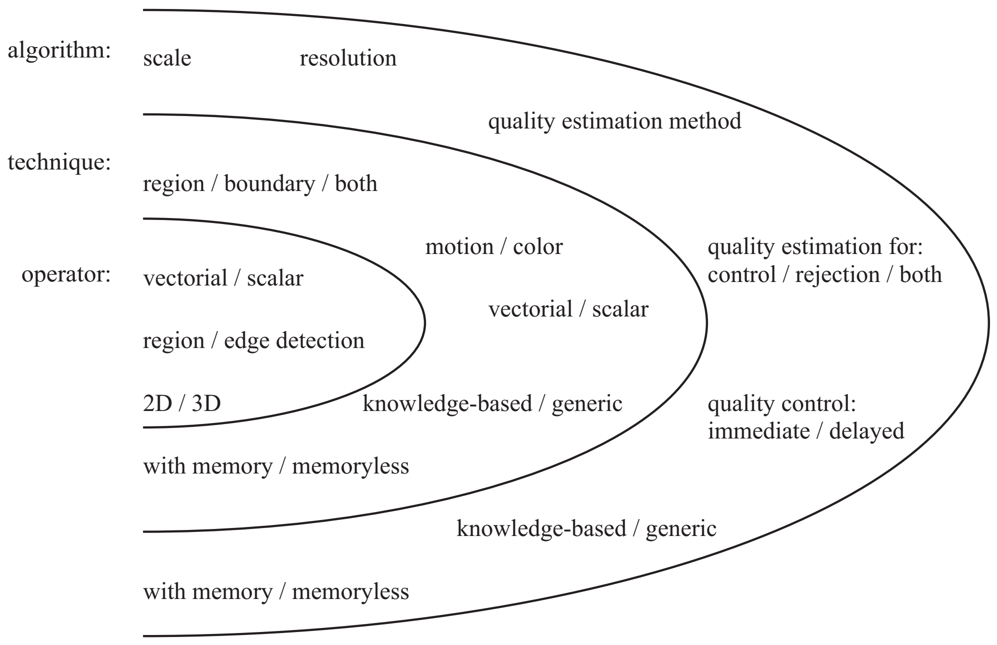


Figure 4.4: Segmentation process hierarchy and classification for operator, technique, and algorithm level.

4.3 Region- and contour-oriented segmentation algorithms

This section overviews several well-known segmentation algorithms and attempts to frame them within the common theory of SSTs. First, the basic versions of region merging and region growing algorithms are shown to be really algorithms solving different graph theoretical problems, all involving SSTs. Then, it will be shown that the distinction between region- and contour-oriented segmentation is not as clear-cut as it may seem at first: the basic contour-closing algorithm is shown to be the dual of the basic region merging algorithm, both being describable again recurring to SSTs. Finally, the problem of globalization of the information along the segmentation process is introduced, along with the problem of choosing appropriate homogeneity criteria for the regions, i.e., the problem of choosing an appropriate region model. It will also be shown that, in this framework, segmentation algorithms can be seen as non-optimal algorithms which attempt to minimize a cost functional, typically related to an approximation error. Globalization methods, region models, and cost functional, are what really distinguishes all the algorithms describable in the SST framework.

4.3.1 Contour-oriented segmentation

As said before, two different approaches can be used for segmentation. The first approach aims at identifying transitions in the image features which are relevant to the task at hand,

and which may be transitions in gray level, color, texture, motion, etc. Most of the available transition detection methods were developed as first steps toward identification of object edges in the sensed 3D natural world of which the image is a projection (e.g., [108]). Hence, the term edge detection, used in all the literature and throughout this thesis, stuck in connection to the low-level transition detection operators.

Contour-based segmentation algorithms typically include edge detection at operator level. Most edge detection operators classify pixels or edges as either deemed to belong to a physical object edge or not, and this decision is made by looking at a small neighborhood of the given pixel or edge, calculating a few parameters, and comparing them to thresholds. Even though some thresholding methods introduce, to a certain extent, more global information, e.g., the hysteresis thresholding for edge localization proposed by Canny [18], edges are mostly detected in a rather local form, and hence do not usually form closed boundaries.

A further problem with edge detection operators is that they often require the tuning of a number of parameters. A typical parameter is the transition strength threshold (or thresholds, in the case of hysteresis), which if set too high leads to edges far from constituting closed contours, and if set to low leads to many erroneously detected edges. The selection of the operator parameters is thus not trivial. Some solutions have been proposed in the past for automating parameter selection [85, 151].

Contour closing

The result of edge detection is a mapping of pixels or edges into one of two classes: elements corresponding to large transitions and elements corresponding to smooth, uniform zones (in terms of the image features of interest). Clearly, such approaches do not directly lead to a partition of the image—the detected edges may not form closed boundaries. Hence, edge detection operators are usually followed by contour closing techniques and then by algorithms which classify as different regions the connected components separated by the estimated contours.

Assuming that the transitions are detected at edges (boundaries of the pixels), perhaps the conceptually simplest method of obtaining closed contours is to apply a threshold to the result of some transition strength component, to build the subgraph of the image line graph induced by the detected edges, and finally to remove all the bridges in this subgraph, thus obtaining a 2-arc-connected graph, which indeed segments the image into several regions. If the transition strength threshold is decreased successively, so that edges are detected with non-increasing strength, then a succession of partitions of the image can be obtained, ranging from a single region covering the whole image, to a region per pixel, when all the edges are detected. This algorithm will be called the basic contour closing whenever the transition strength of an edge is calculated simply as the distance between the colors of the two pixels it bounds.

4.3.2 Region-oriented segmentation

The second approach to segmentation attempts to deal with homogeneity instead of dissimilarities (i.e., transitions). Such methods usually lead directly to a partition of the image, and hence the aforementioned division of the segmentation process into operator, technique, and algorithm level is not as clear as for contour-oriented segmentation.

Two of the most referenced segmentation methods in the literature [154, 68] are region growing and split & merge. Other more recent contenders in this field are watershed segmentation, based on mathematical morphology theory, and SST segmentation. These segmentation algorithms, all region-oriented, will be briefly overviewed in the following sections, and their basic versions will be described and compared. But before that, region segmentation will be defined formally.

A formal definition of segmentation

Segmentation can be seen as an optimization problem. Assuming a uniformity measure has been established for each class in a partition and for the partition as a whole, the optimal partition can be defined as that which:

1. for a given maximum number of classes, maximizes the overall uniformity; or
2. for a given minimum overall uniformity, minimizes the number of classes.

This definition of segmentation lacks a very important concept: the spatial relation of the pixels. Granted, such concerns may be partially embedded into the uniformity measure, but it would be useful if they could be made more explicit. Without such spatial concepts, all permutations of the pixels values in a given image lead to essentially the same segmentation: a segmentation which is based solely on the pixel colors. Without spatial relationships taken into account (i.e., using only the measurement space [68]) segmentation is essentially a clustering problem.

A more restrictive and interesting definition uses connected classes instead of possibly disconnected classes. The optimal partition can thus be defined as that which:

1. for a given maximum number of connected classes, maximizes the overall uniformity; or
2. for a given minimum overall uniformity, minimizes the number of connected classes.

With this definition, by taking into account also the spatial relationships, segmentation can be seen as clustering in both spatial and measurement space [68]. It is also quite an intractable problem because the solution space is huge. For an image with 100×100 pixels, and assuming a partition into 4 disconnected classes, the total number of possible partitions to consider is $4^{10000} \approx 10^{6021}$. When connected classes are required, the solution space is considerably smaller, but still too large to consider a brute force search for the optimum. Thus, most segmentation algorithms are actually non-optimal solutions of the stated problem.

Considerable latitude exists in the choosing of appropriate uniformity measures for each class and for the whole partition. While some may be so simple as the range of grey levels inside a given class, others may recur to more or less sophisticated models for the color variation inside each class, an thus to an uniformity measure which is essentially the inverse of the modeling error. Both the problems of non-optimal approximations to the optimal segmentation and of region modeling will be discussed at more length in Section 4.3.4.

Region growing

Region growing algorithms start with an initial set of seeds or markers (small sets of pixels, possibly disconnected) to which adjacent pixels are successively merged if this merging leads to an homogeneous region according to some criterion. Regions stemming from different seeds are never merged. The process is complete when every pixel is assigned to one of the regions and there are no pairs of adjacent regions stemming from the same seed. Strictly speaking, region growing does not attempt to solve the segmentation problem. The problem solved is actually a restriction of the original problem, where pixels from different seeds are not allowed to belong to the same class. Notice that the number of seeds is a lower bound to the number of classes, but the two are not always equal: a seed may lead to several non-adjacent classes.

The use of seeds is both a curse and a blessing. On the one hand, such algorithms by themselves are unable to perform automatic segmentation. However, several methods have been proposed in the literature to automatically identify appropriate seeds, especially in the case of watershed segmentation, a particular breed of region growing segmentation algorithm which will be discussed below. On the other hand, region growing segmentation algorithms lend themselves very easily to supervised segmentation, in which a (human or not) supervisor classifies some pixels of the image as belonging to different classes, and then the segmentation algorithm attempts to honor these hints.

The basic version of the region growing algorithm starts by labeling the pixels in each seed with a label which is unique for that seed. All other pixels are initially unlabeled. Then, of all the unlabeled pixels which are adjacent to at least one labeled pixel, the one with the smallest color distance to an adjacent labeled pixel is labeled with the label of that pixel. When all pixels are labeled, the labels represent the partition of the original image. Hence, the final partition has as many classes as there are seeds, some of which may have more than one region.

Watershed segmentation

Watershed segmentation has its roots in a topographic problem: given a digitized topographic surface, how can draining basins be identified? Or, by duality, where are the watersheds of the basins located? The solution to this problem involves identifying local minima, piercing these minima, and slowly immersing the topographic surface in some (virtual) liquid. Whenever liquid flowing from different sources is about to mix, a dam is built. After total immersion the dams identify the watersheds of the basins, and each basin corresponds to a local minimum. This process is very nicely described in [132]. This method of identifying basins in a topographic surface can be seen as a form of segmentation. The problem with the method is that it leads to over segmentation. This stems from the fact that each local minimum is considered to give rise to an individual basin, no matter how small. The standard solution to this problem involves piercing the topographic surface at those locations deemed to represent individual basins and apply the algorithm without further changes. Since the number of liquid origins is reduced, so is the final number of basins. In a sense, as was recognized in [132], such solution merely shifts the problem: how to select where to pierce the topographic surface? However, the analogy between watershed segmentation and region growing segmentation is immediate, and the same comments apply as in the case of region growing: watershed segmentation may be a good technique for inclusion in some complete algorithm which either (a) decides by itself or (b) asks some human supervisor where to pierce the surface (where to locate the seeds, in the case of region growing).

This method of identification of basins in digitized topographical surfaces was soon recognized to have a high potential in image segmentation [132]. However, two problems had to be solved: (a) typical images have values in $\mathbb{Z}^3$, so that the analogy with heights of terrains is not immediate, and (b) even in the case of grey scale images, taking values in $\mathbb{Z}$, the basins of the grey levels taken artificially as heights of some imaginary terrain are hardly what image segmentation aims at. This latter problem was solved by recognizing that, if segmentation aims at detecting reasonably uniform regions, then watersheds should be located in pixels where the gradient is high. Hence, the watershed segmentation started to be applied to the absolute value of the image gradient. The gradient, in the original papers [132] and [193] was usually calculated recurring to morphological filters. Estimating derivatives, however, is an ill posed problem, hence other solutions working directly on the original image were necessary.

The above problems are addressed in [131] (see also [177]), which presents a generic watershed segmentation algorithm of which the already described (classical) watershed segmentation algorithm, working on a topographical surface, and the basic region growing algorithm are particular cases. This paper also discusses briefly the problem of selecting an appropriate color distance, which is closely related to the problem of selecting a color space. Its proposal is to select the HSV (Hue, Saturation, and Value) color space. However, see discussion in Section 3.1.1.

Actually, the region growing version of the generic watershed segmentation algorithm is not exactly the basic region growing algorithm as described in the previous section. The region growing version of watershed segmentation does take into account that, when at a certain step of the algorithm there is a tie, that is, several different pixels may be aggregated to different regions, there are solutions which are better than others. In analogy with the classical watershed segmentation, which tried to flood plateaus with liquid flowing at a constant speed from each source, thus locating the watersheds in their "natural" location in the middle of the plateaus, the region growing version of watershed segmentation solves the problem in a similar way: in case of a tie, choose the oldest candidate pixel for merging. If, as will be seen shortly, basic region growing can be implemented using the simple extension of Prim's constructive algorithm for SSSSkT, then the region growing version of watersheds, with its nice treatment of ties, can be implemented by the same algorithm with a further restriction: the pixel queue must be not only hierarchical but also ordered: pixels in the same hierarchical level should be organized in a queue (first came first served). In the particular case of digital images where the image values take only a relatively small set of values (which actually is the case in most situations, since usually 8 and at most 10 bits are used to encode the color components of each pixel), very efficient algorithms can be developed [193].

There are a few reasons why ties should not worry us too much, though. First, it is probable that in the future more and more bits are used to represent images in intermediate steps of processing. As a consequence, when some sort of filtering is performed on images before segmentation, the likelihood of ties decreases and the effects of handling them blindly will probably not be severe. But the best of all reasons is that it will allow us to nicely compare several different segmentation algorithms under the common framework of SSTs.

Region merging

Region merging algorithms, unlike region growing algorithms, do not recur to seeds. A partition of an image is input to the algorithm, typically the trivial partition where each pixel is a single class, and at each step of the algorithm pairs of adjacent regions are examined and merged into one if the result is deemed homogeneous. This version of the algorithm is essentially the RAG-MERGE of [154]. This algorithm can be improved if the pair of regions to be merged at each step is selected as the one leading to the greater uniformity. In this case, the algorithm is called RSST [134], for reasons which will be shown later.

The basic version of the region merging algorithm merges pairs of regions according to the color distance between adjacent pixels each belonging to each of the two adjacent regions candidate for merging. Needless to say, there can be several such pairs of pixels for a given pair of adjacent regions. The smallest color distance of all pairs of pixels in the above conditions is taken as representative of the uniformity of the union of the two regions. Granted, this algorithm is clearly poorer than RSST and RAG-MERGE above. Its interest will be seen later, when discussing methods of globalizing the decisions taken in the basic algorithms that lead to the mentioned RSST and RAG-MERGE algorithms.

Split & merge

In 1976, Horowitz and Pavlidis [72] developed an image segmentation algorithm combining two methods used independently until then: region splitting and region merging. In the first phase, region splitting,[3] the image is initially analyzed as a single region and, if considered non-homogeneous according to some criterion, it is split into four rectangular regions. This algorithm is recursively applied to each of the resulting regions, until the homogeneity criterion is fulfilled or until regions are reduced to a single pixel. At the end of the split phase, the regions correspond to the leaves of a QPT (Quartic Picture Tree).[4] If split were the only phase of the segmentation algorithm, the segmented image would have many false boundaries, since splitting is done according to a rather arbitrary structure, the quad tree. The second phase of the algorithm is region merging,[5] where pairs of adjacent regions are analyzed and merged if their union satisfies the homogeneity criterion.

Several problems may occur in split & merge algorithms, namely artificial or badly located region boundaries. These problems usually stem from the split criterion used, which is thus determinant for the final segmentation quality.

In 1990, Pavlidis and Liow [158] presented a method that uses edge detection techniques to solve the typical split & merge problems (e.g., boundaries that do not correspond to edges and there are no edges nearby; boundaries that correspond to edges but do not coincide with them; edges with no boundaries near them). The method is applied to the over-segmented image resulting

[3]Horowitz and Pavlidis actually define the first phase as the split & merge phase and the second as the grouping phase. However, the first one can be simplified (though this can make it less computationally efficient) to a simple split if one starts by considering the entire image (level 0).

[4]This acronym is the one used in [154]. QPTs are also know as quad trees.

[5]The grouping phase in [72] and RAG-MERGE in [154].

from the split & merge algorithm. It is based on [158]: "criteria that integrate contrast with boundary smoothness, variation of the image gradient along the boundary, and a criterion that penalizes for the presence of artifacts reflecting the data structure used during segmentation." Some ideas along these lines will be discussed later.

One of the main bottlenecks in typical segmentation algorithms is memory, even more so than computation time. The data structures representing the images, and the associated graphs, which algorithms typically use, can easily require hundreds of megabytes. The memory usage grows with the initial number of regions considered, particularly in the case of region merging. Split & merge can be seen as a good method for trading a reduction of memory usage for an increased computation time, since after splitting the number of regions is typically smaller than the number of pixels and splitting can be a time consuming task. Incidentally, this is the reason why in [72] there is a merging phase within the quad tree structure, which allows them to start the process at a lower level in the tree. But there are other reasons which may lead to the splitting process. If the homogeneity is based on how well a region model conforms to the actual color variations along the union of two adjacent regions, and if this model is complex, estimating its parameters for small regions tends to be an ill-defined problem. Since estimation for small regions can be very sensitive to noise, a tradeoff is thus required between noise immunity and accuracy in the location of region boundaries. This tradeoff is typical of the "uncertainty principle of image processing" [199].

4.3.3 SSTs as a framework of segmentation algorithms

The first attempt to describe several segmentation algorithms within the common framework of SSTs was made by Morris *et al.* [134]. Region merging and edge detection were both put into the SSTs framework, even though the description of edge detection with SSTs was not complete. This section will elaborate on the results of [134], by describing region merging, region growing, and contour closing, all within the framework of SSTs. Notice that, even though the results in Chapter 3 are usually given for graphs in general, which may be disconnected, in this chapter our attention is concentrated in typical images, whose image graphs are connected. Hence, SSTs are used instead of SSFs.

Region growing as a solution to the SSSSkT problem

Consider the basic region growing algorithm, as described before. If all seeds are restricted to no more than one pixel, then this algorithm is exactly the constructive extended Prim algorithm for solving the SSSSkT problem. What happens when seeds are allowed to have more than one pixel? For the sake of clarity, each seed pixel will be considered to have a label identifying uniquely the pixels in the corresponding seed: seed pixels of the same seed have the same label while seed pixels from different seeds have different labels. In this case, it can be proved that the extended Prim algorithm results in a SSSSkT which is a subgraph of the required (say) multi seeded k-tree. Some separators of the SSSSkT do not connect trees with seed pixels of different labels, and thus may be part of the solution. If the branches of the SSSSkT are contracted in the original graph and the true separators (separators of seed pixels with different labels) are removed, then the branches of any SSF of the resulting graph can be added to the SSSSkT to obtain a solution to the multi seeded problem. It can also be proved easily that a solution can be obtained if Prim's algorithm is changed so as to allow insertion of branches connecting trees corresponding to seed pixels with the same label. That is, by changing the definition of separator and connector to account for the possible existence of multiple pixels in a seed.

An immediate conclusion of the preceding lines is that region growing is indeed finding a solution to the (multi seeded) SSSSkT problem, not a solution to the segmentation problem. The main reason for this is that the decision about whether or not a pixel should be merged to one of the growing regions is based solely on the difference between that pixel and another pixel in the region, not the complete region. This means that through small changes at each region growth, the regions can turn out to be far from uniform. This problem can be addressed by globalization methods, which will be addressed later.

Destructive algorithms may also be used to obtain the desired segmentation. Use any algorithm to obtain the SST of the image and then cut successively the heaviest branches in the tree standing between seeds of different markers. Or, which is the same, apply Kruskal's extension to solve the (multi seeded) SSSSkT problem over the SST. The advantage of this last algorithm only comes about when multiple segmentations with different markers have to be performed over the same image, as for instance in supervised segmentation. Calculation of the SST is $O(\#\mathbf{V} \lg \#\mathbf{V})$ for planar graphs, but it is done only once. On the other hand, solving the SSSSkT problem over the SST runs in linear time. Hence, when the number of segmentations of an image grows, the amortized computation time of each segmentation tends to linearity on the number of pixels.

The watershed algorithm can be seen as a special case of the region growing algorithm where the pixel queues are not only hierarchical but also ordered. This changes the way the algorithm works in cases of ties, as will be discussed later. It does not change the asymptotic running time of the algorithms. What does change it however, is if the hierarchical queues are hierarchized based on a weight which can only take a small number of values. In this case, as was recognized in [193] and [131], faster hierarchical (and ordered) queues can be devised, taking the form of arrays of queues, one queue for each possible distinct weight.

Region merging as a solution to the SSkT problem

The basic version of the region merging algorithms is immediately recognizable as the Kruskal algorithm for finding a shortest spanning k-tree of a graph, in this case the image graph. This was realized by [134], which labeled this type of segmentation SST segmentation. Each of the k components of the attained k-tree is hence a region of the partition obtained by the algorithm. Two interesting conclusions may be drawn out of this fact.

Firstly, this tells us that region merging is indeed finding a solution to the SSkT problem, not a solution to the segmentation problem. The main reason for this is that decision about whether or not two regions should be merged together is based solely on the difference between two pixels of the two different regions, not on the complete regions. This means that two regions which have totally different global properties can be united by a narrow strip of slowly varying pixels. This problem can be addressed by using globalization methods, to be discussed later.

Secondly, it is now evident that destructive algorithms may also be used to obtain the desired segmentation. Use any algorithm to obtain the SST of the image and then cut the heaviest $k-1$ branches in the tree. Unlike the destructive algorithms for region growing mentioned in the previous section, cuts are now done irrespective of the position of the branches within the tree.

Region merging with seeds

It is possible to extend region merging so as to use seeds. In this case, the algorithm simply prevents regions with different seeds from being merged. This can also be seen easily to be the constructive algorithm for $SSSSkT$ based on the Kruskal algorithm. As before, the definitions of separator and connector may have to be adjusted to account for the existence of seeds with more than one pixel. But this algorithm is solving the SSSSkT problem, which was shown in the previous section to be also solved by the basic version of the region growing algorithm. Hence, region merging with seeds solves the same problem as region growing: the SSSSkT problem. The region growing algorithm follows Prim's approach while the region merging approach follows Kruskal's. Notice that the result of both algorithms, in terms of the attained k-tree, is guaranteed to be the same only if there is a single solution to the SSSSkT problem. However, the results may be equal in terms of the identified regions even if the k-trees attained are different.

By using hierarchical ordered queues, Prim's approach to the SSSSkT can deal with ties (the so-called plateaus in the watershed terminology) in a structured way. This is much harder, if at all possible, with Kruskal's approach. However, Kruskal's approach is such that at each step of the algorithm the already selected arcs form a SSSkT of the graph. Hence, the algorithm may be stopped before all seedless regions have been removed. This gives some autonomy to the algorithm, since it may decide that some seedless regions are to be treated as independent regions. The Prim's approach does not allow such regions to form, at least when using constructive algorithms. When using destructive algorithms both approaches are appropriate.

Region merging and contour closing as duals

Contour closing operates on the line graph, the dual of the image graph. The arcs of the line graph are inserted into a tentative edge contour graph in non-increasing weight order. At each step of the algorithm the edge contour graph can be obtained from the tentative graph by eliminating all bridges, thus leaving a 2-arc-connected planar graph which separates the image into several connected regions.

Suppose that the previous algorithm is modified slightly: each time an arc is to be inserted into the tentative graph, it is first checked whether it would introduce any circuits; if it would, it is put into a queue an left out of the tentative graph. After all arcs having been considered, the arcs which are in the queue are inserted one by one into the tentative graph. If the insertion schedule for arcs in case of ties is the same, both algorithms yield exactly the same result. This can be proved easily. First, observe that the first part of the modified algorithm is actually the Kruskal algorithm for the LST. Hence, after consideration of all arcs, the arcs in the tentative edge contour graph are the branches of a LST of the line graph, and the arcs in the queue are its chords. What is the constitution of the tentative edge contour graph after insertion of the ith chord, say c_i? It contains all branches of the LST and chords c which preceded c_i in the insertion schedule, and hence are not lighter than c_i. Each such chord c as a corresponding fundamental circuit containing no further chords, such that all its branches b preceded c in the insertion schedule, and hence are not lighter than c. Hence, it is clear that all circuit arcs in the tentative edge contour graph preceded c_i in the insertion schedule or, which is the same, all arcs following c_i in the insertion schedule are either bridges of the tentative edge contour graph, or chords which still haven't been inserted. Removing such bridges leaves us with the same tentative edge contour graph as after insertion of c_i using the first algorithm (and the same insertion schedule), so that the two are effectively equivalent.

It has been proved, thus, that the basic contour closing algorithm is in fact the Kruskal algorithm for finding the LST of the line graph followed by successive insertion of chords into the tree. Each such chord introduces a circuit. Since the emphasis here is on planar graphs, each such insertion creates a further face in the graph. But the dual spanning tree of the LST is a SST of the image graph. The insertion of chords of non-increasing weight into the LST of the line graph corresponds thus to the removal of non-increasing arcs from the SST. For each such operation, a face is split in two in the line graph and the corresponding connected component is divided in two in its dual, the image graph. Hence, the modified contour closing algorithm corresponds in the dual graph to the destructive algorithm for obtaining a SSkT from the SST, that is, it is the region merging algorithm. Region merging and contour closing are thus two algorithms which solve the same problem. It is a case where the duality between region- and contour-oriented segmentation is an actual fact.

The first attempt to formalize this interesting fact was made in [134], but the authors failed to recognize that arcs should be *inserted* into the LST, thereby creating circuits, and not removed, which is a pointless operation to perform on the LST of the line graph, even though the right one in the SST of the image graph.

It should be noticed, however, that globalization makes region merging and contour closing diverge, i.e., produce different results, as will be discussed later.

The problem of ties or plateaus

A few notes are in order regarding the problem of ties mentioned before. Knowing that the basic algorithms can all be described in terms of SSTs, it should be clear that, when multiple equivalent solutions exist, this is related to the fact that there are usually no unique solutions to the SSF, SSkT, SSSkT, or SSSSkT problems. However, two different spanning k-trees of an image graph can lead to the same partition, since two regions are equal even if covered by different trees. The issue of multiple solutions, its relation with the multiple solutions of the spanning trees problems, and its relation with mathematical morphology (through watersheds), remained as an issue for future work.

4.3.4 Globalization strategies

The basic versions of the region growing, region merging, and contour closing algorithms all make decisions about when to merge or split two regions using local information, namely the color difference between pairs of pixels. Information is globalized in those algorithms only insofar as they discard arcs between pixels already at the same region of the evolving partition.

Regions of considerable size can thus be merged, namely in the case of region merging, just because they happen to have two adjacent pixels which are similar, even if the regions themselves are quite different globally. This is a problem of scale: as the size of the regions increases, the scale at which their are considered should also increase. But it is also a problem of noise immunity: if whole regions are taken into account, the noise tends to be "averaged out", thus rendering the algorithms more robust. There is thus the need to globalize the information over which decisions are made. Another way of seeing this problem is to recognize that the use of local information leads the algorithms away from the optimum segmentation.

It is through globalization that the algorithms really tend to diverge and to gain new interesting properties. In the previous sections it was shown that region merging and contour closing were really solving the same problem, they were, as a matter of fact, dual algorithms. It was also shown that region merging with seeds and region growing also solve the same problem. This only happens in the case of the basic algorithms. When globalization is enforced, by establishing region and/or boundary models, for instance, the algorithms gain individuality. The next sections will overview the issues of modeling and globalization and discuss briefly their influence on the basic algorithms.

As the different basic algorithms are globalized, they no longer solve the same problem, which might be to find a SSkT, a SSSkT, or a SSSSkT. However, they do attempt to achieve a segmentation which is closer to the optimal segmentation. Hence, most of the globalization methods are actually heuristics towards solving the intractable problem of optimal segmentation.

Region modeling

As defined, segmentation searches for regions which are uniform according to certain criteria. In the past, several such criteria have been used, such as considering a region uniform when the dynamic range or the variance of its gray level is small, or when the maximum distance between colors in the region is also small. These are, in a sense, statistical criteria. Another, more interesting, class of homogeneity criteria states that a region is uniform if the error between the actual pixel values and a model for the region, with estimated parameters, is small. Segmentation into regions which provide the best possible approximation to an image, using a given region model, is actually the same as fitting a facet model to the images [68]. In facet models, each image component is thought to consist of a piecewise continuous surface, which can be constant (the flat facet model), linear or affine (sloped facet model), quadratic, cubic, etc. It is also possible to envisage the use of texture models, for instance.

By far the most commonly used model in segmentation is the flat region model. Most of the algorithms described in the literature (see next sections), use this simple model. It is the case of the watershed and RSST algorithms, and it is also the case of some of the algorithms proposed in this thesis.

As to the error calculation, it is typical to use the root mean square error (related to the Euclidean distance) as the value which should be minimized, since it both averages the error along the region and has nice algebraic properties: the estimates of parameters of linear models are obtained through statistics such as the mean and the variance. The maximum absolute error, on the other hand, worries too much about the deviation of a single pixel, while the sum of absolute errors has algebraic properties which are less amenable to efficient implementation, since the estimated values are obtained through statistics such as the median, which is harder to calculate than the mean value.

As to the distance between colors, even though some authors [112] use the maximum absolute component difference of the vectorial difference between $R'G'B'$ or HSV color spaces and some others suggest CIE $L^*a^*b^*$ and $L^*u^*v^*$ color spaces because of their improved perceptual uniformity [194], the most commonly used metric is the Euclidean distance in the $R'G'B'$ space, which generally leads to reasonable results (see [164]).

The next sections derive the equations for the flat and affine region models and show how the approximation parameters for the union of two regions can be obtained from a reduced set of statistics for each of the individual regions.

The flat region model equations

Let $\mathbf{R}$ be a region in the domain of a digital image f. The flat region model states that $\hat{f}$, the approximation of f, is $\hat{f}[v] = a\ \forall v \in \mathbf{R}$, i.e., the approximate image color is constant inside that region. Let $e(f, \hat{f}, \mathbf{R})$ be the approximation error between f and $\hat{f}$ inside $\mathbf{R}$. Then

$$e(f, \hat{f}, \mathbf{R}) = \sqrt{\frac{\sum_{v\in\mathbf{R}} d^2(f[v], a)}{\#\mathbf{R}}}$$

with the distance

$$d(x, y) = \|x - y\| = \sqrt{(x-y)^T(x-y)} = \sqrt{\sum_{l=0}^{n-1} |x_l - y_l|^2} \tag{4.7}$$

where n is the number of color components of the image.

Differentiating the error relative to the color vector a, the minimum of the error is

$$e(f, \tilde{f}, \mathbf{R}) = \sqrt{\frac{B(f, \mathbf{R}) - \#\mathbf{R}\tilde{a}^T\tilde{a}}{\#\mathbf{R}}} \tag{4.8}$$

where $B(f, \mathbf{R}) = \sum_{v\in\mathbf{R}} f[v]^T f[v]$, and it is obtained for

$$\tilde{f}[v] = \tilde{a} = \frac{A(f, \mathbf{R})}{\#\mathbf{R}},$$

where $A(f, \mathbf{R}) = \sum_{v \in \mathbf{R}} f[v]$.

Suppose now that two disjoint regions $\mathbf{R}_j$ and $\mathbf{R}_k$ are to be merged into a single region $\mathbf{R}$, i.e., $\mathbf{R} = \mathbf{R}_j \cup \mathbf{R}_k$.

Before merging, the image is approximated by

$$\tilde{f}[v] = \begin{cases} \tilde{a}_j = \frac{A(f,\mathbf{R}_j)}{\#\mathbf{R}_j} & \text{if } v \in \mathbf{R}_j \text{, and} \\ \tilde{a}_k = \frac{A(f,\mathbf{R}_k)}{\#\mathbf{R}_k} & \text{if } v \in \mathbf{R}_k; \end{cases}$$

and the total approximation error before merging is

$$E = \sqrt{\frac{\sum_{l=0}^{r-1} \#\mathbf{R}_l e^2(f, \tilde{f}, \mathbf{R}_l)}{\sum_{l=0}^{r-1} \#\mathbf{R}_l}}$$

where r is the total number of regions.

After merging, the total error is

$$E' = \sqrt{\frac{\#\mathbf{R}e^2(f, \tilde{f}', \mathbf{R}) - \#\mathbf{R}_j e^2(f, \tilde{f}, \mathbf{R}_j) - \#\mathbf{R}_k e^2(f, \tilde{f}, \mathbf{R}_k) + \sum_{l=0}^{r-1} \#\mathbf{R}_l e^2(f, \tilde{f}, \mathbf{R}_l)}{\sum_{l=0}^{r-1} \#\mathbf{R}_l}}$$

and the image is approximated, inside $\mathbf{R}$, by

$$\tilde{f}'[v] = \tilde{a} = \frac{A(f, \mathbf{R})}{\#\mathbf{R}}.$$

Since

$$\begin{aligned} A(f,\mathbf{R}) &= A(f,\mathbf{R}_j) + A(f,\mathbf{R}_k), \\ B(f,\mathbf{R}) &= B(f,\mathbf{R}_j) + B(f,\mathbf{R}_k), \text{ and} \\ \#\mathbf{R} &= \#\mathbf{R}_j + \#\mathbf{R}_k, \end{aligned} \tag{4.9}$$

the approximation of the image inside $\mathbf{R}$ can be written in terms of its approximation inside $\mathbf{R}_j$ and $\mathbf{R}_k$, i.e.,

$$\tilde{a} = \frac{A(f,\mathbf{R}_j) + A(f,\mathbf{R}_k)}{\#\mathbf{R}_j + \#\mathbf{R}_k} = \frac{\#\mathbf{R}_j \tilde{a}_j + \#\mathbf{R}_k \tilde{a}_k}{\#\mathbf{R}_j + \#\mathbf{R}_k}$$

Thus, if the pair of regions $\mathbf{R}_j$ and $\mathbf{R}_k$ to merge, usually restricted to being adjacent, is supposed to minimize E', it must be chosen so as to minimize the squared error contribution to the total error $D(\mathbf{R}_j, \mathbf{R}_k) = D_{jk} = \#\mathbf{R}e^2(f, \tilde{f}', \mathbf{R}) - \#\mathbf{R}_j e^2(f, \tilde{f}, \mathbf{R}_j) - \#\mathbf{R}_k e^2(f, \tilde{f}, \mathbf{R}_k)$. But, using (4.8) and (4.9) (cf. with Appendix B of [194]),

$$\begin{aligned} D(\mathbf{R}_j, \mathbf{R}_k) = D_{jk} &= B(f,\mathbf{R}) - \#\mathbf{R}a^T a - B(f,\mathbf{R}_j) + \#\mathbf{R}_j a_j^T a_j - B(f,\mathbf{R}_k) + \#\mathbf{R}_k a_k^T a_k \\ &= \frac{\#\mathbf{R}_j \#\mathbf{R}_k}{\#\mathbf{R}_j + \#\mathbf{R}_k} (\tilde{a}_j - \tilde{a}_k)^T (\tilde{a}_j - \tilde{a}_k) = \frac{\#\mathbf{R}_j \#\mathbf{R}_k}{\#\mathbf{R}_j + \#\mathbf{R}_k} d^2(\tilde{a}_j, \tilde{a}_k). \end{aligned} \tag{4.10}$$

It should be noticed that quantity D_{jk} for a pair of (adjacent) regions $\mathbf{R}_j$ and $\mathbf{R}_k$ does not change unless one of the two regions has been merged to another. Hence, if these quantities are stored for each pair of adjacent regions, the consequences of merging two regions remain relatively localized.

Finally, it should be noticed that, if the flat region model is to be used during region-oriented segmentation (either region growing or region merging), then the only quantities which must be stored inside the data structure representing the regions are $\#\mathbf{R}$, the number of its pixels, a, which is the approximation parameter, and perhaps $\mathbf{R}$, the set of region's pixels, in the form of a pixel list, for instance.

The affine region model

The case of the affine region model is simply a generalization of the flat region model. In each region $\mathbf{R}$ the image is approximated by

$$\hat{f}[v] = a + bv \quad \forall v \in \mathbf{R} \tag{4.11}$$

where b is a $n \times m$ parameter matrix, n is the color space dimension, and m is the dimension of the space over which the image is defined (2 for 2D images, 3 for 3D images). Hence, now $m + 1$ n-dimensional parameters have to be estimated.

Let Θ stand for $\begin{bmatrix} a & b \end{bmatrix}$. Then, equation (4.11) can be written

$$\hat{f}[v] = \Theta \begin{bmatrix} 1 \\ v \end{bmatrix} \tag{4.12}$$

Again the objective is to choose Θ so as to minimize the approximation error

$$e(f, \hat{f}, \mathbf{R}) = \sqrt{\frac{\sum_{v\in\mathbf{R}} d^2(f[v], \hat{f}[v])}{\#\mathbf{R}}}$$

or, given the definition of distance in (4.7),

$$e(f, \hat{f}, \mathbf{R}) = \sqrt{\frac{\sum_{v\in\mathbf{R}} (f[v] - \hat{f}[v])^T (f[v] - \hat{f}[v])}{\#\mathbf{R}}}.$$

Since $f[v]$ and $\hat{f}[v]$ are n-dimensional vectors for each v, it is obvious that

$$e(f, \hat{f}, \mathbf{R}) = \sqrt{\frac{\sum_{v\in\mathbf{R}} \sum_{l=0}^{n-1} (f_l[v] - \hat{f}_l[v])^2}{\#\mathbf{R}}}$$

and, exchanging the summation order

$$e(f, \hat{f}, \mathbf{R}) = \sqrt{\frac{\sum_{l=0}^{n-1} \sum_{v\in\mathbf{R}} (f_l[v] - \hat{f}_l[v])^2}{\#\mathbf{R}}}$$

Let the sites v in region $\mathbf{R}$ be arranged in a sequence v_k with $k = 0, \ldots, \#\mathbf{R} - 1$ (the order of the site vectors in the sequence is irrelevant). Then

$$e(f, \hat{f}, \mathbf{R}) = \sqrt{\frac{\sum_{l=0}^{n-1} \sum_{k=0}^{\#\mathbf{R}-1} (f_l[v_k] - \hat{f}_l[v_k])^2}{\#\mathbf{R}}},$$

or

$$e(f, \hat{f}, \mathbf{R}) = \sqrt{\frac{\sum_{l=0}^{n-1} \|f_l - \hat{f}_l\|^2}{\#\mathbf{R}}},$$

where $f_l = \left[f_l[v_0] \ldots f_l[v_{\#\mathbf{R}-1}]\right]^T$ and $\hat{f}_l = \left[\hat{f}_l[v_0] \ldots \hat{f}_l[v_{\#\mathbf{R}-1}]\right]^T$. Using (4.12),

$$e(f, \hat{f}, \mathbf{R}) = \sqrt{\frac{\sum_{l=0}^{n-1} \|f_l - V(\mathbf{R})\theta_l^T\|^2}{\#\mathbf{R}}},$$

where θ_l is the lth line of matrix Θ, and

$$V(\mathbf{R}) = \begin{bmatrix} 1 & v_0^T \\ \vdots & \vdots \\ 1 & v_{\#\mathbf{R}-1}^T \end{bmatrix}$$

Since the term l of the summation depends only on θ_l, minimization of $e(f, \hat{f}, \mathbf{R})$ is equivalent to minimization of each term of the summation. Minimizing each of these terms is the same as finding the least squares solution to the equations

$$V(\mathbf{R})\theta_l^T = f_l \quad \text{for } l = 0, \ldots, n-1. \tag{4.13}$$

It is well known that [15]:

1. each equation $V(\mathbf{R})\theta_l^T = f_l$ has a least squares solution;
2. there is a unique least squares solution to each of these equations iff $\operatorname{rank}(V(\mathbf{R})) = m+1$; and
3. a vector $\tilde{\theta}_l$ is a least squares solution to $V(\mathbf{R})\theta_l^T = f_l$ iff $\tilde{\theta}_l$ is a solution to $V^T(\mathbf{R})V(\mathbf{R})\tilde{\theta}_l^T = V^T(\mathbf{R})f_l$.

Given that

$$V^T(\mathbf{R})V(\mathbf{R}) = \begin{bmatrix} \#\mathbf{R} & \sum_{k=0}^{\#\mathbf{R}-1} v_k^T \\ \sum_{k=0}^{\#\mathbf{R}-1} v_k^T & \sum_{k=0}^{\#\mathbf{R}-1} v_k v_k^T \end{bmatrix} = \begin{bmatrix} \#\mathbf{R} & \sum_{v\in\mathbf{R}} v^T \\ \sum_{v\in\mathbf{R}} v & \sum_{v\in\mathbf{R}} vv^T \end{bmatrix}$$

and

$$V^T(\mathbf{R})f_l = \begin{bmatrix} \sum_{k=0}^{\#\mathbf{R}-1} f_l[v_k] \\ \sum_{k=0}^{\#\mathbf{R}-1} f_l[v_k]v_k \end{bmatrix} = \begin{bmatrix} \sum_{v\in\mathbf{R}} f_l[v] \\ \sum_{v\in\mathbf{R}} f_l[v]v \end{bmatrix},$$

the least squares solution to the set of equation in (4.13) can be written

$$\tilde{\Theta} K(\mathbf{R}) = L(f, \mathbf{R}), \tag{4.14}$$

where

$$\begin{aligned}
K(\mathbf{R}) &= V^T(\mathbf{R}) V(\mathbf{R}) = \begin{bmatrix} \#\mathbf{R} & D^T(\mathbf{R}) \\ D(\mathbf{R}) & E(\mathbf{R}) \end{bmatrix}, \\
L(f, \mathbf{R}) &= \begin{bmatrix} f_0^T V(\mathbf{R}) \\ \vdots \\ f_{n-1}^T V(\mathbf{R}) \end{bmatrix} = \begin{bmatrix} A(f, \mathbf{R}) & C(f, \mathbf{R}) \end{bmatrix}, \\
C(f, \mathbf{R}) &= \sum_{v \in \mathbf{R}} f[v] v^T, \\
D(\mathbf{R}) &= \sum_{v \in \mathbf{R}} v, \text{ and} \\
E(\mathbf{R}) &= \sum_{v \in \mathbf{R}} v v^T.
\end{aligned}$$

Equation (4.14) is guaranteed to have a solution. It is also a good candidate for input to a numerical routine, since, unlike the previous equations, it has a fixed dimension. The solutions are obviously equivalent, given the properties of the least squares problem, with the advantage that least squares routines usually provide a solution even in the case of underdetermination.

By simple algebraic manipulation, it is straightforward to see that the minimum error is

$$e(f, \tilde{f}, \mathbf{R}) = \sqrt{\frac{B(f, \mathbf{R}) - \sum_{l=0}^{n-1} \tilde{\theta}_l K(\mathbf{R}) \tilde{\theta}_l^T}{\#\mathbf{R}}}.$$

As in the case of the the flat region model, if two disjoint regions $\mathbf{R}_j$ and $\mathbf{R}_k$ are united into a single region $\mathbf{R}$, the following results hold trivially

$$\begin{aligned}
K(\mathbf{R}) &= K(\mathbf{R}_j) + K(\mathbf{R}_k), \\
L(f, \mathbf{R}) &= L(f, \mathbf{R}_j) + L(f, \mathbf{R}_k), \\
C(f, \mathbf{R}) &= C(f, \mathbf{R}_j) + C(f, \mathbf{R}_k), \\
D(\mathbf{R}) &= D(\mathbf{R}_j) + D(\mathbf{R}_k), \text{ and} \\
E(\mathbf{R}) &= E(\mathbf{R}_j) + E(\mathbf{R}_k),
\end{aligned} \tag{4.15}$$

from which

$$\tilde{\Theta}\big(K(\mathbf{R}_j) + K(\mathbf{R}_k)\big) = \tilde{\Theta}_j K(\mathbf{R}_j) + \tilde{\Theta}_k K(\mathbf{R}_k)$$

Finally, the contribution of this union to the squared error is

$$D_{jk} = B(f, \mathbf{R}) - \sum_{l=0}^{n-1} \tilde{\theta}_l K(\mathbf{R}) \tilde{\theta}_l^T - B(f, \mathbf{R}_j) + \sum_{l=0}^{n-1} \tilde{\theta}_{j_l} K(\mathbf{R}_j) \tilde{\theta}_{j_l}^T - B(f, \mathbf{R}_k) + \sum_{l=0}^{n-1} \tilde{\theta}_{k_l} K(\mathbf{R}_k) \tilde{\theta}_{k_l}^T$$

which, using (4.9), can be reduced to

$$D_{jk} = \sum_{l=0}^{n-1} \tilde{\theta}_{j_l} K(\mathbf{R}_j) \tilde{\theta}_{j_l}^T + \tilde{\theta}_{k_l} K(\mathbf{R}_k) \tilde{\theta}_{k_l}^T - \tilde{\theta}_l K(\mathbf{R}) \tilde{\theta}_l^T \tag{4.16}$$

Each term of equation (4.16) represents the contribution of each color component, and can be further reduced whenever $K(\mathbf{R})$ is non-singular.

For the sake of brevity, let $K = K(\mathbf{R})$, $K_j = K(\mathbf{R}_j)$, $K_k = K(\mathbf{R}_k)$, $\theta = \tilde{\theta}_l$, $L = L_l(f, \mathbf{R})$, $L_j = L_l(f, \mathbf{R}_j)$, $L_k = L_l(f, \mathbf{R}_k)$, $\theta_j = \tilde{\theta}_{j_l}$, and $\theta_k = \tilde{\theta}_{k_l}$, where $L_l(f, \mathbf{R})$ is the lth line of $L(f, \mathbf{R})$. Then

$$\begin{aligned}
&\theta_j K_j \theta_j^T + \theta_k K_k \theta_k^T - \theta K \theta^T \\
&\quad = \theta_j K K^{-1} K_j \theta_j^T + \theta_k K K^{-1} K_k \theta_k^T - \theta K K^{-1} K \theta^T \\
&\quad = \theta_j (K_j + K_k) K^{-1} K_j \theta_j^T + \theta_k (K_j + K_k) K^{-1} K_k \theta_k^T - L K^{-1} L^T \\
&\quad = \theta_j K_j K^{-1} K_j \theta_j^T + \theta_j K_k K^{-1} K_j \theta_j^T + \theta_k K_j K^{-1} K_k \theta_k^T + \theta_k K_k K^{-1} K_k \theta_k^T \\
&\qquad - (L_j + L_k) K^{-1} (L_j^T + L_k^T) \\
&\quad = \theta_j K_j K^{-1} K_j \theta_j^T + \theta_j K_k K^{-1} K_j \theta_j^T + \theta_k K_j K^{-1} K_k \theta_k^T + \theta_k K_k K^{-1} K_k \theta_k^T \\
&\qquad - (\theta_j K_j + \theta_k K_k) K^{-1} (K_j \theta_j^T + K_k \theta_k^T) \\
&\quad = \theta_j K_j K^{-1} K_j \theta_j^T + \theta_j K_k K^{-1} K_j \theta_j^T + \theta_k K_j K^{-1} K_k \theta_k^T + \theta_k K_k K^{-1} K_k \theta_k^T \\
&\qquad - \theta_j K_j K^{-1} K_j \theta_j^T - \theta_j K_j K^{-1} K_k \theta_k^T - \theta_k K_k K^{-1} K_j \theta_j^T - \theta_k K_k K^{-1} K_k \theta_k^T \\
&\quad = \theta_j K_k K^{-1} K_j \theta_j^T + \theta_k K_j K^{-1} K_k \theta_k^T - \theta_j K_j K^{-1} K_k \theta_k^T - \theta_k K_k K^{-1} K_j \theta_j^T \\
&\quad = (\theta_j - \theta_k) K_k K^{-1} K_j \theta_j^T + \theta_k K_j K^{-1} K_k \theta_k^T - \theta_j K_j K^{-1} K_k \theta_k^T \\
&\quad = (\theta_j - \theta_k) K_k K^{-1} K_j (\theta_j - \theta_k)^T
\end{aligned}$$

where use has been made of the fact that $A(A+B)^{-1}B = B(A+B)^{-1}A$.[6]

Hence

$$D(\mathbf{R}_j, \mathbf{R}_k) = D_{jk} = \sum_{l=0}^{n-1} (\tilde{\theta}_{j_l} - \tilde{\theta}_{k_l}) K(\mathbf{R}_k) K^{-1}(\mathbf{R}) K(\mathbf{R}_j) (\tilde{\theta}_{j_l} - \tilde{\theta}_{k_l})^T \tag{4.17}$$

which has the same role for affine region models that equation (4.10) had for flat region models.

As in the case of the flat region model, quantity D_{jk} for a pair of (adjacent) regions $\mathbf{R}_j$ and $\mathbf{R}_k$ does not change unless one of the two regions has been merged to another. Hence, if these quantities are stored for each pair of adjacent regions (for each arc in the RAG), the consequences of merging two regions remain relatively localized.

Also, if the affine region model is to be used during region-oriented segmentation (either region growing or region merging), then the only quantities which must be stored inside the data

[6]Assuming that $A+B$ is non-singular $A(A+B)^{-1}B = (A+B)(A+B)^{-1}B - B(A+B)^{-1}B = B - B(A+B)^{-1}B = B - B(A+B)^{-1}(A+B) + B(A+B)^{-1}A = B(A+B)^{-1}A$.

structure representing the regions are $K(\mathbf{R})$, Θ, and perhaps $\mathbf{R}$, the set the region's pixels, in the form of a pixel list, for instance. Although not strictly necessary, $L(f, \mathbf{R})$ may also be stored, at the expense of extra memory requirements, in order to increase processing speed. Notice that $K(\mathbf{R})$ plays the role of $\#\mathbf{R}$ in the flat region model: it depends only on the region shape, and not on its color, the color information being concentrated into the parameter Θ. Also notice that in this case, instead of storing an integer ($\#\mathbf{R}$) and n floating points ($n \times 1$ vector a), as in the case of the flat region model, $(m+1)^2$ integers ($m+1 \times m+1$ matrix $K(\mathbf{R})$) and $n(m+1)$ floating points ($n \times m+1$ matrix Θ) have to be stored. Since segmentation algorithms have typically heavy memory requirements, the use of the affine region model for all but the smallest images is still not practical on typical workstations.

Two important questions must still be answered if this model is to be applied with success in the future. What should be done if the least squares solution is not unique, i.e., if $\operatorname{rank}(V(\mathbf{R})) < m+1$? And what is the meaning of such multiple solutions?

Conditions for uniqueness

Multiple solutions occur when $r = \operatorname{rank}(V(\mathbf{R})) < m+1$. Matrix $V(\mathbf{R})$ is a $\#\mathbf{R} \times m+1$ matrix, and thus its rank is always $r \leq m+1$ and $r \leq \#\mathbf{R}$. If $\#\mathbf{R} < m+1$, then $r < m+1$, and there are multiple solutions to the least squares problem. In this case the problem is simply that there isn't enough data to compute the approximation parameters. If $\#\mathbf{R} \geq m+1$ (or $\#\mathbf{R} > m$) but nevertheless $r < m+1$ (or $r \leq m$), then there must be exactly r linearly independent rows of $V(\mathbf{R})$.[7] Without loss of generality, let the first r rows of $V(\mathbf{R})$ be linearly independent. Then, all other rows must be linear combinations of those r rows. That is,

$$\begin{bmatrix} 1 \\ v_j \end{bmatrix} = \sum_{k=0}^{r-1} \alpha_{jk} \begin{bmatrix} 1 \\ v_k \end{bmatrix},$$

for some set of α_{jk} with $j = r, \ldots, \#\mathbf{R}$ and $k = 0, \ldots r-1$. Separating the first row of the matrix equation

$$\begin{aligned} 1 &= \sum_{k=0}^{r-1} \alpha_{jk} \\ v_j &= \sum_{k=0}^{r-1} \alpha_{jk} v_k \end{aligned}$$

or

$$\begin{aligned} \alpha_{j0} &= 1 - \sum_{k=1}^{r-1} \alpha_{jk} \\ v_j &= (1 - \sum_{k=1}^{r-1} \alpha_{jk}) v_0 + \sum_{k=1}^{r-1} \alpha_{jk} v_k \end{aligned}$$

[7]Notice that $r \geq 1$, since $V(\mathbf{R})$ by construction cannot consist solely of null rows and since $\mathbf{R}$ is non-empty by assumption.

that is,

$$v_j = v_0 + \sum_{k=1}^{r-1} \alpha_{jk}(v_k - v_0). \tag{4.18}$$

But (4.18) is the equation of a point, if $r = 1$, of a line, if $r = 2$, of a plane, if $r = 3$, an so on. Hence, the least squares solution is not unique whenever the set of sites is "aligned" along a hyperplane of dimension $r - 1 < m$. In the case of 2D images, with $m = 2$, there are multiple solutions if the sites in $\mathbf{R}$ are aligned along a line. In the case of 3D images, with $m = 3$, there are multiple solutions if the sites in $\mathbf{R}$ are aligned along a line or "aligned" along a plane.

The case of $r = 1$, where all sites coincide, can only occur if $\#\mathbf{R} = 1$, since sets do not have repeated elements. But since $\#\mathbf{R} > m$ by hypothesis, these cases are automatically ruled out in the cases of interest ($m = 2$ for 2D images and $m = 3$ for 3D images). These cases have been classified above as cases without enough data.

In conclusion, in the case of 2D (3D) images, there is a unique least squares solution if there are three (four) non-collinear (non-coplanar) sites in $\mathbf{R}$.

Dealing with non-uniqueness

If there is no unique least squares solution, which of the possible solutions should be chosen? How will it affect segmentation, in the case of region oriented segmentation? If the initial regions are all one-pixel wide, it is clear that the error contribution of all possible mergings will be the same (viz. zero). But this is clearly undesirable. The problem stems from using a powerful model for modeling regions which are too small (with only two pixels, resulting from merging any pair of adjacent one-pixel wide regions). This may be solved, in the case of 2D images, by specifying that a flat region model should be used for one and two-pixel wide regions, the affine model being reserved for regions with more than two pixels. Even though this does not eliminate the all the sources of non-uniqueness (see the previous section), it does eliminate those cases where non-uniqueness is really a problem.

Other solutions may also be used. If the image is split initially into 2×2 square regions, then there is always a unique solution to the least square problem. However, the segmentation resolution will clearly suffer. An alternative solution, without this drawback, is to upsample the image by a factor of two in both directions before splitting into 2×2 square regions. This will, however, lead to increased memory requirements (by a factor of at least 4).

Strictly speaking, the above problem does not have to do with non-uniqueness. It is related with the steepest descent approach to obtaining the optimal segmentation used by most segmentation algorithms: at each step choose the regions to merge so as to minimize the error. However, these incremental minimizations are not guaranteed to converge to a true global minimum. Actually, they seldom do. The problem above is actually one of over adjustment of the model to the data, which leeds to some bad decisions by the segmentation algorithms. It seems that the model to be used should always be insufficient to represent general data accurately, if it is to have some meaning. This, at least intuitively, is coherent with the knowledge that the "best" possible region model is the one which specifies independent values for the colors of all the pixels in the image, which can model without error the image segmented into a single region, but which conveys no useful information.

Boundary modeling

Boundary modeling can be useful both for region- and contour-oriented segmentation, as will be seen in the following. Notice that, even though the issue is certainly important enough to deserve separate treatment in Section 6.2, boundary shape (region shape), will not be dealt with here. As in the case of region modeling, the problem is to model the image around a boundary.

Unlike region model, where a region consists of a finite, well known set of pixels, boundaries are contiguous sets of edges. Images do not have values at edges. Two approaches are possible for boundary modeling. The first, which may be said to be the classical one, is concerned about the derivatives of the image along a boundary. The second attempts to model the image on a more or less narrow strip of pixels along a boundary.

The classical approach, which estimates image derivatives, is typically used in straightforward extensions of the basic contour closing algorithm. What's more, the basic contour closing algorithm can be though of as using the roughest possible estimate of the image gradient in the direction orthogonal to the edge direction: the difference of the pixel colors. Hence, the edge model, in this case, is simply a horizontal facet which passes through the two pixels separated by the edge. Modeling in this case is the process which leads to estimation of derivatives, and thus is equivalent to the derivative computation component of edge detection operators. A complicated issue, which has not been dealt with in this thesis, is establishing the meaning of derivatives in the case of non-scalar color models [36].

The second approach has been typically used for image representation. Some articles, notably [58, 19, 37, 43], recognized that, since the HVS is especially sensitive to rapid color transitions, usually corresponding to physical edges, images may be represented by edges without loss of semantical information, in very much the same way artists can economically represent a scene with a few strokes. In [37], for instance, the image in a section orthogonal to the detected edge is modeled as a step edge blurred by a Gaussian filter. Hence, three parameters have to be estimated at each such section: the mean point, the amplitude, and the sharpness of the transition. In pratice, a fourth parameter is also estimated: the extent to both sides of the edge over which the model is a good approximation. Notice, however, that in [37] this image model is used to represent the image, not to segment it.

Globalization of region growing

Globalization is simple, in the case of region growing. Instead of basing the pixel aggregation order on pairwise pixel color distances, the order is now based on how well the candidate pixels fit into the corresponding region model, whose parameters are estimated using the complete set of pixels in the region at each instant.

For the flat region model used on typical algorithms, the degree of fitness into a region may be simply the distance between the color of the candidate pixel $f[v]$ and the estimated parameter $\tilde{a}$ of the corresponding region $\mathbf{R}$, i.e., $d^2(f[v],\tilde{a})$, if the pixel is at site v. The squared distance will be used in order to make the comparison between several globalization methods direct. It is immaterial to use the squared distance, since the order relations are preserved by the monotonous function $f(x)=x^2$ for $x \geq 0$.

For the affine region model, the fitness may be calculated as the distance between the color of the candidate pixel $f[v]$ and $\tilde{\Theta}\begin{bmatrix}1 & v^T\end{bmatrix}^T$, that is $d^2(f[v],\tilde{\Theta}\begin{bmatrix}1 & v^T\end{bmatrix}^T)$. In both cases, the distance may be expressed more concisely as $d^2(f[v],\tilde{f}[v])$, where $\tilde{f}$ is an approximation to f over the union of the pixel and region in consideration but whose parameters have been estimated without considering that pixel's value, i.e., it is the (squared) distance between the pixel's color and the color obtained by extrapolating the region model to the pixel's location.

Another possibility is to select the pixel to aggregate as the one leading to the smallest increase of the global approximation error, as given by equations (4.10) and (4.17), according to the model used. These equations, assuming that $\mathbf{R}_j$ corresponds to the candidate pixel at site v, i.e., $\mathbf{R}_j=\{v\}$, and $\mathbf{R}_k$ corresponds to the region with which it may be merged, i.e., $\mathbf{R}_k=\mathbf{R}$, simplify respectively to

$$D(\{v\},\mathbf{R})=\frac{\#\mathbf{R}}{1+\#\mathbf{R}}d^2(f[v],\tilde{a})$$

and

$$D(\{v\},\mathbf{R})=\sum_{l=0}^{n-1}\left(\begin{bmatrix}f_l[v] & 0\end{bmatrix}-\tilde{\theta}_l\right)K(\mathbf{R})\left(K(\mathbf{R})+\begin{bmatrix}1\\v\end{bmatrix}\begin{bmatrix}1 & v^T\end{bmatrix}\right)^{-1}\begin{bmatrix}1\\v\end{bmatrix}\begin{bmatrix}1 & v^T\end{bmatrix}\left(\begin{bmatrix}f_l[v] & 0\end{bmatrix}-\tilde{\theta}_l\right)^T,$$

where $\tilde{a}$ and $\tilde{\Theta}$ are the estimated parameters for region $\mathbf{R}$ using the flat and affine region models respectively, and where $\tilde{\theta}_{j_l}=\begin{bmatrix}f_l[v] & 0\end{bmatrix}$, for $l=0,\ldots,n-1$, is the simplest solution to (4.13) when $\mathbf{R}_j=\{v\}$.

Notice that $d^2(f[v],\tilde{\Theta}\begin{bmatrix}1 & v^T\end{bmatrix}^T)$ may be written as

$$d^2\left(f[v],\tilde{\Theta}\begin{bmatrix}1\\v\end{bmatrix}\right)=\sum_{l=0}^{n-1}\left(\begin{bmatrix}f_l[v] & 0\end{bmatrix}-\tilde{\theta}_l\right)\begin{bmatrix}1\\v\end{bmatrix}\begin{bmatrix}1 & v^T\end{bmatrix}\left(\begin{bmatrix}f_l[v] & 0\end{bmatrix}-\tilde{\theta}_l\right)^T.$$

Hence, minimizing the increase in global approximation error tends to favor merging of pixels with smaller regions, though in the case of the affine region model the effect of region size may be countered by effects of region shape.

It should be noticed that, after each pixel aggregation, the parameters of the region are adjusted to reflect the presence of that further pixel, but, even more important, the contribution to the global error of all other pixels adjacent to that region also change as well. The consequences of this fact are that:

1. the arcs which are inserted into the tree do not in general form a SSSSkT of the image graph at the completion of the algorithm; however, since at each step the arcs are chosen "the right way", this may be said to be a recursive SSSSkT algorithm; and

2. the algorithm complexity increases, since several arcs in the priority queue have their weight updated after each step, which requires reorganization of the queue.

Notice that the applicable algorithms are adjustments of constructive SSSSkT algorithms. The destructive algorithms cannot be changed in any simple way to achieve the same result. Also, even though the complexity of the constructive algorithm increases, hierarchical queues still seem to be the best possible structures to use. Both these issues have been left for future work.

Watersheds

In order to avoid the increased algorithm complexity which results from recalculating weights of arcs already in the priority queue, [177] proposed to reestimate the region model parameters after pixel aggregation but to leave unchanged the weight of pixels already in the priority queue (this problem is not acknowledged in [177]). This means that, at each moment, the arcs in the queue have weights corresponding to region model parameters estimated at different time instants. The consequences of this fact, however, do not seem to be tragic, since for regions of reasonable size the parameters do not change much after each pixel aggregation.

This algorithm is essentially the one used in Sesame [30], the segmentation-based candidate for Verification Model during the development of MPEG-4. The region model used in Sesame is still the flat region model. However, a hierarchy of segmentation results is produced by encoding the results of segmentation, using a more powerful region model, and resegmenting with increased detail those parts of the image where the approximation error is larger. The merits of this idea are threefold: it allows for scalability in a quite elegant way, it takes quantization effects into account during the segmentation process (not during each of the runs of the segmentation algorithm, but during the calculation of the segmentation hierarchy), and it leads to acceptable computational complexity. As computer power increases, however, the justification for not integrating more complex region models (and maybe quantization effects), during the segmentation algorithm tends to vanish.

Globalization of region merging

As in the case of region growing, region merging has been typically performed in two ways: either by comparing the region model parameters using some kind of metric, or by using the contribution of the merging to the global error. In both cases, in parallel with the region growing case, the constructive algorithms change, since there is the need to update the weight (priority) of several arcs in the queue whenever two regions are merged. The solved problem thus ceases to be the SSkT problem, though at each step of the algorithm the "right" arc is chosen. Hence, these algorithms have both been labelled by [134] and [194], who first proposed them, recursive SST algorithms, or RSST. Since the weights of the arcs not directly involved in a merging operation can change, it is not guaranteed that the arcs of the resulting spanning tree (the recursive one), are inserted in increasing order of weight. Hence, even though destructive algorithms can be applied afterwards, their meaning is less than clear.

Notice that the first type of globalization, which uses direct comparison of region model param-

$\#\mathbf{R}_j$	$\#\mathbf{R}_k$	$\frac{\#\mathbf{R}_j \#\mathbf{R}_k}{\#\mathbf{R}_j + \#\mathbf{R}_k}$
1	1	0.5
1	100	0.99
100	100	50

Table 4.1: The $\frac{\#\mathbf{R}_j \#\mathbf{R}_k}{\#\mathbf{R}_j + \#\mathbf{R}_k}$ factor for some region sizes.

eters, is well defined only for simple models, such as the flat region model, which corresponds to calculating the distance between the average colors of the two region candidate to a merging. In the case of more complicated models, appropriate metrics may be hard to find. But, since this globalization method has an inherently worst behavior than the second one, as will be seen in the sequel, the search for such metrics seems to be a worthless task.

The advantage of the second type of globalization, namely the one using the contribution to the global approximation error, stems from the inherent good treatment of small regions. Consider for a moment the flat region model. In the first case, the weight attributed to the union of two regions $\mathbf{R}_j$ and $\mathbf{R}_k$ is simply the (squared) distance between their average colors $d^2(\tilde{a}_j, \tilde{a}_k)$. In the second case, it is the contribution to the global approximation error, i.e., $\frac{\#\mathbf{R}_j \#\mathbf{R}_k}{\#\mathbf{R}_j + \#\mathbf{R}_k} d^2(\tilde{a}_j, \tilde{a}_k)$. The factor $\frac{\#\mathbf{R}_j \#\mathbf{R}_k}{\#\mathbf{R}_j + \#\mathbf{R}_k}$ accounts for the different treatment of small regions, since it is small whenever either (or both) of the regions are small (see Table 4.1 for three examples). Hence, small regions tend to grow faster than large regions, and the likelihood of small regions hanging around is reduced. Such small regions can really be a plague if the first type of globalization is used. Most of them derive from the unfortunate fact that, when using 4-neighborhoods, thin lines with about 45 degrees of slope produce a series of disconnected regions of a singe pixel. This effect will be shown in Section 4.5, which proposes an alternative way for dealing with this problem.

If the affine region model is used, the same comments apply, even if in this case the comparison is hampered by the influence of region shape in the contribution to the global approximation error (4.17) and by the absence of meaningful metrics for direct parameter comparison. However, a straightforward metric corresponding to $\sum_{l=0}^{n-1}(\tilde{\theta}_{j_l} - \tilde{\theta}_{k_l})(\tilde{\theta}_{j_l} - \tilde{\theta}_{k_l})^T$ may be used for comparison purposes.

It should be stated here that, even though globalization of information allows the segmentation to get closer to the optimum, it can be shown easily, by a counter example, that the algorithm is not guaranteed to attain the optimum. In the following example, a grey-scale (scalar color) 1×4 image is segmented into two regions by globalized region merging using the contribution to the global error and the flat region model (regions numbered from 0 at the left):

(a)	1	2.1	2.9	4
(b)	1	2.5		4
(c)	2			4
(d)	1.55		3.45	

In the original original image (a), the contributions to the global error are $D_{01} = 0.605$, $D_{12} = 0.32$, and $D_{23} = 0.605$. In (b) the center regions, which contribute less to the global error, have been merged, resulting in $D_{01} = 1.5$ and $D_{12} = 1.5$. Finally, in (c) the two first regions have been merged (in this case the choice is irrelevant), leading to segmentation with a global approximation error of $E = \sqrt{0.455}$. This segmentation is worst than the optimum segmentation in (d), which has a global approximation error of $E = \sqrt{0.3025}$.

In a sense, the globalized segmentation algorithms work like steepest descent optimization, which are known to lead to local optima but in general not to global optima. Reviews of methods which attempt to solve this problem, at the expense of increased computational complexity, can be found in [104, 147].

Region merging with seeds

Globalization can be applied in much the same way to region merging with seeds. While the non-globalized, basic region growing and seeded region merging algorithms lead to equivalent results, in the sense that both solve the SSSSkT, their globalizations have different properties.

Firstly, it should be noticed that, unlike the case of region growing, now arbitrary regions can be merged, unless both contain pixels of different seeds, which results in a faster globalization of information, especially in the case of using the contribution to the global approximation error as arc weights, since, as seen in the last section, it tends to favor mergings of the smaller regions.

One of the practical results of this fact is that pixels of a seedless but uniform zone of the image tend to be aggregated into a single region, which at a later time will be merged to some seeded region. Such seedless uniform zones are often split between two or more different neighboring seeds in the case of region growing, especially in the case of watershed segmentation, which was built so as to explicitly divide those zones, plateaus, among various basins. But the division of these zones typically corresponds to splitting part of an object, which is undesirable in the case of image analysis, which aims at identifying whole objects.

Another advantage of globalized region merging with seeds already existed in the basic algorithm: the intermediate segmentation results are meaningful. This means that the algorithm may be stopped immediately if, for instance, the global approximation error exceeds a threshold, thereby producing a meaningful partition of the image which includes some seedless regions, which were deemed unmergeable to neighboring seeded regions. This nice behavior of region merging with seeds is the reason why it was chosen as a good tool for supervised segmentation.

Globalization of contour closing

Globalization of contour closing is not easy. Since the arcs are selected not for merging but for splitting regions, it is not clear what data should be used to estimate the boundary model parameters. That is why such models are estimated in a somewhat arbitrary neighborhood of edges. Such is the case of Sobel and related estimators of the image derivatives.

Boundary information may be used to improve the results of region-oriented segmentation [194, 158]. The rationale for such methods stems from the fact that segmentation often leads to boundaries in zones where there are really no strong transitions in the image. However, the reason for these false contours, as they are called, is the failure of models to represent faithfully the color of real regions. This is obviously the case if the flat region model is used to segment a uniformly sloped region, which would be perfectly represented by the affine region model. It is arguable that the best solution to the false contours problem would be to devise better region models, but in practice this is often a hard task, fundamentally because of the added algorithm complexity. Besides, in order to produce meaningful segmentations, the region models cannot be so complex as to represent every possible image: segmentation is well defined only if the region models are not too powerful, and hence false contours must be dealt with using other techniques, such as the ones in [194, 158].

Shape restrictions

Boundary information may also be used to restrict region-oriented segmentation so as to produce partitions where the boundaries do not exhibit too much busyness. The reasons for this derive from the fact that everyday objects often have regular boundaries,[8] and, more importantly, from the fact that, if the partition is to be encoded, boundary busyness can lead to very expensive representations. Instead of forcing the encoder to use lossy techniques, and thus to introduce boundary simplifications in a blind way, if image analysis and coding are more closely integrated, segmentation algorithms can attempt to reduce boundary busyness themselves, thus achieving a result which is hopefully equivalent in terms of savings at the encoder. This idea has been suggested in [30], where increases in boundary complexity, which result from adding a pixel to a region, are used together with color differences to decide which pixel to merge next in the watershed segmentation algorithm.

4.3.5 Algorithms and the dual graphs

All the globalized algorithms, with the exception of globalized contour closing and of region-oriented algorithms making use of boundary information, require only information about the adjacency of regions. A RAG in which each region contains a list of its pixels is sufficiently powerful to represent the partition at each step of the algorithm. However, when boundary information must be taken into account, it is often of interest to distribute that information through the pieces of border that constitute each boundary. Also, since partitions will often need to be encoded, and some of the partition encoding schemes make use of contour topology, it may be important to use the dual RAMG and RBPG graphs while performing segmentation. This, of course, assuming 2D partitions are the aim.

For all constructive segmentation algorithms, it is possible to keep a pair of dual region (RAMG) and contour (RBPG) graphs, that is, a map, representing the current partition. All that has to be done is to perform region merging as indicated in Section 3.5.1 as regions are merged, starting with the trivial graph in which each pixel is a region.

[8]However, some boundaries in natural scenes can have a fractal nature.

The data structure implementing the pair of dual graphs representing the map, in this case an evolving partition, may store information about borders, in each region, in a hierarchical way. It may be useful to access borders one by one, or bunched together in super-borders consisting of all the borders between given pairs of adjacent regions. Such super-borders correspond obviously to arcs of the underlying RAG.

In order to save memory and to speed up access to the borders of a region or to the regions separated by a border, the data structure can also make use of the fact that arcs are shared among two dual graphs. Several data may be stored in region nodes, and in border arcs, and even in super-borders. Regions nodes typically store the region size (measured by the number of pixels, which is an area for 2D partitions and a volume for 3D partitions), the set of pixels in the region, a set of statistics of these pixels, and parameters of a model adapted to the values of the image at the region pixels. Borders typically store the border size (a perimeter measured in number of edges, in the case of 2D partitions, and a surface area measured in number of faces, in the case of a 3D partition),[9] a deque (double-ended queue) of their edges, which may be useful for tracing the border later on, statistics of the transitions in image color along the border, and parameters of a boundary model adapted to the values of the image around the border. Finally, super-borders typically store a weight which has to do with the homogeneity of the region resulting from removal of the corresponding borders. This weight may ponder also the characteristics of the corresponding borders, in the case of region-oriented algorithms making use of boundary information.

4.3.6 Conclusions

This section presented a structured overview of various segmentation algorithms, whose basic versions are related to SST problems, and whose globalization, while making their properties diverge, hopefully leads to algorithms which are closer to the optimum in some sense.

Using the classification in Section 4.2, the presented algorithms are, strictly speaking, segmentation techniques, which may or may not be included into segmentation algorithms. They are region-oriented (with the exception of contour closing), generic, memoryless (except in the case of 3D), and either vectorial or scalar.

4.4 A new knowledge-based segmentation algorithm

After ITU-T[10] issued H.261, aiming at bitrates $p \times 64$ kbit/s with $p = 1, \ldots, 32$, when ISO/IEC had already issued MPEG-1, for up to about 1.5 Mbit/s, and work on MPEG-2 was being finalized, the need for very low bitrate coding techniques and standards began to be felt. The main application behind the expected need for such very low bitrate techniques was the mobile telephony. Eventually, the research in this area gave birth to a new standard, H.263, and

[9]Maps have not been defined for 3D partitions, so strictly speaking the surface would be stored only in super-borders of a RAG.

[10]Then CCITT (Comité Consultatif Internationale de Télégraphique et Téléphonique).

sparked work on MPEG-4, which was later revamped to be much more than a standard for very low bitrate video coding, as seen in Chapter 2.

Two parallel paths were taken towards the development of very low bitrate video coding techniques. One tried to make many small improvements to the existing techniques, basically motion compensated hybrid coding, and another attempted to reach a breakthrough in compression by using techniques which, being related or based on mid-level vision concepts, may be termed second-generation. The first path was quite successful at squeezing more compression out of old techniques: H.263 substantially outperformed H.261 at low bitrates and even above. For the second path, however, there did not seem to exist mature enough technology. Effective analysis techniques were required but unavailable. Without such analysis techniques, how could a standard be developed for very low bitrate applications on a tight agenda? Besides, in terms of compression, though the research investment seems clearly worthwhile, the results attained did not seem too good: in the framework of MPEG-4, core experiments demonstrated either the superiority of the mature motion compensated hybrid coding technology, or that improvements using other techniques were not very significant.

The solution to this dilemma was found by recognizing the growing importance of the interaction with the visual scene. Mid-level vision concepts, such as the object, might still be useful, if not for compression at least for manipulation, or for added functionalities. The object thus became the center of MPEG-4. Easy access to content as one of the objectives of encoding was no longer frame- or image-based, but became object-based. Granted, full-proof automatic second-generation analysis techniques were, and still are, but a wish, but MPEG-4 will not standardize analysis, just syntax and decoding. Hence, it will be ready by the time those techniques finally arrive. Expertise will grow meanwhile, through the use of supervised analysis techniques, and MPEG-4 will still be usable, e.g. if objects are segmented using classical TV techniques such as chroma-keying.

In between the two stated paths, a few other paths were also taken towards very low bitrate coding and ultimately object-based content access. One of them was the improving of existing codecs through slow integration of second-generation techniques. The knowledge-based segmentation proposed in this section, and global motion estimation, cancellation and compensation, as described in Sections 5.5 and 6.1, can be seen as the result of this effort, and thus clearly belong to the so-called transition towards second-generation video coding tools.

4.4.1 Introduction

Knowledge-based video coding algorithms can be applied to advantage in the path towards second-generation video coding and very low bitrate video coding. The main idea behind it is that the observer of a videotelephone sequence, typically a head and shoulders scene, is particularly sensitive to the image quality in areas such as the speaker's eyes and mouth, less to the quality in the speaker's body, and even less to the quality in the background. Knowledge-based video coding algorithms attempt to distribute the available bits so that quality is concentrated where it is really needed. This scheme, while keeping or even lowering the global objective quality measures (e.g., PSNR), improves the subjective quality of the encoded sequence. What's more, it can be easily integrated in existing first-generation encoders (e.g., H.261 compatible [62]) while maintaining full compatibility with existing decoders.

Segmentation is a fundamental step in knowledge-based video coding algorithms. Again using Pavlidis' words [156] "segmentation identifies areas of an image that appear uniform to an observer, and subdivides the image into regions of uniform appearance." As said before, the uniformity criterion can be chosen in many different ways. One may, for instance, envisage a type of segmentation where one desires to identify certain objects known to be in an image or image sequence. This is knowledge-based segmentation. This section presents a knowledge-based segmentation algorithm for videotelephony which can cope with a wide range of sequences, studio based or mobile.

The objective is the segmentation of each image in a videotelephone sequence into three regions: head, body and background, each having different subjective quality impact upon the observer. However, as a first approach, admitting that the head/body separation can be based solely on geometrical reasoning, the objective can be reduced to the segmentation into two regions: speaker and background. This segmentation falls somewhere between the two definitions given before:

- a specific "known" object should be identified (the speaker);
- the object appearance is only know to a certain extent (must cope with any human speaker); and
- the position of the object is known a priori with a high probability (centered, facing camera, neither too close nor too far).

In spite of the fact that the segmentation of the typical videotelephone sequences (e.g., "Claire", "Trevor", "Salesman" and "Miss America") is relatively easy, see for instance [99, 160], the expected emergence of mobile/hand-held videotelephone services demand much more robust segmentation algorithms. Plompen [163], for instance, describes a simple method for segmentation. The rationale behind it is that, in studio or fixed camera videotelephone sequences, the significantly changed blocks (e.g., in terms of the mean absolute difference) will very likely be located only over the moving speaker. The complete segmentation is then obtained through a split and merge algorithm [55]. For the head/body segmentation simple geometric considerations are used, as in the method proposed below. However, this method is not appropriate for use in mobile sequences, where the whole background is potentially moving. See also [123, 125, 126] for preliminary versions of the algorithm.

Typical mobile sequences (e.g., "Foreman" and "Carphone") contain a lot of background movement (originated by hand-held camera movement in "Foreman", and by camera vibration and the passing landscape in "Carphone"), making it difficult for techniques using simple difference operators to produce acceptable results. These sequences also usually contain a highly detailed background, complicating the task of edge detection based segmentation algorithms.

The algorithm presented here, which is an evolution of the knowledge-based algorithms proposed by [160] and by [166], divides each image into three distinct areas, head, body and background, at the H.261 MB resolution (i.e., 16×16 pixels). The robustness of the algorithm stems from its attempt to dynamically classify the input sequence into one of four classes, according to the uniformity of the background and to the presence of background/camera motion. Each videotelephone sequence is dealt with using the segmentation techniques appropriate for the detected class. The robustness of the proposed algorithm has been lacking in the knowledge-based segmentation algorithms available [99, 163, 160], which could only handle sequences with fixed, or even uniform, backgrounds.

The necessary segmentation resolution depends on the video encoder to be used. For instance, if the codec is compliant with H.261 and the quantization step is used to control the quality of the different segmented regions, then a segmentation at a MB resolution, as proposed here, suffices. It should be noticed here that the algorithm was developed for CIF (Common Intermediate Format) images. However, the algorithm is applicable, with adaptations, for other image sizes and at other resolutions.

Two basic pixel level operators, namely Sobel and image difference, are used to construct an activity map for the current image, which is subsequently decimated to MB resolution. The remaining steps of the algorithm operate at this lower resolution, which has the advantage that much of the processing deals with "images" of a much smaller size (with 256 times less elements than the original input images), resulting in a reduced computational weight.

The MB level processing includes the application of inertia to the results of segmentation, thus taking into account the high probability of small changes of the speaker position from image to image in a typical videotelephone sequence. It also includes knowledge-based geometric techniques which correct the shape of the obtained segmentation, and a final knowledge-based geometrical coherence quality estimation step whose result is used to adapt the algorithm parameters. The quality control is delayed, as the changes in the parameters take effect only for the next image in the sequence. The estimated quality is used as well for deciding whether to accept or reject the current segmentation.

4.4.2 Algorithm description

This section describes the main steps of the algorithm proposed. This algorithm can be classified as:

- MB (16×16 pixels) resolution, since the result of the segmentation has MB resolution;
- with memory, since it uses information about the previous images (by using the image difference operator, by using inertia of segmentation, and by using the memory mechanism, all explained in the following sections);
- knowledge-based, since it uses the a priori knowledge that the images represent typical head-and-shoulders videotelephone scenes;
- with knowledge-based, geometrical coherence quality estimation;
- with quality estimation for quality control and segmentation acceptance/rejection; and

- with delayed quality control, since the segmentation algorithm parameters are adjusted according to the current quality estimate but this adjustment is effective only in the next image.

A flow chart of the algorithm is presented in Figure 4.5.

Edge detection operators

Transition strength operators

At the low-level, the algorithm uses two different transition strength operators. The first is the Sobel transition strength operator, where the magnitude of the gradient, in order to reduce computational effort, is approximated here by the sum of the absolute values of the partial derivatives [56]:

$$\|\nabla f[i,j]\| \approx \mathrm{Sobel}(f[i,j]) = G[i,j] = |f_x[i,j]| + |f_y[i,j]|$$

where f_x and f_y are given by (4.1) and (4.2). It can be classified as a transition strength, scalar, 2D operator.

The second operator is the image difference described in equation (4.6). It can be classified as a scalar, 3D, transition strength operator, since, when movements from one image to the next are small, the image difference operator produces an approximation to the derivative in the direction of the motion.

Edge localization methods

The results of the low-level transition strength Sobel and image difference operators are then used for locating edges in the images. Since, as will be seen later, the results of this localization will be understood more as activity measures than as detected edges, the edge localization methods used are very simple:

Sobel operator
A pixel $p = [i,j]$ is considered to be detected (i.e., to belong to an edge) if the corresponding transition strength $G[p]$ is above a given threshold (see below) and if there is at least one pixel $q \in N_8(p)$ (in the 8-neighborhood of p) such that $0.9G[p] \leq G[q] \leq 1.1G[p]$. The last condition is used to reduce isolated detected pixels due to noise.

Image difference operator
A pixel $p = [i,j]$ is considered to be detected (i.e., to belong to a changed area) simply if the corresponding value of the image difference operator $D_n[p] = \mathrm{Diff}(f_n[p], f_{n-1}[p])$ is above a given threshold (see below).

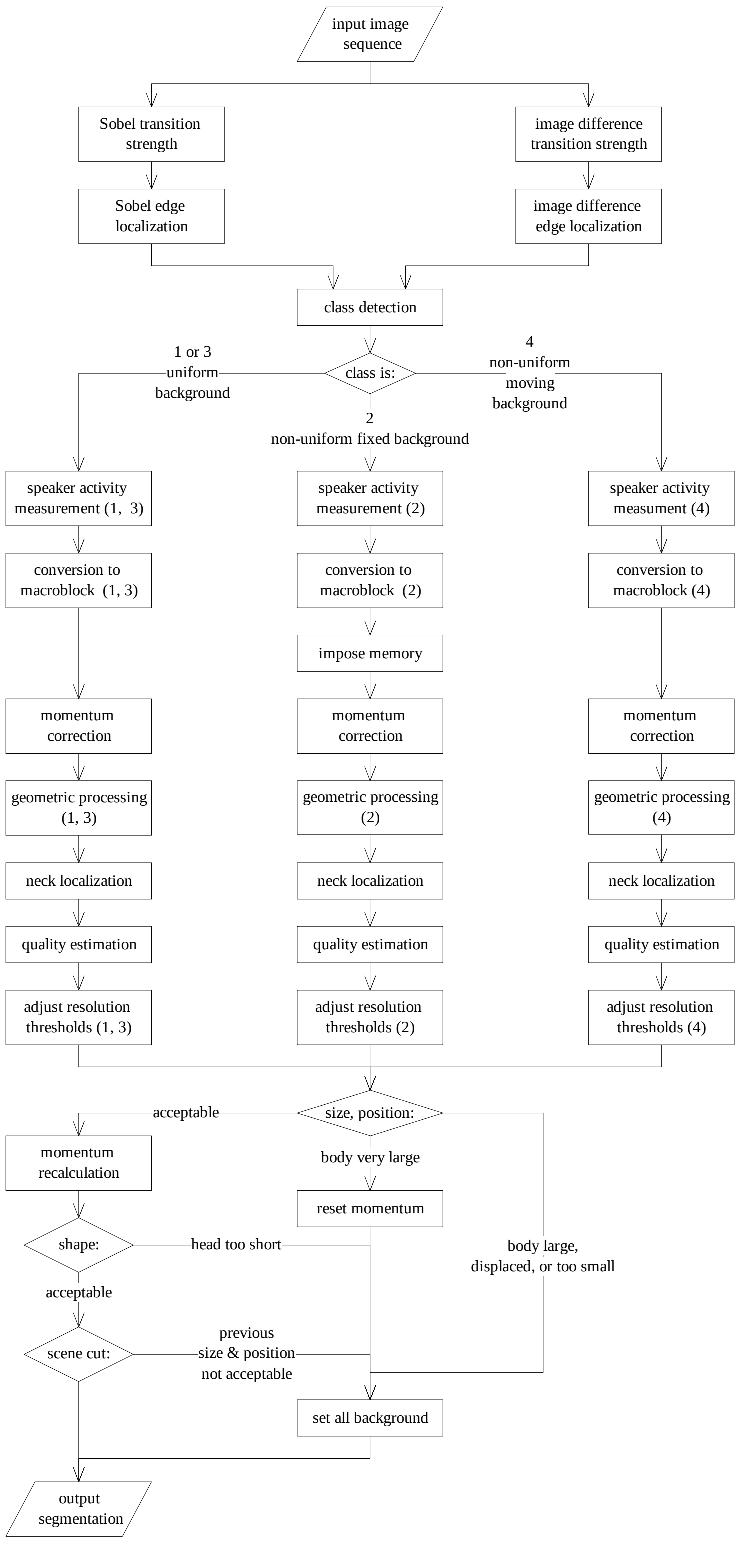

input image sequence
Sobel transition strength
image difference transition strength
Sobel edge localization
image difference edge localization
class detection
class is:
1 or 3 uniform background
2 non-uniform fixed background
4 non-uniform moving background
speaker activity measurement (1, 3)
speaker activity measurement (2)
speaker activity measument (4)
conversion to macroblock (1, 3)
conversion to macroblock (2)
conversion to macroblock (4)
impose memory
momentum correction
geometric processing (1, 3)
geometric processing (2)
geometric processing (4)
neck localization
quality estimation
adjust resolution thresholds (1, 3)
adjust resolution thresholds (2)
adjust resolution thresholds (4)
size, position:
acceptable
body very large
body large, displaced, or too small
momentum recalculation
reset momentum
shape:
head too short
acceptable
scene cut:
previous size & position not acceptable
set all background
output segmentation

Figure 4.5: Segmentation algorithm flow chart. When blocks are identified with class numbers, they perform differently according to the class of the input sequence.

Knowledge-based thresholding

The results of the low-level edge detection operators, obtained by estimating transition strength and then localizing the edges, are afterwards used to measure the activity of each pixel, which will in turn be converted from pixel to MB (16×16 pixels) resolution.

Since the sequences are assumed to consist of a speaker reasonably centered within each image (this is knowledge-based segmentation), the activity measurements should be reinforced at those places where the speaker is more likely to be found in the image. This is done by using variable thresholding during edge localization. The thresholds vary along the image as indicated in Figure 4.6.

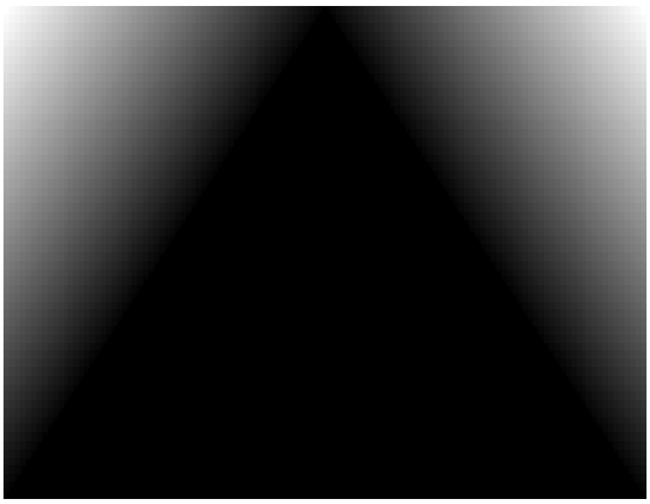


Figure 4.6: Variable threshold patterns consist of a central plateau of constant, low threshold, which grows linearly towards the top corners of the image.

Two variable threshold patterns are used: the first for the Sobel operator (with plateau level 20 and top corners level 64), and the second for the image difference (with plateau level 10 and top corners level 40). In the case of class 4 image sequences, however, the threshold pattern indicated for the image difference is valid only for an image rate of 25 Hz (sequence classes will be defined later). For other image rates the thresholds are adjusted linearly with the inverse of the image rate. For instance, at 5 Hz the plateau level is 30 and the top corners level is 90. This adjustment is necessary in order to avoid detecting too much activity on a moving background, since the range of the movements tends to increase with an increased time between successive images.

Filtering the activity measurements

Before the resolution conversion takes place, the activity at each pixel must be computed. Though the exact method depends on the detected class of the current image, two filtering operators may be used: purge and fill. Both operate on a binary image, e.g., obtained after edge localizing the results of Sobel or image difference:

Purge operator

The purpose of this operator is to eliminate isolated detected pixels. All detected pixels with less than two detected 8-neighbors are cleared. Detected pixels with two detected 8-neighbors are cleared only if not both neighbors are m-connected. The last condition avoids the clearing of pixels belonging to a continuous edge.

Fill operator

This operator sets as detected all undetected pixels which have more than two detected 8-neighbors.

Sometimes the binary results of the described operators are “ored” together. A “ored” pixel is detected if at least one of the corresponding pixels of the two operands being “ored” is detected.

Sequence classes

Of paramount importance for the development of the knowledge-based segmentation technique proposed is the classification of the input videotelephony image sequences into classes. Each of the considered classes, by its own nature, requires different processing. Videotelephone sequences can be classified according to:

1. the uniformity, in terms of texture or complexity, of the background; and to
2. the existence of movement in the background, which may be due to movement of the camera, to movement of the background, to movement of objects seen in the background or to any combination of the three.

Classes 1 and 3

Sequences having a uniform background (it is impossible to distinguish whether the background is fixed, class 1, or has any movement, class 3) such as “Claire” and “Miss America”, which are typical studio sequences.

Class 2

Sequences having non-uniform but fixed background, for instance, “Trevor” and “Salesman”, which are typical in-house videotelephony sequences.

Class 4

Sequences having movement in a non-uniform background, for example, “Carphone”, which has movement in the background due to camera vibration and due to the passing landscape seen through the windows, and “Foreman”, which has movement in the background due only to movements of the hand-held camera.

Segmentation of these classes of sequences has various degrees of difficulty, and for each one there are preferred segmentation techniques:

- Sequences of classes 1 and 3 can be segmented using both 2D and 3D segmentation techniques [99, 160].

- Sequences of class 2 pose more problems to 2D techniques because of the non-uniform background. It is not simple to distinguish the speaker from the complex background. However, if the speaker is moving, and she usually is, 3D techniques may be used to obtain a first approximation to the speaker's position [99, 160]. This information may then be used to restrict the search area for 2D techniques [99].

- Sequences of class 4 are more difficult to segment. For these (typically mobile) sequences special methods must be devised. The main contribution of the proposed algorithm is its ability to cope reasonably with sequences of this class.

Each class is segmented here using different low-level operators. For instance, sequences of classes 1 and 3 can be processed using only 2D operators, which have a very small response in the uniform background, or a combination of 2D and 3D operators, while class 2 sequences are better dealt with using only 3D operators, as they do not respond to a fixed background, however structured it may be.

It is very important for the algorithm to be able to detect the class of the input sequence automatically. Note that class detection is applied to each image in a sequence and thus the class of a sequence may change over time, as will be explained in the next section.

Class detection

Class detection is implemented in a very simple way in this algorithm. First, the two transition strength operators used in the algorithm, Sobel and image difference, are applied to the current image. Each result is then passed through the corresponding edge localization process, which involves the use of a variable threshold. Notice that, for class detection purposes, the use of a variable threshold is not necessary. However, since the results of thresholding Sobel and image difference are used in the steps of the algorithm following class detection, the computational effort is thus reduced. The results are finally analyzed in three zones, shown if Figure 4.7, in which the probability of finding some part of the speaker's head or body is low.

If the activity, measured as the percentage of detected pixels of the edge localized Sobel operator, is above 2% in any of the three zones, the sequence is assumed to have a non-uniform, structured background. Similarly, if the activity of the edge localized image difference operator is above 0.2% in any of the mentioned zones, the sequence is considered to have motion in the background.

Three detection zones of comparable size are used instead of a single one since a very localized movement or structure may be easily missed when a single large zone is used, and using a smaller threshold might lead to erroneously detection of apparent background motion or structure caused by noise.

This type of detection leads to good results for typical videotelephone sequences. It may however fail in images where the speaker moves into one of the analyzed zones. When this happens, some images of a class 1, 2, or 3 sequence can be mistakenly detected as belonging to class 4. However, since the techniques used for the segmentation of class 4 sequences are more robust than for any of the other classes, this will very likely cause little problem.

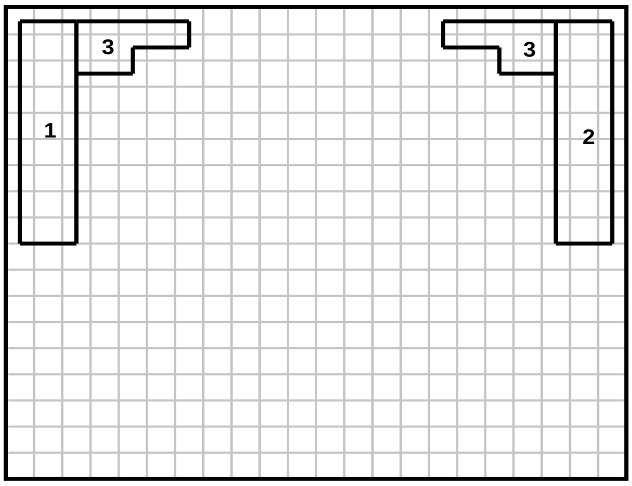


Figure 4.7: Class detection zones in a CIF image (divided into a MB spaced grid).

A common problem that occurs with class detection, if no further processing is done, is the occasional classification of an image, within a class 4 sequence, as belonging to class 2. This usually occurs either when the camera instantly stops between two pan movements or at the apogee of a camera oscillation. For reasons related to the use of memory in class 2 sequences, this occasional detection of a class 2 image may be problematic. Thus, the algorithm was implemented in such a way that after two or more successive class 4 images, a change into class 2 only occurs after at least three successive class 2 images are detected.

Activity measures

The purpose of the low-level operators in the segmentation algorithm is not so much to detect edges as to detect the activity, hopefully the speaker activity, within each image. High activities should indicate the presence of the speaker, while low activities should be found in the background. It is thus clear that the activity measurement process should vary according to the class of the sequence.

Classes 1 and 3

In this case, where the background is uniform, activity may be obtained using solely 2D operators. In this algorithm, however, 3D operators where also used, as a way to improve detection when the speaker moves. Activity is obtained as follows:

1. apply the purge operator to the edge localized image difference;
2. apply the "or" operator to the matrix resulting from the previous step and to the edge localized Sobel; and
3. apply the fill operator to the result of the previous step.

Class 2

In this case the background is structured but it has no motion. Thus, only 3D operators are used. Activity is obtained as follows:

1. apply the purge operator to the edge localized image difference; and
2. apply the fill operator to the result of the previous step.

This procedure eliminates isolated detected pixels but reinforces the differences whenever they are strong.

Class 4

This class cannot be dealt with by using only one type of operator. Using only 2D operators may make it difficult to distinguish the speaker from the structured background, while using only differences may have the same problem whenever the background motion is comparable to that of the speaker. It thus uses both kinds of operators, as class 2 does, though the activity is computed differently:

1. apply the purge operator to the edge localized image difference (the variable threshold applied to the image difference now varies with the image rate, as mentioned before);
2. apply the "or" operator to the matrix resulting from the previous step and to the edge localized Sobel; and
3. apply the purge operator to the result of the previous step.

The last step is to purge, instead of to fill, as in classes 1 and 2, because moving structured backgrounds tend to produce a too dense activity pattern.

Resolution conversion

After class detection and activity measurements, the next step is to convert the pixel level activity to a MB level activity matrix. This conversion is make in two phases:

1. the activity matrix is decimated from pixel to MB resolution, and
2. the resulting MB level activity matrix is filtered to eliminate spurious detections.

Decimation

For class 4 images, decimation is done simply by counting the number of detected pixels in each MB. If the count exceeds a given threshold, the MB is active.

For class 1, 2 and 3 images, the decimation is first performed to block level, using the same method as before, and then to MB level: a MB is active if any of its blocks is detected.

The difference between class 4 images and class 1, 2, and 3 images is that the latter tend to produce more concentrated activities, which might fail to be detected if decimating directly to

MB level, unless the threshold would be set to a lower value. However, reducing the threshold would lead the erroneous detection of regions with more sparsely distributed activities.

Different thresholds are maintained for each class which are also changed adaptively, according to the quality control, so as to match the current sequence characteristics, as will be seen later.

Filtering

The filtering process consists in the application of a series of operators:

1. Any inactive MBs between two active MBs in a line are changed to active. However, this only happens if the MB to be changed is within the knowledge-based mask in Figure 4.8(b).
2. Inactive MBs with three active 4-neighbors are set to active. This filling, however, avoids filling the neck of the speaker, which is the only concave part of the typical speaker's contour: if the inactive MB belongs to the left half of the image, hence probably also to the left half of the speaker, and has its right, top and bottom neighbors active, then it will be filled only if its left neighbor is also active.
3. Segments of lines of at least two active MBs are cleared, since the speaker's silhouette rarely contains such features.
4. Active MBs without any active 4-neighbors are cleared.

Only sequences of classes 1 to 3 undergo this filtering phase. This type of filtering is not convenient for class 4 sequences since these sequences require more sophisticated geometrical methods. For class 2 sequences the filtering is applied only after imposing memory, since often to few MBs are selected without recourse to memory.

Inertia

In typical videotelephone sequences, even in mobile ones, there is usually considerable redundancy in the evolution of the speaker's position: changes are mostly relatively small from one image to the next. This fact can be used to advantage by building some inertia into the segmentation process.

A momentum matrix, with values between 0 and 1, is updated after the segmentation of each image. A matrix element which is close to 1 signals a MB which has been active often in the short past. After resolution conversion to MB level, the momentum matrix is used to correct the matrix of active MBs.

Momentum correction

Momentum correction is a very simple process: all inactive elements in the MB activity matrix with a momentum larger than a threshold are set as active and marked as having been adjusted by this process. The threshold value was chosen empirically as 0.39, which gives good results in practice.

Momentum recalculation

Each element $M_n[i,j]$ of the momentum matrix at image n is updated according to:

$$M_{n+1}[i,j] = 0.7M_n[i,j] + 0.3v_n$$

where $v_n = 1$ if the MB $[i,j]$ is active and has *not* been adjusted during momentum correction, otherwise $v_n = 0$.

The use of the adjustment information avoids the artificial perpetuation of active MBs. If a MB was active in several images, and hence the corresponding momentum is approximately 1, it will remain detected for at most the next two images, unless geometrical processing chooses to clear it.

Memory

For class 2 sequences only the 3D image difference operator is used. Thus, if the speaker does not move enough, not enough MBs will be considered active. The same thing may happen if the image rate is too high. In order to solve this problem, an extension to the inertia process described above was devised: memory.

Memory is imposed during the processing of class 2 images right after the resolution conversion. There is a memory matrix containing the number of image periods (i.e., the time interval) during which a MB should be artificially considered as active after it has been found to belong to the speaker. This memory is dynamically adjusted so as to allow the segmentation to quickly adapt (by reducing memory or even forgetting about the past) if the speaker starts moving enough and to keep a long memory if the speaker does not move enough.

Memory dynamics

The number of MBs set after the resolution conversion is counted N_n and a moving average A_n of it is kept according to:

$$A_n = 0.5A_{n-1} + 0.5N_n$$

If N_n is above 120 (more than the minimum typical speaker size) and A_n is above 80, then the memory matrix is reset, since the speaker's motion is strong enough to dispense with the use of memory.

If A_n is below 100 (less than the minimum typical speaker size) and the number of MBs reduced by more than 10 from the previous image, then MBs having a memory larger than zero and smaller than 10 images have their memory incremented by $10/r$, where r is the ratio between

25 and the current image rate. This will tend to prolong the memory of previously detected MBs when the speaker's motion starts to decrease.

The memory matrix is then decremented, thereby reducing the memory of all MBs detected.

Finally, a memory is attributed to the current active MBs according to the value of A_n:

1. if $A_n < 10$, then memory is set to $100/r$;
2. otherwise, if $A_n < 20$, then memory is set to $80/r$;
3. otherwise, if $A_n < 40$, then memory is set to $50/r$;
4. otherwise, if $A_n < 80$, then memory is set to $30/r$;
5. otherwise, memory is set to $1/r$;

At the final stage the MB activity matrix is substituted by the memory matrix. If an element in the memory matrix has a non-zero memory, the corresponding element in the activity matrix is considered active.

Geometrical processing

As intermediate steps between resolution conversion and quality estimation, several geometric knowledge-based operators are applied in order to build a segmentation matrix from the MB level activity matrix. These operators aim at producing a segmented region which "makes sense", given that it should represent a human speaker in a normal head-and-shoulders framing.

Some of these operators use heuristic knowledge-based masks having highest values where it is more likely to find part of the speaker's body, see Figure 4.8.

Different operators are applied to sequences of different classes.

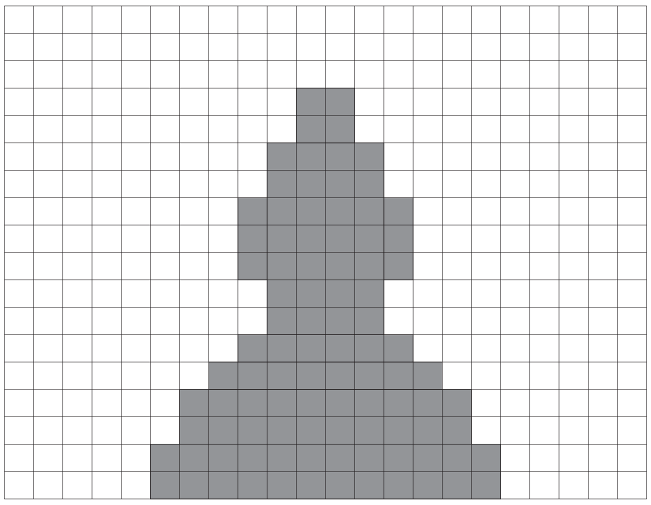
Classes 1 to 3

Four operators are applied in order:

1. All inactive MBs having at least three active 4-neighbors are set to active. The process is continued recursively until no changes are made. The changes, however, are only allowed to happen inside the wide knowledge-based mask in Figure 4.8(b). This operator fills small holes inside the speaker's silhouette.

2. All active MBs having no more than one active 4-neighbor are set to inactive in a recursive manner. This operator clears erroneously detected MBs due to noise or structure in the background.

3. The connected components of the active part of the MB activity matrix are identified. All connected components, except the largest, are cleared. The largest connected component is kept because it is deemed to correspond to the speaker's silhouette. The connected


(a) Narrow mask.

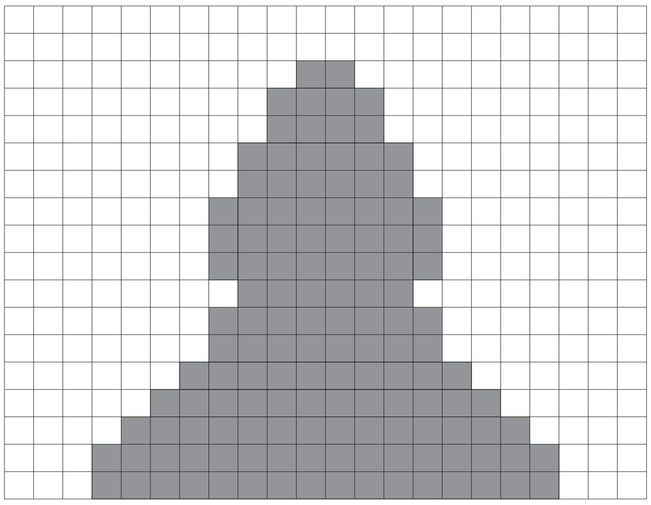

(b) Wide mask.

Figure 4.8: Knowledge-based heuristic masks.

components are defined in terms of a 4-neighborhood structure superimposed to the MB activity matrix.

4. The connected components of the inactive part of he MB activity matrix are identified. They correspond to the background. All background connected components with less than 10 MBs are set to active. Larger connected components are only set to active if they do not include any MB which touches the border of the image. I.e., of the larger background connected components only those which are holes are cleared.

Class 4

The same operators as for classes 1 to 3 are applied, but they are preceded by:

1. The connected components of the active part of the MB activity matrix are identified. All connected components, except the largest, are cleared, but only if these connected components do not have any MBs inside the narrow knowledge-based mask in Figure 4.8(a). The largest connected component is kept because it is deemed to correspond to the speaker's silhouette. Unlike the similar operator which is applied in the case of sequences of classes 1 to 3, more than one connected component may result, since connected components touching the areas where the user may be with a high probability are not cleared. The connected components are again defined in terms of a 4-neighborhood structure superimposed to the MB activity matrix.

2. Inactive MBs with three active 4-neighbors are set to active. This filling, however, avoids filling the neck of the speaker using the same technique as in the filtering part of resolution conversion.

3. Structures with shapes that are unlikely to belong to a real speaker's silhouette are cleared. The operator searches for such structures on the left and right sides of the estimated head position. The shapes considered invalid are those corresponding to large regions connected to a side of the head by a small or highly bent isthmus of active MBs.

4. Inactive MBs between lines (at least two MBs long) of active MBs are set to active. This often joins together the head and body which would otherwise remain separated.

Neck localization

Before quality estimation, the segmentation result is analyzed so as to try to estimate the position of the line separating head and body, the neck line. The algorithm developed for this purpose uses geometrical, knowledge-based reasonings. It also uses feedback from the estimates obtained in previous images in order to avoid large changes from image to image caused by errors in the segmentation results.

The neck position estimation algorithm tries to estimate the head width and then searches from top to bottom for the first pair of MB lines whose width exceeds 1.7 times the head width. These lines will likely correspond to the speaker's shoulders. Since the speaker's chin often prolongs somewhat below the line of the shoulders, the first of these shoulder lines is deemed to belong to the head and the second to the body.

The estimated shoulder line is only allowed to change by one MB line per image, so as to average out errors during its estimation.

The active MBs are then classified as either belonging to the body (those below the shoulder line) or to the head. If the shoulder line was not found, no such distinction is made.

Knowledge-based quality estimation

The quality estimation algorithm tries to ascertain whether the active MBs, which identify those parts of the image (in MB resolution) where the speaker is believed to be, correspond to an appropriately sized and positioned speaker in a typical head-and-shoulders videotelephone scene. It first counts the total number T of detected MBs and the total number T_K of detected MBs inside the narrow knowledge-based mask (see Figure 4.8(a)). Then:

1. if $T > 300$, the speaker size is considered very large, and the segmentation is discarded;
2. otherwise, if $T > 200$, the speaker size is considered large,[11] and hence the segmentation is discarded and the activity resolution conversion threshold is increased (this change affects only the subsequent image);
3. otherwise, if $T_K < 0.5T$ or if $T < 60$, the speaker position is considered displaced or its size too small, and thus the segmentation is discarded and the activity resolution conversion threshold is reduced (affects the subsequent image);
4. otherwise, the quality is acceptable, hence the segmentation is accepted and no threshold adjustments are done.

The quality estimation described so far was developed mainly for segmentation quality control purposes, though it does reject low quality segmentation results.

A different quality assessment procedure is also used, although this time only for the acceptance/rejection process. It uses geometric knowledge-based criteria to decide whether the segmentation result, in term of the attained head and body shape, is acceptable. It is based on the ratio between head height and visible body height in typical videotelephone scenes. If the body is wider than about 90% of the image width and the head height is smaller than 56% of the visible body height, then the shape is declared disproportionate, the segmentation is deemed invalid and its results are rejected.

Whenever quality estimation leads to rejection, the segmentation matrix is filled with the value representing background everywhere. This situation often happens during the first few images after a scene cut or a large pan or zoom. Such rejection of segmentation results, in the framework of first-generation video coding techniques, is not too problematic, unless it occurs often: it just means that no quality discrimination will be made during a rejection. Since quality tends to increase and decrease slowly in time when temporal prediction is used in typical first-generation codecs, the subjective impact of rejection of segmentation results is small.

[11] Always considering CIF format. The number of MBs in CIF is 396, hence 200 corresponds to activity in a little above half the image area.

When the segmentation results for a given image are considered unacceptable (region too large, too small, or displaced) and hence rejected, the segmentation of the next image is also rejected. Since often rejection stems from scene cuts or large changes, this method allows for some recovery time before segmentation is accepted. The rationale being that it is better to have no segmentation than to have a bad segmentation.

Quality control

The segmentation algorithm uses delayed quality control: the estimated quality of the current segmentation affects only the segmentation of future images. This method has the advantage that is keeps the computational requirements of the algorithm small.

Quality control is done in two ways:

1. If the estimated segmentation quality leads to considering the speaker size very large, the inertia matrix is reset, thus eliminating partly eliminating influence of the past into the future. This makes sense because most rejections of segmentation occurring for this reason take place at scene changes, where the sequence characteristics vary considerably and changes from one image to the next are very large. In the case of class 2 sequences, however, memory is not reset, as it would thus lead to the slow rebuilding of the segmentation based solely on 3D operators. The memory is reset only when the estimated class changes.

2. The resolution conversion thresholds are changed according to the estimated quality in a way which depends on the current image class:

   Classes 1 and 3
   The threshold is initially 13 pixels (about 20% of the pixels of a block). When the speaker size is deemed very large, the threshold is increased by 6 (but limited to a maximum of 26, or 41%). When the speaker size is considered large, the threshold is increased by 3 (again limited to a maximum of 26, or 41%). When the speaker size is too small or its position displaced, the threshold is reduced by 6 (but limited to a minimum value of 13, or 20%). Nothing changes otherwise.

   Class 2
   The threshold is initially 6 pixels (about 9% of the pixels of a block). When the speaker size is deemed very large, the threshold is increased by 6 (but limited to a maximum of 26, or 41%). When the speaker size is considered large, the threshold is increased by 3 (again limited to a maximum of 26, or 41%). When the speaker size is too small or its position displaced, the threshold is reduced by 6 (but limited to a minimum value of 1, or 2%). Nothing changes otherwise.

   Class 4
   The threshold is initially 110 pixels (about 43% of the pixels of a MB). When the speaker size is deemed very large, the threshold is set to its initial value of 110 pixels. When the speaker size is considered large, the threshold is increased by 20 (but limited to a maximum of 173, or 68%). When the speaker size is too small or its

position displaced, the threshold is reduced by 15 (but limited to a minimum value of 95, or 37%). When the speaker size and position are accepted, the threshold is reduced by 15, but only if it is larger than 125.

4.4.3 Computational effort

A thorough computational effort analysis of the algorithm has not been done. However, compiler optimized[12] code segmenting the "Foreman" sequence (a 25 Hz sequence consisting mostly of class 4 images) in a Sun Sparc 10 spent about 40% of its time in the purge operator, which involves just comparisons and memory addressing operations. Also, the operation intensive Sobel, requiring (with non-optimized code) $11N$ sums and $4N$ multiplications by 2 (shift lefts), where N is the total number of pixels, contributed to about 10% of the time. Hence, the rest of the processing, though more involved from an algorithmic point of view, did not account but to about 50% of the time. These figures hint that the code efficiency can be increased by optimization and/or hardware implementation of the bit-level operators.

4.4.4 Results

Several standard videotelephone test sequences were used:

"Foreman" (25 Hz)
Most of the time a class 4 sequence.

"Carphone" (25 Hz)
Most of the time a class 4 sequence.

"Claire" (10 Hz)
A typical class 1 sequence.

"Miss America" (10 Hz)
Also a typical class 1 sequence.

"Trevor" (second shot only, 10 Hz)
A typical class 2 sequence.

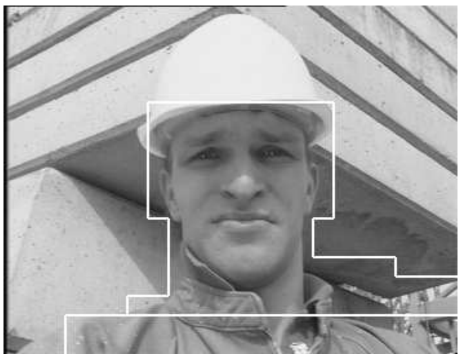
"Salesman" (30 Hz)
Also a class 2 sequence.

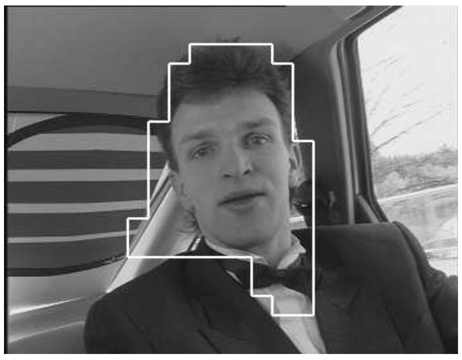
The "Claire", "Miss America" and "Trevor" sequences used are all part of a single 10 Hz image rate sequence named "VTPH" (for videotelephone).

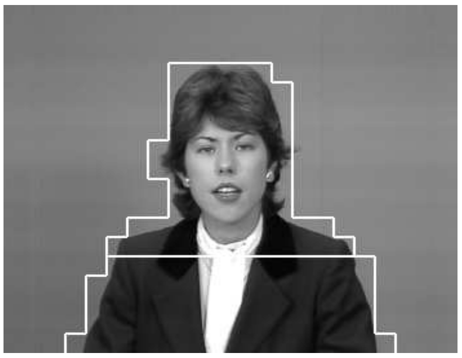
The results obtained were good, as can be seen in the representative images in Figure 4.9. Comparable results were obtained repeating the above experiments for the same sequences downsampled to 5 Hz image rate (downsampling by image dropping). Notice that the first

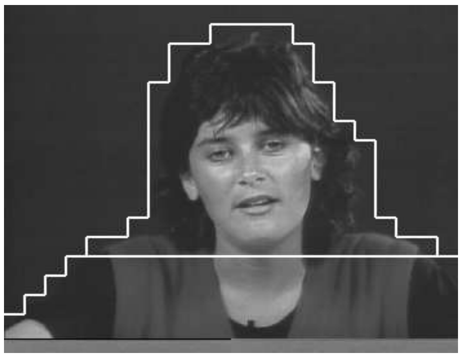
[12]Using a gcc 2.* compiler.

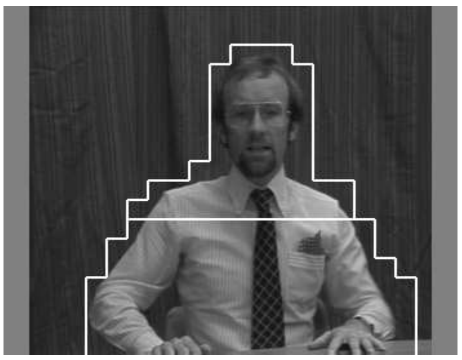
image of the sequences is not segmented by the algorithm, since there is no past image to be used by the 3D operators.


(a) "Foreman" image 2


(b) "Carphone" image 2


(c) "VTPH" image 2 ("Claire" image 6)


(d) "VTPH" image 126 ("Miss America" image 125)


(e) "VTPH" image 143 ("Trevor" image 96)

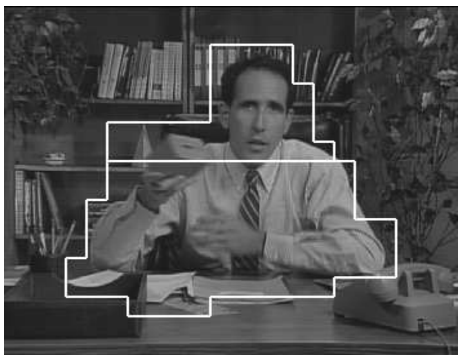

(f) "Salesman" image 10

Figure 4.9: Knowledge-based segmentation results for several videotelephone sequences.

4.4.5 Conclusions

A new MB level knowledge-based segmentation algorithm for videotelephone sequences has been developed which is able to cope with a wide range of sequences. It was developed has an evolution of the algorithms in [160, 166]. The algorithm demands relatively low computational power, since most of the processing is done at MB level, which has 256 less elements than the original images.

The algorithm is a good candidate for integration in very low bitrate first-generation video coders. The estimation of the speaker's position can be used to improve the subjective quality of the encoded sequence by increasing the objective quality of the image at the speaker's face and body at the expense of a reduced objective quality in the background (which is subjectively less important).

A classification of videotelephony sequences has been proposed. According to it, sequences can be classified as belonging to classes 1 and 3—uniform background—, class 2—non-uniform but fixed background—, and class 4—non-uniform moving background.

4.5 RSST segmentation algorithms

This section presents a new image segmentation algorithm which is based on the RSST concept [134] and on split & merge (with elimination of small regions) [72]. Unlike [72], the regions are merged by order of similarity (or uniformity of the result), and the elimination of small regions is but an intermediate step of the process. Unlike [134], the problem of small regions is dealt with. Also, a mathematical morphology image simplification technique is proposed for application before segmentation, which is typically used in Watershed algorithms such as the one in Sesame [30]. The resulting algorithm has a performance comparable to the RSST algorithm in [194], although with a slightly less elegant formulation. This algorithm is the result of collective work, and was first proposed in [32, 33].

The new algorithm presented is compared with the RSST algorithm proposed in [194], and with its extension using an affine, instead of flat, region model. The performance of these other algorithms is also assessed when the image simplification pre-processing and the split phase of the new algorithm are added.

The outline of the new algorithm is as follows. Images are first simplified using a mathematical morphology operator, which attempts to eliminate less perceptually relevant details. The simplified image is then split according to a QPT and the resulting regions are then merged using one of three criteria: merge, elimination of small regions and control of the number of regions.

The split phase generally produces an over-segmented image, its interest stemming from the reduction in the total number of regions, which leads to a reduced computational effort, especially when compared to the typical region merging solutions, where each pixel is initially considered as an individual region.

The merge criterion merges successively the most similar adjacent regions resulting from the split step, thus removing the false boundaries introduced by the QPT structure used.

The elimination of small regions criterion removes a large number of the small, often less relevant, regions which typically result when the merge criterion starts to fail. If not eliminated, small regions lead frequently to an erroneous final segmentation, since they have a large contrast relative to their surroundings. Small regions are eliminated by merging them to their most similar neighboring regions.

The control of the number of regions criterion is similar to the merge step. However, this criterion fails when a certain final number of regions is attained (or, alternatively, when a threshold of region dissimilarity is exceeded).

At each step the algorithm produces segmented images with one less region. Hence, it can be seen as originating an image hierarchy with increasing simplification levels. There are two sources of image simplification in the algorithm. Firstly, simplification in a pre-processing step, which eliminates details which are deemed irrelevant before applying the three other steps of the algorithm: merging, eliminating small regions, and controlling the number of regions. Secondly, the segmentation itself can be seen as successively simplifying the image, by approximating it with the same region model (in the case the flat region model) over larger and larger regions.

4.5.1 Pre-processing

Since more often than not images are meant to be appreciated by humans, the economy of representation mandates that details which are perceptually less relevant from the HVS point of view should be eliminated as much as possible. This process is labeled simplification. It is done in part by the segmentation algorithm itself, but it may be advantageous to simplify the image before performing segmentation proper. The purpose of the pre-processing stage is thus to simplify the original image in order to eliminate at least part of the less relevant information. Incidentally, this may also reduce the computational load of the subsequent segmentation steps.

A typical method of simplifying an image is by using low pass filters with an appropriate region of support (typically a finite window, so that FIR filters can be used). However, these filters tend to attenuate the color transitions corresponding to physical edges. Some may even modify their position. These effects can have a very negative impact on the segmentation results, especially in terms of boundary localization.

Tools without the aforementioned problems have been proposed in [176], namely the opening-closing by reconstruction operator $\varphi\gamma^{(rec)(n)}(\cdot)$

$$\varphi\gamma^{(rec)(n)}(I) = \varphi^{(rec)(n)}(\gamma^{(rec)(n)}(I))$$

and the closing-opening by reconstruction operator $\gamma\varphi^{(rec)(n)}(\cdot)$

$$\gamma\varphi^{(rec)(n)}(I) = \gamma^{(rec)(n)}(\varphi^{(rec)(n)}(I)),$$

both of which produce comparable (if different in general) results. These mathematical morphology operators [181, 180] are based on geodesic erosion and dilation operators as specified in [34] (see also [33] for details).

These opening-closing and closing-opening operators by reconstruction eliminate both bright and dark details without corrupting thin edges and without affecting edge positioning. Figure 4.10 exemplifies its application to image 50 of the "Table Tennis" sequence. In this work, simplification was always performed by first converting the sequence to $R'G'B'$ color space and then simplifying each color component of the image separately. This is arguably not an optimal solution, since the simplifications are thus introduced independently in each component. However, these non-linear operators do not translate easily into a non-scalar world and, besides, the results obtained in practice seem to be acceptable.

In simplification there is a trade-off between computational load and final segmentation quality. Increasing too much the simplification reduces the computational load, but can also decrease the final segmentation quality. On the other hand, an adequate simplification degree may in fact improve the segmentation results by reducing the effect of undesirable image features, such as noise and less relevant details.

4.5.2 Segmentation algorithm

The segmentation algorithm consists of two phases: the split phase and the merge phase.

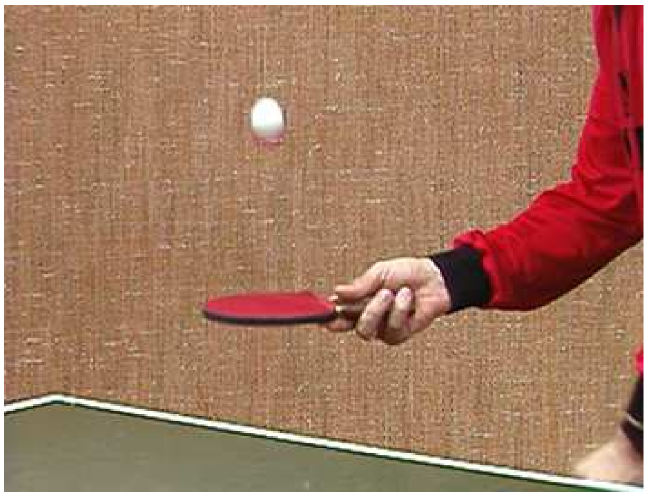
Split

During the split phase, the image is recursively split into smaller regions according to a QPT structure. At each step a region is analyzed and, if it is considered inhomogeneous, it is split into four. The adopted homogeneity criteria was the dynamic range, since the use of the variance or the use of the similarity between the average colors of the four sub-regions both were found in [33] to lead to worst results. Hence, for an image $i[\cdot]$ to segment, a rectangular region $\mathbf{R}$ is split if $\max_{v\in\mathbf{R}} i[v] - \min_{v\in\mathbf{R}} i[v] \geq t_s$, where t_s is the split threshold.

The main purpose of this step is to reduce the computational load of the merge step and hence of the overall algorithm: the smaller the initial number of regions for the merging steps, the less memory is used. There is a compromise between computational effort and segmentation quality: the higher the threshold, the lower the number of regions and hence computational costs, but the higher the inhomogeneity of the resulting regions. In order to avoid compromising segmentation quality, the split threshold is typically set to a low value, so that after the split step the regions have a nearly constant grey level. In this work a threshold of $t_s = 12$ has been chosen empirically so as to lead to acceptable results for a wide range of test sequences.

Merge

Merging is actually based on three criteria, which are tried in sequence at each step of the algorithm. First, an attempt to merge by order of region similarity is performed: the two most similar adjacent regions are merged, but only if their color distance is small enough. If the previous criterion fails, the smallest region, if considered small enough, is merged to its most similar neighboring region. If that also fails, then the two most similar adjacent regions are


(a) Original.

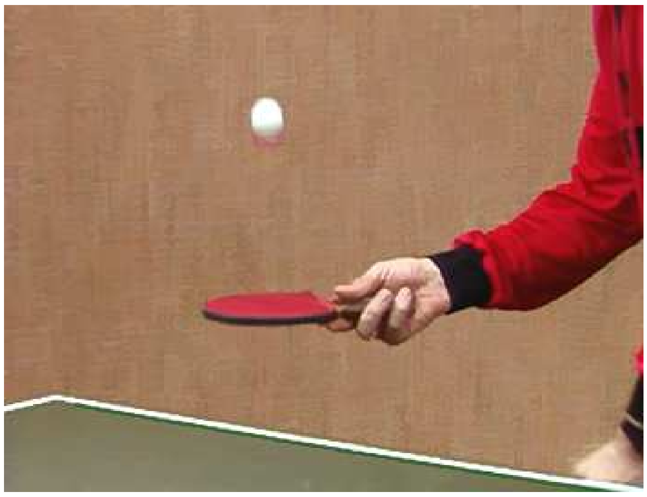

(b) Simplification of each $R'G'B'$ component with the opening-closing by reconstruction operator using a 3×3 structuring element.

Figure 4.10: "Table Tennis" image 50.

merged, but now only if the number of regions in the partition still exceeds the number of required regions. These three criteria will be detailed in the following.

Merge by similarity

This is the first criterion of the merge phase of the algorithm. Merging, in this case, is based on the distance between the average colors of the pairs of adjacent regions, as in [134]. In [33] two different homogeneity criteria were tested, namely the dynamic range and the variance of the color on the union of the two regions. Both alternatives were discarded since they consistently led to worst results. As in [134], regions are merged in non-decreasing order of similarity. In this case, however, the merge takes place only if the color distance of the two regions to be merged is below the so-called merge threshold t_m.

Elimination of small regions

This is the second criterion of the merge phase of the algorithm. Many small, perceptually less relevant, regions tend to remain after the merge step. These regions are usually contrasted with their surroundings, and hence are not easily merged into the larger, more perceptually relevant, neighbors by the merge by similarity criterion of the previous section. These small regions, if not dealt with adequately, usually lead to erroneous final results. It is common, if small regions are not eliminated, that the majority of the final regions are small and more often than not irrelevant, while the most perceptually relevant regions are merged together or into the background.

In this step, any region not larger than 0.004% of the total image area (4 pixels in a CIF image) is eliminated. Afterwards, regions smaller than 0.02% of the total image area (20 pixels in a CIF image) are eliminated by increasing size, but only while the overall area of eliminated regions is not larger than 10% of the total image area. In case of size ties, the similarity between the small regions and any of their neighbors is used to decide which of the small regions to eliminate. The thresholds were chosen empirically so as to lead to reasonable results for a wide variety of test images. This algorithm outperforms the one in [134], since the problem of small regions was not considered there. An evolution of the algorithm in [134], presented in [194], minimizes contributions to the global approximation error, and yields results which are generally better. It has the additional advantage that it does not recur to empirical thresholds and ad-hoc elimination of small regions.

The elimination, in the algorithm proposed here, is always done by merging the small regions to the most similar adjacent region. When merging a small region into a larger neighbor, the algorithm does not change the larger neighbor parameters (e.g., grey level average, variance, etc.). Thus, small regions do not "pollute" the average color of the larger regions they are merged to.

Control of the number of regions

The objective of the third criterion of the merge phase is to control the segmentation result in terms of the final number of regions. It is equal to the merge step, though now the process stops when the required number of regions is attained. Since this step successively produces segmented images with a decreasing number of significant regions, it can be seen as originating an image hierarchy with increasing simplification levels.

In comparison to the algorithm in [194], where a maximum global approximation error can be used to decide when to stop the algorithm (using a single threshold), either the final number of regions or the maximum color distance of the regions have to be used, both of which have a much less clear relation to the global segmentation quality.

4.5.3 Results and discussion

The algorithm proposed in the previous sections is compared with the RSST algorithm presented in [194], which uses a flat region model, and with the RSST algorithm in [194] updated to use the affine region model. These algorithms will be referred to as new RSST, flat RSST, and affine RSST, respectively, even though the last two are actually the same algorithm using different region models. The last two algorithms were also tried with an initial split phase, so as to observe the changes in results in terms of quality and computational efficiency.

The next section describes the experimental conditions, and the ones following it present and discuss the results. First the new RSST algorithm will be discussed. Then, it will be compared to the flat RSST algorithm. Finally, the strengths and weaknesses of the affine region model will be assessed by comparing the flat and affine RSST algorithms. These comparisons all assume algorithms without a split phase. The advantages of the split phase will be dealt with in a separate section.

Experimental conditions

The new RSST segmentation algorithm was run always with $t_s = 12$ (when a split phase was used) and $t_m = 10$. These specific thresholds were chosen empirically, since it was observed that they led to reasonable results for a wide range of images.

The test images are the first images of some of the sequences described in Appendix A. All experiments were run until a final number of 25 regions was attained, with a few exceptions where, in order to show some special feature of an algorithm, a smaller number of regions was used. Since a fixed number of regions was used, the results should be compared according to the overall uniformity, or approximation quality. Notice that the inverse procedure might be used, i.e., establishing a final quality and comparing the number of regions required. However, since the new RSST algorithm does not aim explicitly at maximizing quality, this would complicate the stopping condition of the algorithm needlessly. In practice, on the other hand, both objectives (a given quality with the smallest possible number of regions, or the highest possible quality for a given number of regions) may make sense depending on the application.

The test images are all either CIF of QCIF (Quarter-CIF) in terms of spatial resolution. Regardless of the size of the image to segment, however, and before the split phase (when it exists), the image has been unconditionally split into 32×32 blocks, instead of starting with the whole image.[13] In those cases where the split phase is not used, the image is initially divided into blocks of a given size, typically with a single pixel.

When simplification is used, a 3×3 ($n = 1$) structuring element (see Figure 4.10) was chosen empirically, since it produced good results for a wide range of images.

The segmentation results are presented in the form of a picture where the region borders are drawn in black and the region interiors are drawn according to the estimated parameters of the parametric model used (i.e., either the flat region model, for the new and flat RSST, or the affine region model, for the affine RSST) or with the texture of the original image.

The new RSST algorithm

Image simplification and elimination of small regions

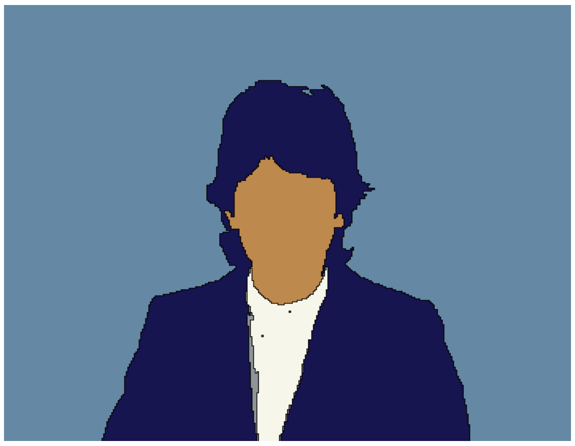
Figure 4.11 shows the same test image segmented with the new RSST algorithm. The result using both pre-processing (image simplification) and elimination of small regions can be seen to be acceptable. However, neither elimination of small regions nor image simplification by themselves yield acceptable results (remember that the four results have exactly 25 regions). Image simplification, on the one hand, has a limited power in removing detail: if a window larger than 3×3 is used, and even though the morphological filter used tends to preserve edges, the boundary positioning suffers. On the other hand, small region removal, being based solely on region size, can hardly solve the problem in a totally generic way. The combination of the two, however, yields an algorithm which is much more robust, even though somewhat inferior to the flat RSST, as will be seen in the next sections.

Notice that, without either simplification or removal of small regions, a considerable proportion of the total number of regions is rather small and perceptually irrelevant. They exist as separate regions because they have a strong contrast to their surroundings. None of the RSST algorithms (new, flat, and affine) fully solves the problem of selecting regions according to their perceptual relevance. However, it will be seen that flat and affine RSST partially solve the problem by implicitly assuming larger regions to be more relevant than smaller ones, instead of relying solely on contrast, as does the new RSST when small regions are not removed.

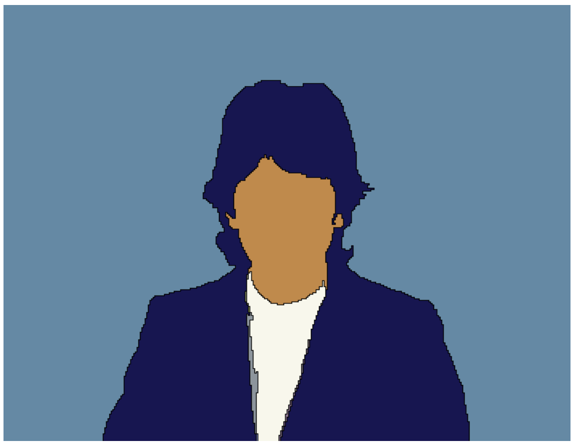
Also notice that, without removal of small regions and without the split phase, the new RSST algorithm is essentially the same as the RSST algorithm in [134].

The results presented in the following sections for the new RSST algorithm assume that image simplification and removal of small regions are indeed performed.

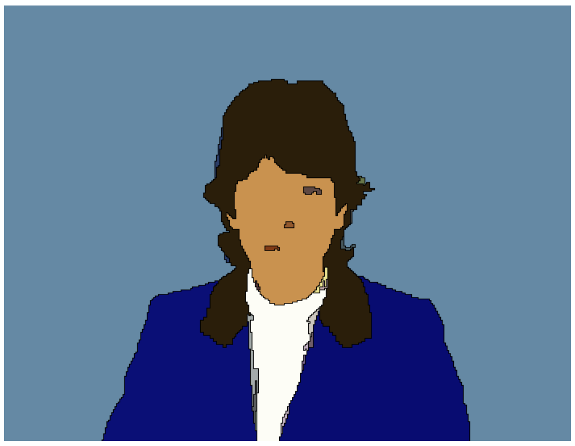
[13]For images whose size is not a multiple of 32, which is the case of the QCIF test images, the top-leftmost block is always 32×32, which means the blocks along the bottom and right border of the image may be rectangles.



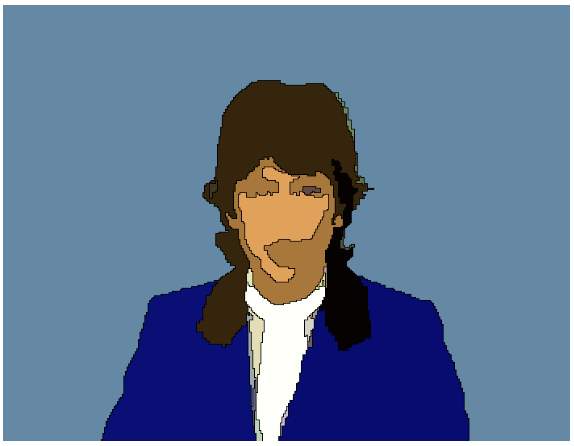
(a) Without small region removal, without simplification.



(b) Without small region removal, with simplification.



(c) With small region removal, without simplification.

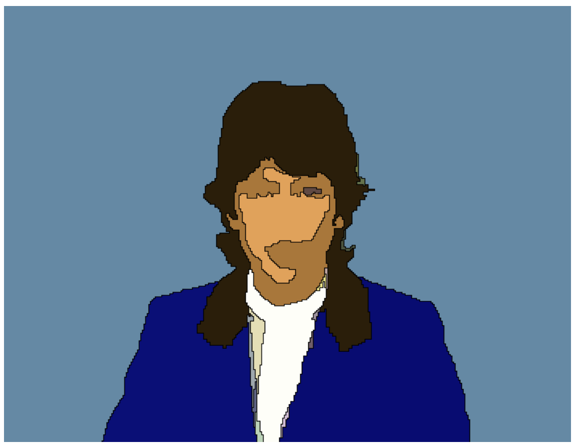


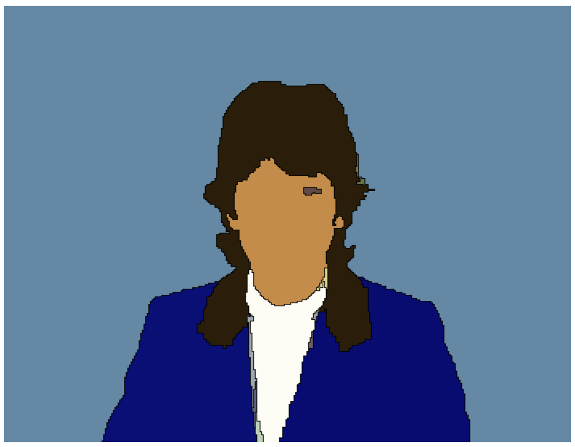
(d) With small region removal, with simplification.

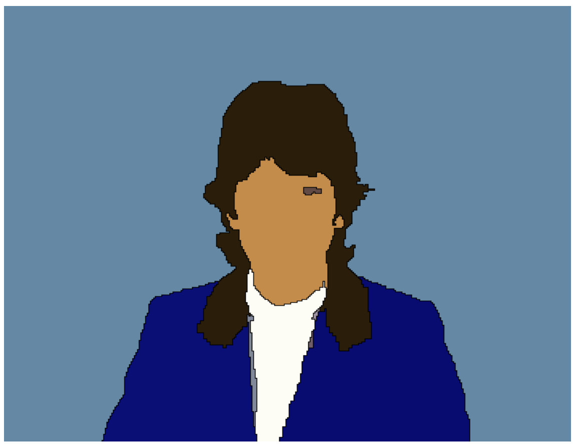
Figure 4.11: "Claire" image 0: segmentation into 25 regions using the new RSST algorithm (without the split phase).

Control of the number of regions

This step controls the final number of segmentation regions, hence implicitly controlling the final level of detail of the segmentation. Figure 4.12 shows the results of the new RSST algorithm with 20 to 5 regions. It can be clearly seen that these segmentations form a hierarchy of decreasing detail.


(a) 20 regions.


(b) 15 regions.


(c) 10 regions.

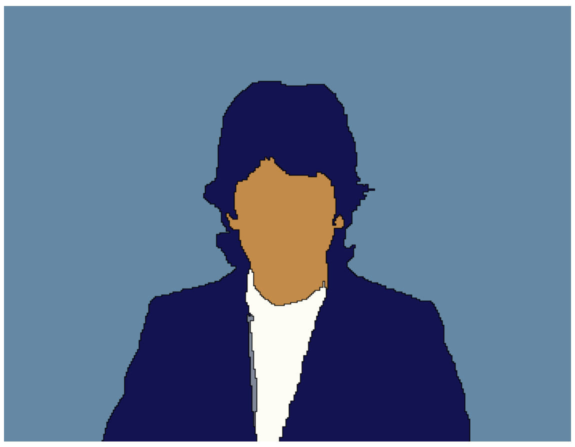

(d) 5 regions.

Figure 4.12: "Claire" image 0: segmentation into 20 to 5 regions using the new RSST algorithm (without the split phase).

The flat RSST algorithm

Image simplification

Image simplification, as a pre-processing step, does not yield significant improvements in the case of the flat RSST algorithm. An example can be found in Figure 4.13, where the "Carphone" sequence is segmented with and without simplification. This insensitiveness to the effects of simplification are due to the fact that, in algorithms attempting to reduce the global approximation error, as is the case of the flat RSST algorithm, small regions tend to be merged sooner. Hence, segmentation itself can be seen as a simplification process, which eliminates successively the details of the image, and a simplification pre-processing step is redundant.

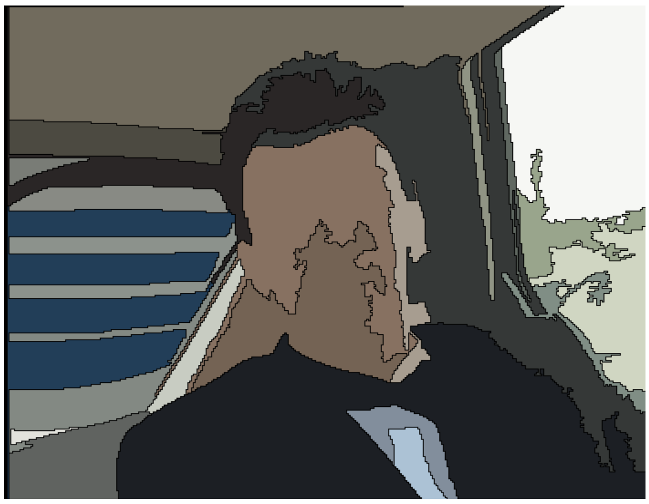
Since simplification, for algorithms attempting to minimize the global approximation error, is performed by the algorithm itself, the results presented in the following for both the flat and affine RSST algorithms assume that no image simplifying pre-processing is done prior to segmentation.

Comparison with the new RSST algorithm

Figure 4.14 shows segmentation results of four different test images using the new and the flat RSST algorithms. The same segmentation results are shown in Figure 4.15 superimposed on the original images, so that the boundary accuracy can be assessed more easily.

For all but the "Flower Garden" sequence the results attained are comparable, if slightly better in the case of the flat RSST. Flat RSST seems to yield regions which are globally more significant, even though it tends to eliminate small details which are semantically relevant but which do not contribute much to the global error. It is the case of the eyes of "Claire" and the shades in the ball in "Table Tennis". However, it must be remembered that these details are taken into account in the new RSST algorithm not because of their semantical relevance, but simply because they are contrasted to their surroundings. Also, flat RSST tends to produce more false contours, i.e., region boundaries neither corresponding to transitions in the image nor to physical edges in the scene. However, these false contours stem not so much from the algorithm as from the model used. As will be seen in the next section, more sophisticated region models tend to solve the false contour problem. Other methods for removing false contours can be found in [194].

In the case of the "Flower Garden" sequence the flat RSST algorithm greatly outperforms the new RSST algorithm. This is due to the fact that the new RSST algorithm relies on a somewhat ad-hoc method for removing small regions, which are very abundant in such a textured image. The parameters (thresholds) used in the elimination of small region criterion were chosen empirically so as to yield acceptable results in general. The fact that there are sequences such as "Flower Garden" where the new RSST algorithm fails (unless the thresholds are adjusted in a rather ad-hoc way) and the fact that the flat RSST yields acceptable results also for these problematic sequences, proves that the flat RSST algorithm is indeed more generic than the new RSST algorithm.


(a) Without simplification

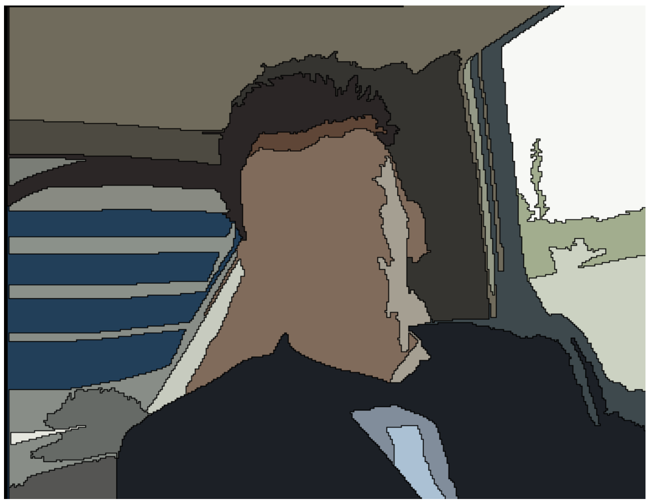

(b) With simplification

Figure 4.13: “Carphone” image 0: segmentation into 25 regions using the flat RSST algorithm.

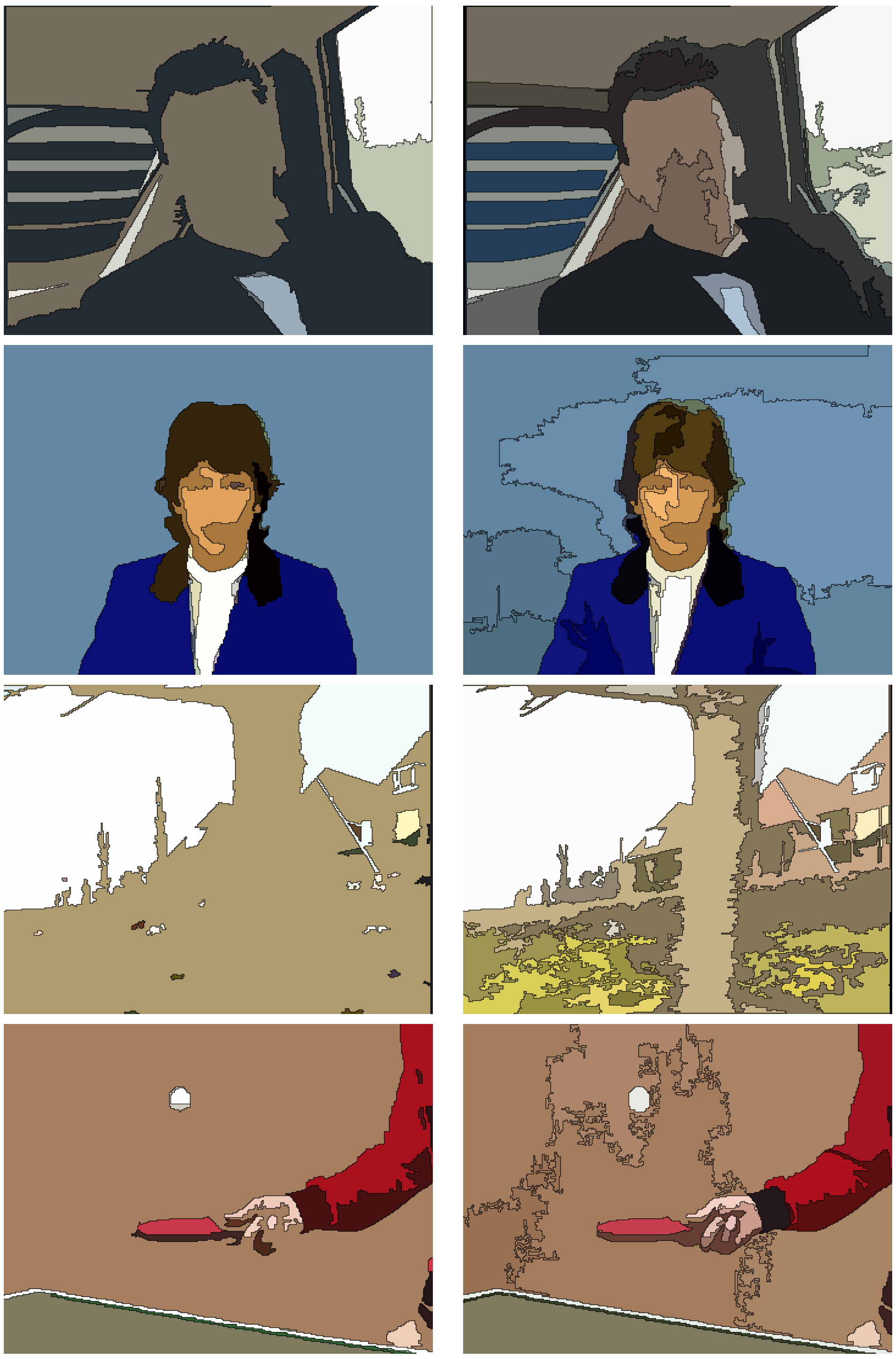


Figure 4.14: "Carphone", "Claire", "Flower Garden", and "Table Tennis" (image 0): segmentation into 25 regions using the new RSST algorithm (left) and the flat RSST algorithm (right). No split phase was used.

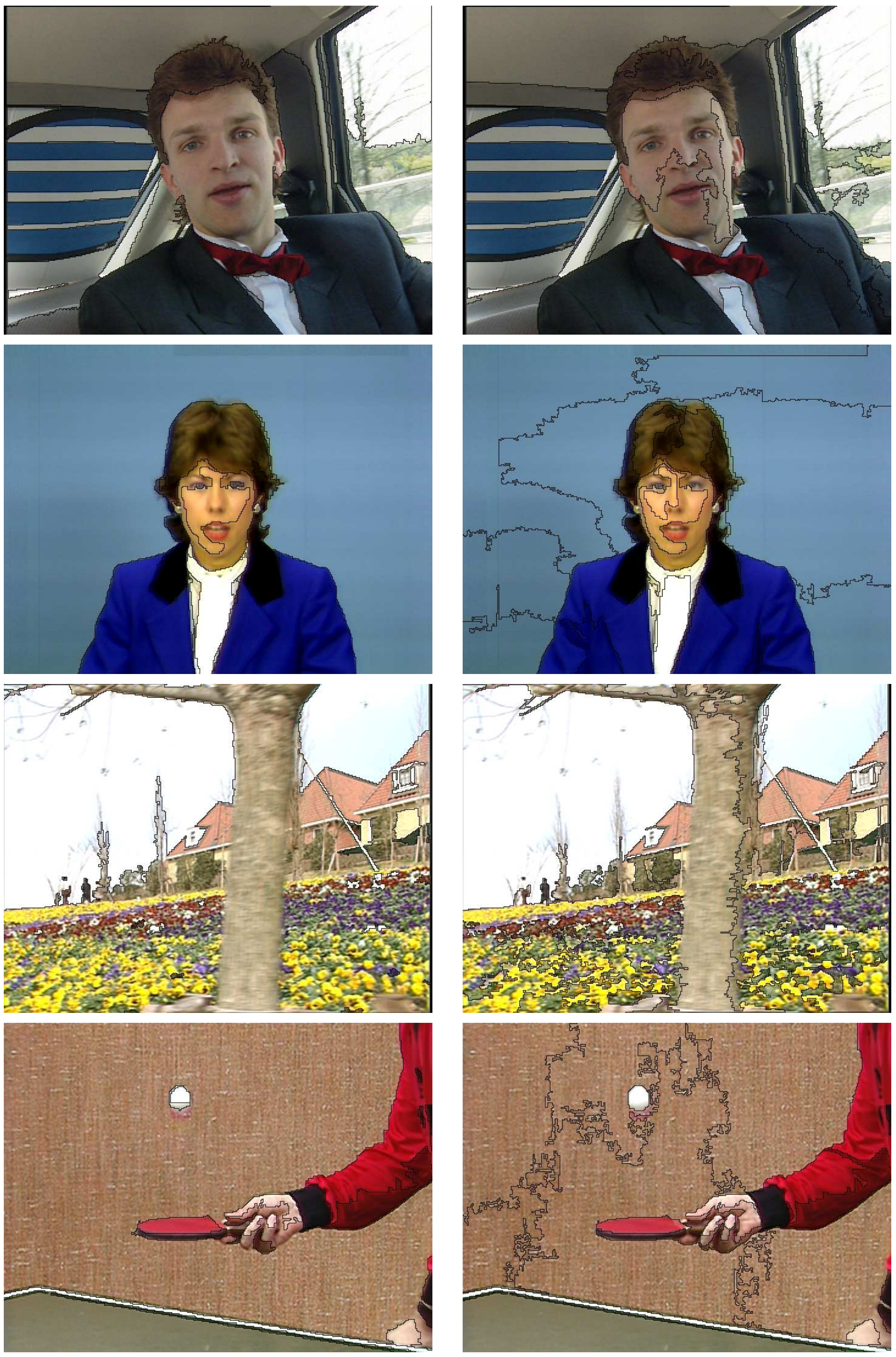


Figure 4.15: “Carphone”, “Claire”, “Flower Garden”, and “Table Tennis” (image 0): segmentation into 25 regions using the new RSST algorithm (left) and the flat RSST algorithm (right). No split phase was used. Boundaries superimposed on the original.

Using an affine model

The RSST algorithm proposed in [194], which uses the flat region model, was adapted to use the affine region model. Figure 4.16 shows the results of segmenting a few test images with such an algorithm.

It must be noticed here that the images segmented in Figure 4.16 are QCIF size. This is due to the fact that the region model parameters occupy considerable more memory for the affine region model than for the flat region model, which rendered segmentation of CIF images impractical whenever a split phase was not used. However, the implementation of the algorithms did not attempt to minimize the required memory. With a suitably optimized implementation, CIF images and beyond might well be within the powers of a normal PC.

The results show considerable boundary raggedness. The reason for this raggedness is that the affine model adjusts with error zero to any region with up to three pixels (in 2D). Hence, when starting from one-pixel regions, the first merges are essentially random, since all merges contribute zero to the global error, and the algorithm does not specify what to do in case of ties. Despite this fact, it can be seen that the model is powerful enough to represent shaded areas, for instance in the building in the background of "Foreman" or in the arm of "Table Tennis".

Comparison with the flat RSST algorithm

Figure 4.17 shows the comparison of the affine model to the flat model (both in the framework of a global error minimization RSST, i.e., flat and affine RSST), when the image is initially split into 2×2 blocks. An area of four was chosen to avoid the problem described above of over adjustment of the model to the data inside very small regions, which leads to boundary raggedness. Since the initial number of regions is divided by four, the results are presented for CIF sized sequences, which have the same memory requirements. The same segmentation results are shown in Figure 4.18 superimposed on the original images, so that the boundary accuracy can be assessed more easily.

The use of initial 2×2 blocks leads to inevitable loss of resolution in boundary positioning. However, when the flat region model is used in the same circumstances, the affine region model leads to an economy of representation which is not possible with the flat model. See for instance the red sleeve or the ball in "Table Tennis", which are now well represented with a smaller number of regions. Also, the use of more powerful region models leads to less false contours, as can be seen in the background of "Table Tennis" and "Claire".

The use of 2×2 blocks has other drawbacks, besides lost resolution in boundary positions. A block may fall in a transition which is two pixels thick, thus creating a new region which may never again be merged to either side. Both these problems might be solved by an improved algorithm where the resulting regions might be split wherever necessary and then re-merged, or simply by post-processing the boundaries pixel by pixel, deciding whether they should belong or not to any of the adjacent regions [146].

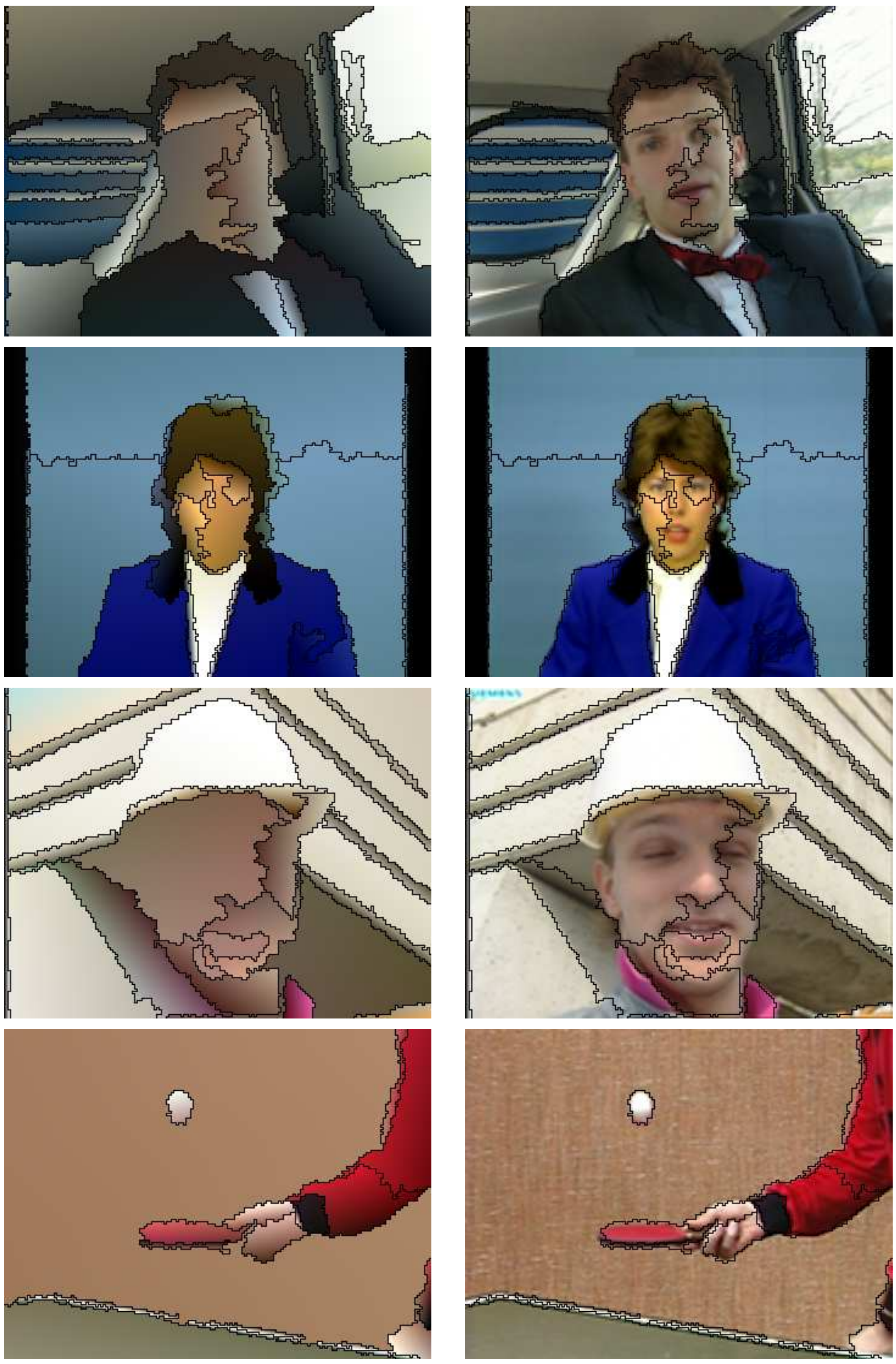


Figure 4.16: “Carphone”, “Claire”, “Foreman”, and “Table Tennis” (QCIF, image 0): segmentation into 25 regions using the affine RSST algorithm; boundaries superimposed over the color according to the estimated region model parameters (left) and over the original image (right). No split phase was used.

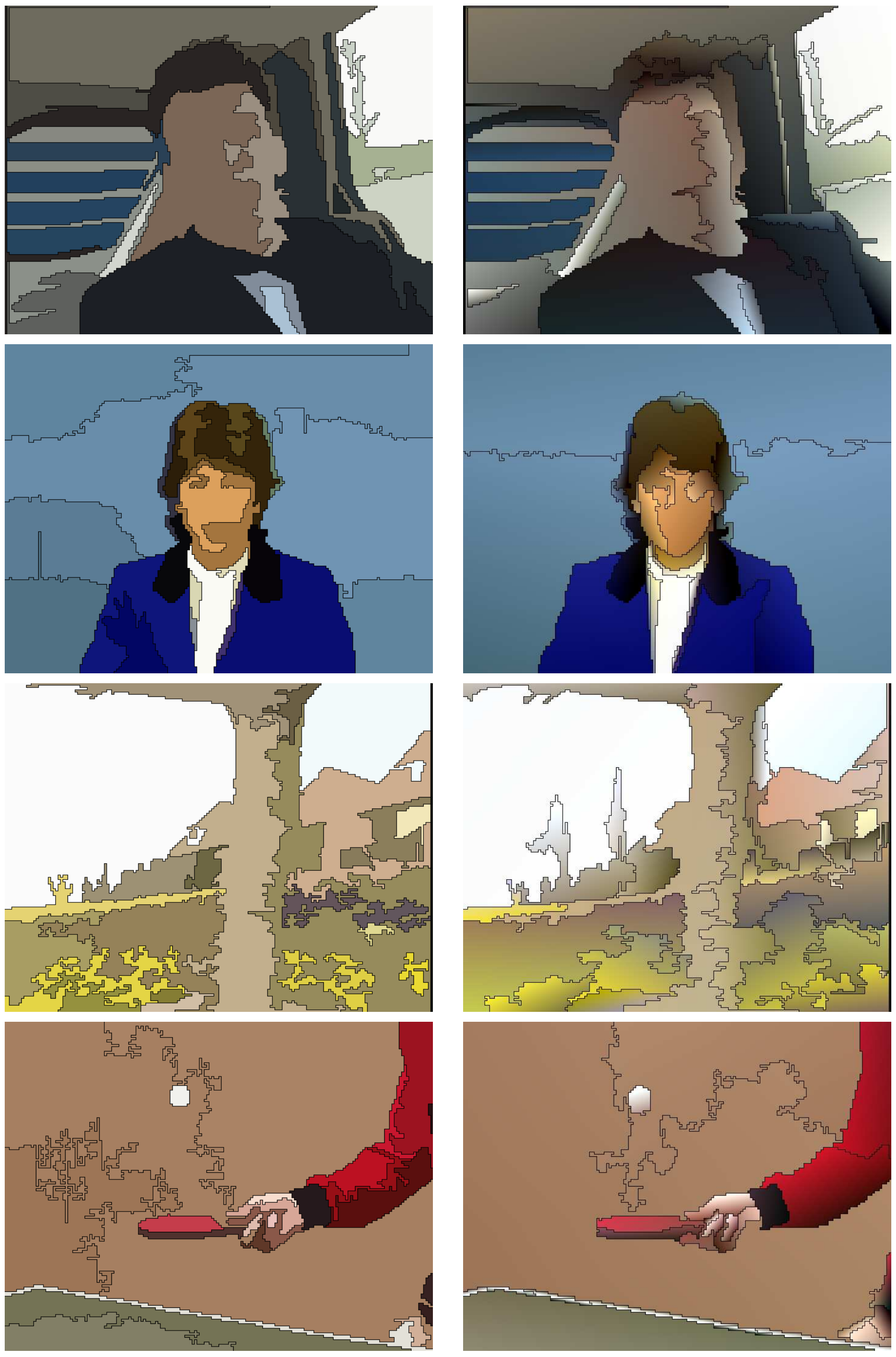


Figure 4.17: “Carphone”, “Claire”, “Flower Garden”, and “Table Tennis” (image 0): segmentation into 25 regions using the flat RSST algorithm (left) and the affine RSST algorithm (right). Images initially split into 2×2 blocks.

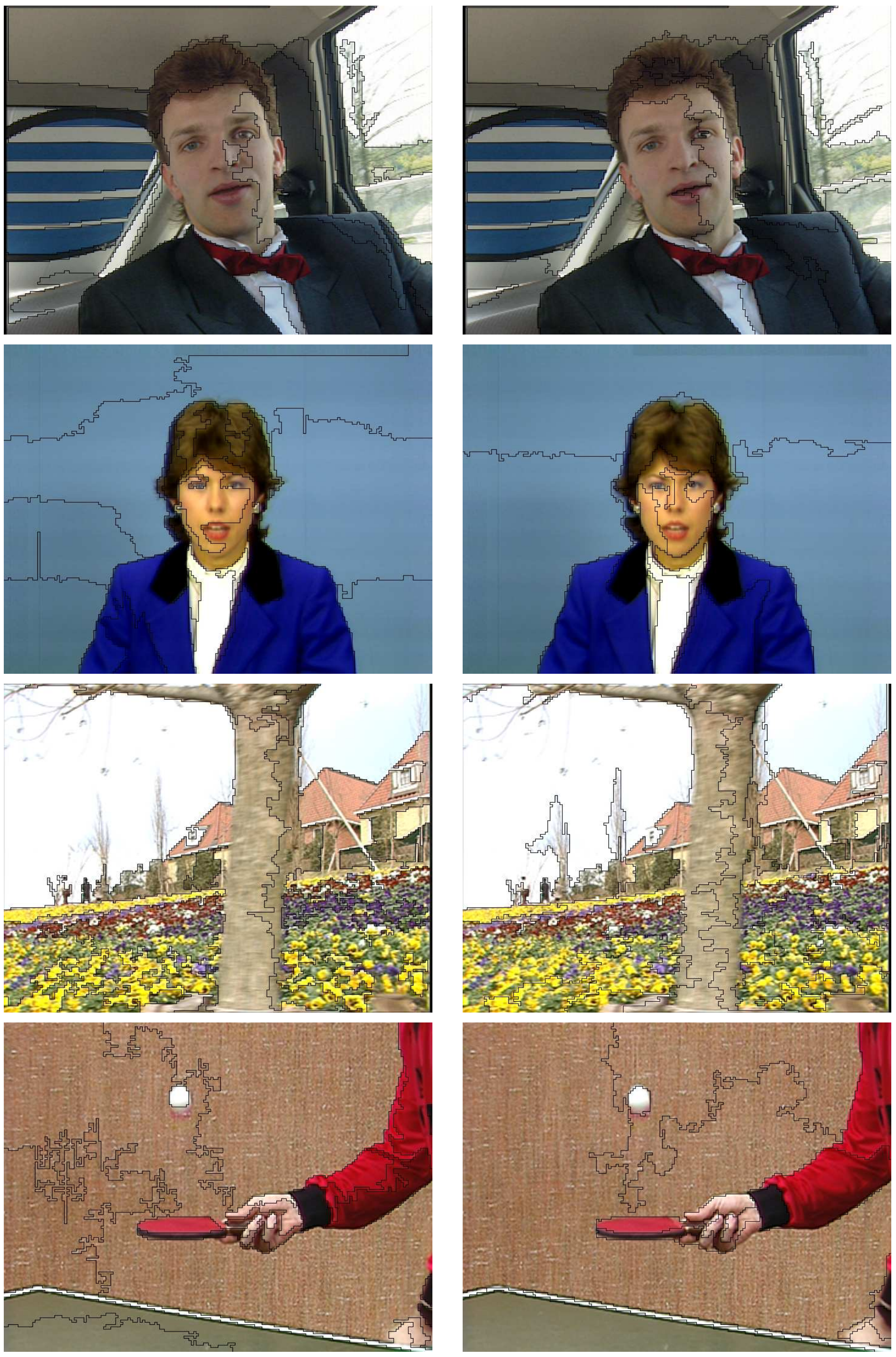
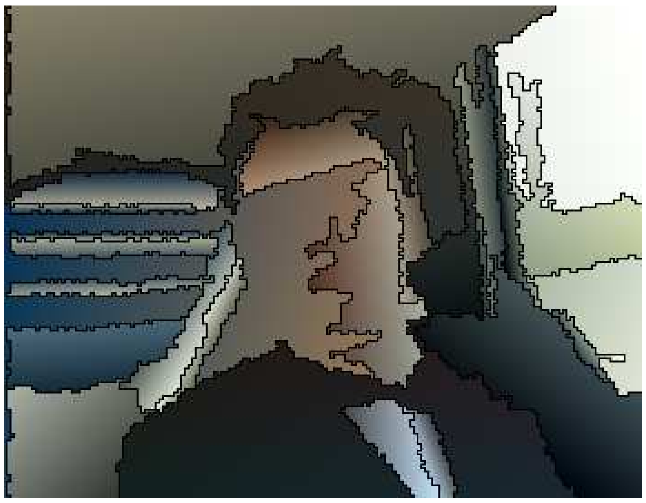


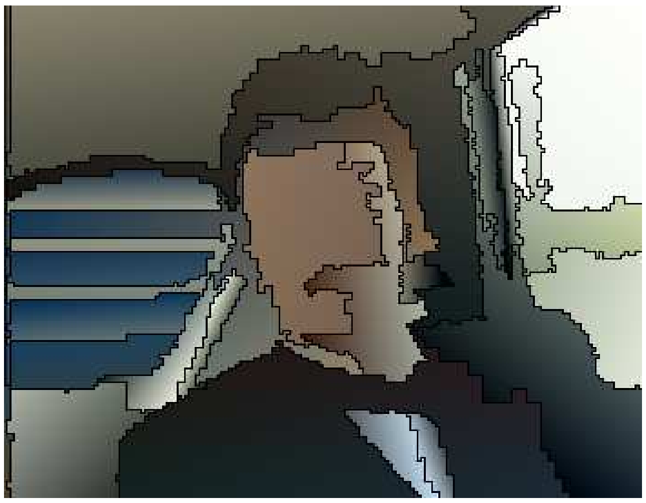
Figure 4.18: “Carphone”, “Claire”, “Flower Garden”, and “Table Tennis” (image 0): segmentation into 25 regions using the flat RSST algorithm (left) and the affine RSST algorithm (right). Images initially split into 2×2 blocks. Boundaries superimposed on the original.

Advantage of using a split phase

Figure 4.19 shows the segmentation of the first image of the QCIF "Carphone" using the affine RSST with and without a split phase.



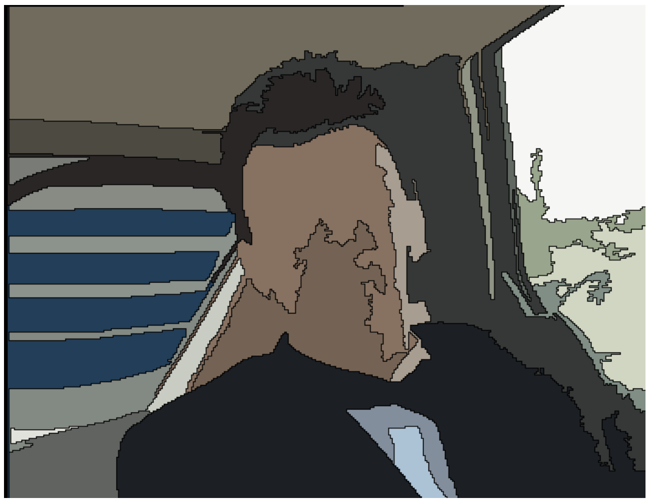
(a) Without split.



(b) With split.

Figure 4.19: QCIF "Carphone" image 0: segmentation into 25 regions using the affine RSST algorithm.

These results show clearly that the use of a split phase, aside from leading to considerable increase of computational efficiency and reductions in memory requirements, has a very positive impact on the performance of the algorithm. This performance can also be assessed in Figure 4.20, which compares the flat RSST algorithm (without split) with the affine RSST algorithm using the split phase, this time applied to the CIF "Carphone". Notice that, as will be seen in the next section, the split phase does not yield significant improvements in segmentation quality in the case of the flat RSST. In fact, the affine RSST is the only algorithm whose segmentation quality improves with a split phase. This effect is easily explained by the fact that, using a split phase, the attained regions are seldom small, thus avoiding the over adjustment described previously.


(a) Flat RSST without split.

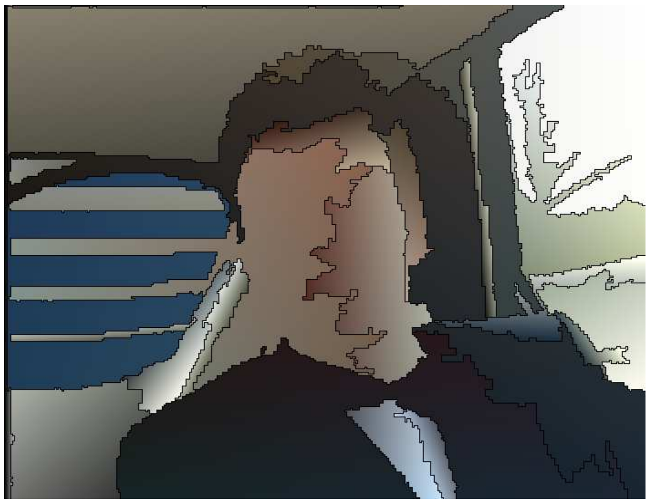

(b) Affine RSST with split.

Figure 4.20: "Carphone" image 0: segmentation into 25 regions.

It may be argued, and probably very rightly so, that the results might be further improved if splitting were done also using the affine model; for instance, by splitting regions while the average squared approximation error is above a threshold. This issue, as well as the more interesting problem of reflecting in the split phase the objective of minimizing the *global* approximation error, have been left for future work.

The split phase

Impact on segmentation quality

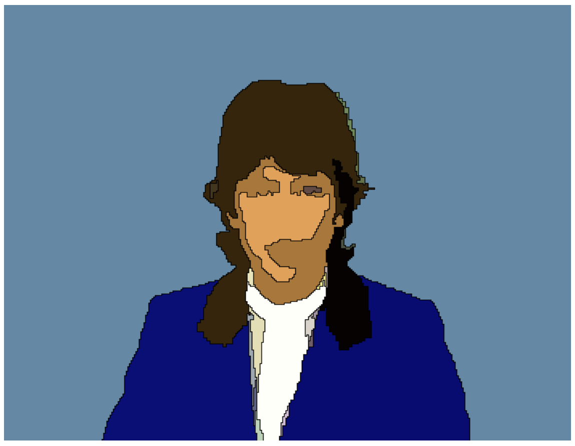
It was shown previously that using a split phase in the affine RSST algorithm is a good way of avoiding the over adjustment problems for small regions characteristic of the affine model. Figure 4.21 shows results of applying the new and flat RSST algorithms with and without a split phase to the first image of "Claire".

The quality of the new and flat RSST algorithms' results are not strongly affected by the split phase. This fact may be confirmed for a wide range of test images. The strongest negative impact of split in the appearance of artificial shapes in false contours, as can be seen in the background of the flat RSST result (the new RSST algorithm is more robust to false contours). It is only the shapes of false contours which are affected by the split phase, not false contours themselves, which exist with or without it. However, it may be argued that, since they are false contours anyway, their shape is not too relevant and they can be removed by post-processing, as done in [194]. Also, the use of more powerful models leads to less false contours and hence to a less negative visual impact of the split phase. All in all, it may be said that, since split has either a positive or a slightly negative impact in segmentation quality, it will be amply justified if it leads to significant savings in either or both computational time and memory requirements.

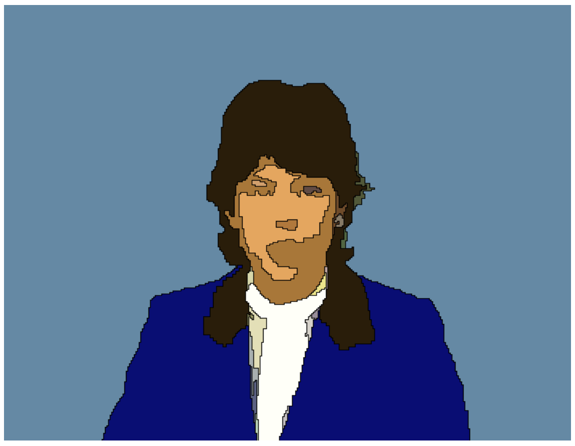
Impact on running times

The execution times of the new and flat RSST algorithms is given in Table 4.2.[14] The first thing to notice is that the new RSST algorithm is slower than the flat RSST algorithm. This is due to the fact that merges in new RSST are done according to three different criteria, which imply two different orderings of the graph arcs: one according to the color distance, another according to the size of the regions. On the contrary, the flat RSST (and the affine RSST, for that matter) require only a single ordering of the graph arcs. A similar comparison is performed in Table 4.4 between the flat and affine RSST algorithms, though with a minimum block size of 2×2. Noticing that the segmentation algorithm in flat and affine RSST is actually the same, the performance difference is due essentially to the increased affine model complexity, which implies solving a linear least squares problem for calculating the weight of the graph arcs. Both tables also show the considerable speed gains when using a split phase, with the exception of two cases: "Salesman" and "Table Tennis" for the flat and affine RSST algorithms, in which the added computational complexity of the split phase outweighs the savings in the merge phase. This is probably due to the fact that the implementation used calculates the region model parameters during the split phase. Since the split phase, as implemented, does not use the region model, these calculations would be better postponed to the end of the split phase. This was not done,

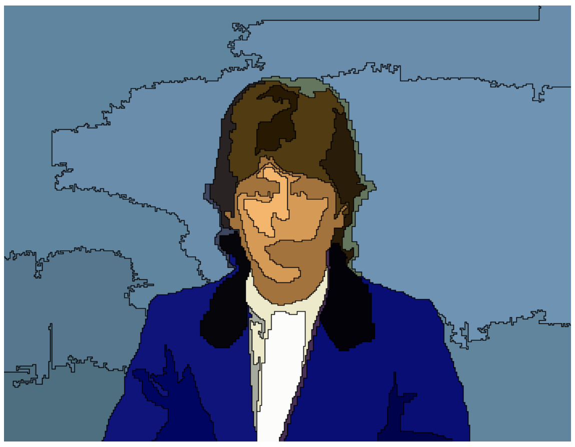
[14]Results on a Pentium 200 MHz, with 64 Mbyte RAM, running RedHat Linux 5.0 (kernel 2.0.32), programs compiled with gcc 2.8.1 and full compiler optimization (-O3), times obtained with gprof 2.8.1.



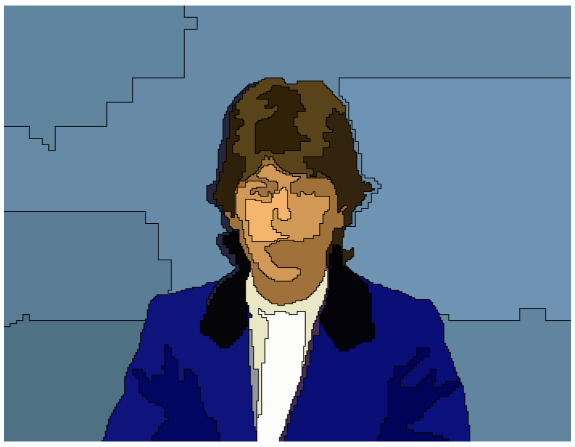
(a) New RSST without split.



(b) New RSST with split.



(c) Flat RSST without split.



(d) Flat RSST with split.

Figure 4.21: “Claire” image 0: segmentation into 25 regions.

sequence	no split	split
"Carphone"	29.52	8.56
"Claire"	24.20	3.29
"Flower Garden"	34.16	19.06
"Foreman"	29.44	9.46
"Miss America"	25.81	3.88
"Salesman"	29.11	12.45
"Table Tennis"	35.71	11.99
"Trevor"	27.71	5.42
average	29.46	9.26

(a) New RSST algorithm.

no split	split
13.83	6.39
13.48	2.25
22.64	13.25
14.76	6.40
13.70	3.94
14.49	9.41
15.34	12.78
13.85	5.57
15.26	7.50

(b) Flat RSST algorithm.

Table 4.2: Execution times (in seconds) of 4.2(a) new RSST and 4.2(b) flat RSST for CIF sequences (image 0).

though. Table 4.3 shows the execution times of the affine RSST when applied to QCIF images, which also confirms the advantages of the split phase.

It should be noticed, though, that the performance improvement due to the split phase may be reduced if the merge phase is further optimized, and vice-versa. Hence, as both phases are optimized, an eye should be kept on the global result so as to assess whether or not the split phase is really advantageous.

Finally, these results, and especially the averages, should be taken with a grain of salt, since, even though the test images used form an acceptable sample for testing generic algorithms, the results may be different if other images are used.

Sequence	no split	split
"Carphone"	31.05	17.73
"Claire"	34.03	9.33
"Foreman"	32.53	19.29
"Grandmother"	31.70	17.19
"Miss America"	30.94	9.46
"Mother and Daughter"	31.45	17.40
"Salesman"	31.64	23.55
"Table Tennis"	33.68	24.25
"Trevor"	34.10	14.28
average	32.35	16.94

Table 4.3: Execution times (in seconds) of affine RSST for QCIF sequences (image 0).

sequence	no split	split
"Carphone"	3.62	3.32
"Claire"	3.33	1.23
"Flower Garden"	4.93	3.73
"Foreman"	3.81	3.29
"Miss America"	3.59	3.10
"Salesman"	3.24	3.88
"Table Tennis"	3.49	4.50
"Trevor"	3.18	3.23
average	3.65	3.29

(a) Flat model.

no split	split
37.13	33.28
34.32	13.14
47.15	41.69
37.25	34.71
33.00	27.71
34.07	37.30
37.90	45.48
40.29	32.11
37.64	33.18

(b) Affine model.

Table 4.4: Execution times (in seconds) of 4.4(a) flat RSST and 4.4(b) affine RSST for CIF sequences (image 0) using a minimum block size of 2×2.

4.5.4 Conclusions

Three segmentation algorithms have been compared: the new RSST [32, 33], the flat RSST [194], and the affine RSST, which is the same algorithm as in [194] extended so as to use an affine region model. The last two algorithms have been tested also with optional image simplification pre-processing and a split phase. The flat RSST was shown to be more generic than the new RSST, even though it is less robust relative to false contours. The flat RSST was shown to be relatively insensitive, in terms of segmentation quality, to the use of image simplification, which is essential in the new RSST, and to the use of a split phase. The affine RSST has shown a great potential, even though some problems still need to be solved. In the case of the affine RSST, the split phase was shown to have a positive impact on segmentation quality, since it tends to minimize the negative effects of the over adjustment of the model for small regions. In terms of computational requirements, the split phase was shown to provide considerable savings, even if its implementation still requires optimization. The algorithms proposed in the next sections, which deal with supervised segmentation and coherence of temporal segmentation, make use of the flat RSST with a split phase and no image simplification.

4.6 Supervised segmentation

4.6.1 RSST extension using seeds

The global approximation error minimizing RSST algorithms, either using the flat or the affine region models, may be stopped either when the error exceeds a certain threshold, or when a required number of regions has been attained. These algorithms provide no means for controlling the position of the resulting regions or for specifying seeds around which the regions of interest should be obtained, as happens in region growing algorithms (such as watersheds [105]). However, they can be easily extended with such features, as mentioned previously, and as first proposed in [119].

Consider a label image with the same size as the original image. Let label zero denote unseeded pixels and seed s, with $s \neq 0$, be the set of all pixels with label s (which may or may not be connected). The set of existing seeds plus the set of unseeded pixels can be seen as a partition of the image. The label images may be restricted to those having connected seeds, but this is not necessary. The RSST algorithms can be extended such that an initial labeling is taken into account during segmentation. During the split phase (if there is one) the regions are forced to be split if they contain pixels belonging to different seeds. During the merge phase, when searching for the next two adjacent regions to merge, one may simply say that regions with pixels of different seeds should be merged last, if ever, or that regions with the same seed should be merged first. Further, whenever an unseeded region is being merged with a seeded region, the resulting region will inherit the seed of the seeded one. If adjacent regions with different seeds are prevented from being merged, the final partition respects the seeds, in the sense that all regions contain at most pixels of one seed.

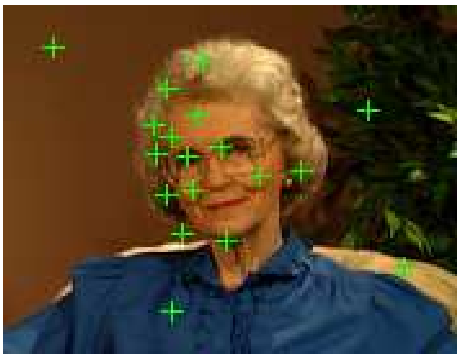
Occasionally, however, it may be acceptable to merge regions with different seeds. If this happens, the resulting region may inherit either the highest (or lowest) label or the label of the largest region, for instance.

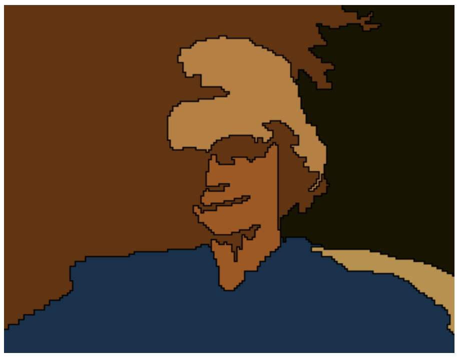
4.6.2 Results

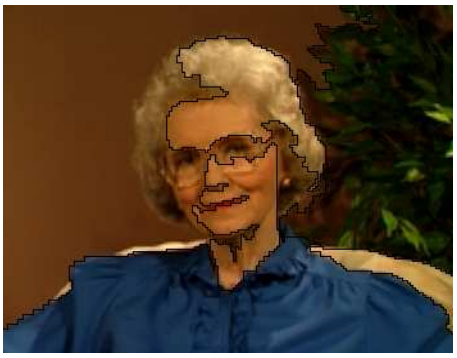
Figures 4.22(b) and 4.22(c) show the result of segmenting the first image of the "Grandmother" sequence with the flat RSST algorithm and targeting at six final regions. Figures 4.22(d) and 4.22(e) show the result of segmenting the same image though with the *seeded* flat RSST algorithm, and resorting to the set of seed pixels represented by crosses in Figure 4.22(a). Six different seeds were used: background, plant, sofa, hair, face and body. The pixel seeds on the hair belong all the the hair seed, the same thing happening with the crosses on the face and the background. The seed locations were chosen in an empirical way, until the desired result was attained.

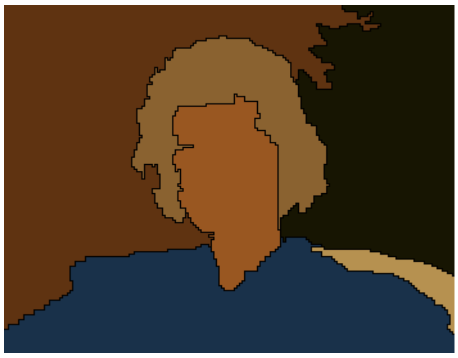
4.6.3 Conclusions

It is obvious, from the example in Figure 4.22, that, by simple mouse clicks, a human can supervise the segmentation algorithms of the previous sections to improve the results, i.e., by imposing semantical meaning. In a way, thus, a second-generation, mid-level vision tool is being helped to attain high-level vision results. It may be argued that unsupervised third-generation algorithms may also be attained by developing new segmentation algorithms which incorporate semantical information from the very start. However, it may be more natural to rely on algorithms of the lowest levels and to find good automatic supervision algorithms, since this makes the problem much more amenable. This was already recognized, in a way, by Pavlidis in [154, p.91], which called supervision "interpretation guided editing."


(a) Original image with pixel seeds represented by crosses.


(b) Without using seeds.


(c) Without using seeds, over the original.


(d) Using seeds.

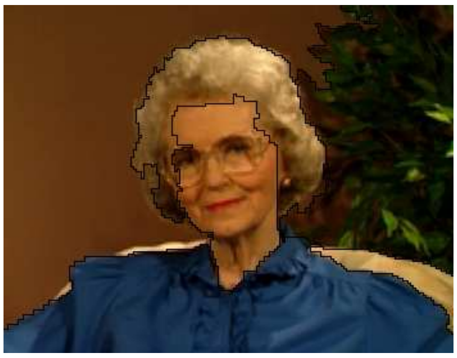

(e) Using seeds, over the original.

Figure 4.22: Segmentation of "Grandmother" image 0 with the flat RSST algorithm (using a split phase with $t_s = 12$) targeting at six final regions and with the same algorithm equipped with six different seeds: background, plant, sofa, hair, face, and body.

The simple extension of the RSST algorithms proposed here can also be used, as will be seen in the next section, to effectively segment a set of images in an image sequence in a time recursive fashion, and so as to maintain coherence between the segmentation results along time.

4.7 Time-coherent analysis

In the framework of moving image analysis, viz. for image coding, the time coherence of segmentation is important for at least two reasons. The first one has to do with manipulation/interactivity: when the user/spectator of the information is allowed to interact with the scene, it must be possible for him to select, zoom, rotate, or change any scene objects. This implies that objects must be identified, and this identification must be coherent along each of the images of the sequence in which the objects occur. Hence, if the user decides to change the color of a given car which he selects in a particular image of the moving scene, that car's color must be changed along the whole set of images in the sequence. The second reason has to do with coding efficiency: segmentation coherence is important because it allows for improved time prediction tools to be used.

This section extends the RSST segmentation algorithms so as to deal with moving images. This extension makes use of the extension of the RSST algorithms using seeds to perform a time-recursive segmentation. Time recursion is introduced by using the previously segmented regions as seeds for the segmentation of groups of images, such that the previous segmentation results are projected into the current image. Projection of segmentation results is not a new idea, having been proposed in [105] and in its precursor [154, p.92] which states that "... in moving scenes where the segmentation of a previous frame may be used as an initialization for the current frame." However, its use in conjunction with the RSST algorithms is, to the knowledge of the author, original, and was first proposed in [119].

4.7.1 Extension of RSST to moving images

The extension of the RSST algorithms to moving images is simple: stack the successive 2D images of a sequence into a 3D image, consider the 3D 6-neighborhood, and leave the rest of the algorithms unchanged. If the image sequence grid is rectangular, this is its natural extension into three dimensions, assuming, of course, a progressive sequence format. Of course, the split phase, if there is one, now proceeds according to an octal picture tree, instead of a QPT. Also, it is now less than clear whether the affine region model (for 3D) should be used, and what its meaning is. The flat model can and will be used without change, though.

Requirements

When segmentation of long sequences of 2D images is the aim, simply segmenting the 3D images obtained by stacking a few 2D images at a time may cause problems. The first problem is that the aim is to obtain a sequence of 2D partitions. For instance, a perfectly acceptable

3D partition with connected regions may lead to some 2D partitions containing disconnected regions.[15] Also, if an object has undergone a large movement from one image to the next, it will result in two objects in 3D segmentation. It should, however, result in a single object whose location in two successive images does not overlap.

Another problem has to do with the number of images that should be stacked before performing 3D segmentation. Clearly, the ideal would be to stack as many images as possible. However, this can easily result in both an overwhelming amount of data to process (segmentation can be very memory demanding) and unacceptable delays when segmentation is to be performed in real time, because segmentation cannot proceed until all images are available. Also, no matter how many images one stacks before 3D segmentation, there will always come a time when the next stack of images will have to be processed. The coherence between the previous and the next partitions of stacks of images will then be lost, unless other measures are taken.

Concluding, 3D segmentation algorithms should be able to track regions along time, should not demand too much computational power, and should introduce small delays for real-time applications.

Time recursiveness

A solution for the problem of maintaining temporal coherence in segmentation is described in [105], even though the idea is clearly a variation of the one exposed in [154, p.92]. The idea is to perform 3D segmentation on stacks of images that overlap along time, and use partitions obtained in the past segmentations as seeds for the present segmentation, thus introducing time recursiveness. The minimum configuration providing time recursiveness thus consists of stacks of two images, maintaining a time overlap of a single image. Since the RSST segmentation algorithms tend to consume a large amount of memory, the use of pairs of images is amply justified by implementation considerations.

When the past partitions are grown into the present through 3D segmentation, a connected 3D region in the obtained partition may turn to be disconnected if restricted to a smaller time range. For instance, in the pairwise 3D segmentation scheme suggested above, the 2D partition corresponding to the present image in a just segmented pair of images may have disconnected regions. This problem is solved easily if,[16] after the 3D segmentation involving all the images in the stack, the segmentation proceeds using only the present images. Before that, however, the regions which were split into more than one connected component will be unseeded, with the exception of one of its connected components. Usually the largest connected component retains the label of the originating seed in the past. This simple solution thus may create regions with new labels.

Using time recursiveness, some regions may not grow into the present, and hence regions, and the corresponding seed labels, may disappear when no correspondence is found from the past

[15]However, in some cases this may actually be desirable. For instance, when the 2D projection of an object is split into two disjoint regions by another object closer to the observer. In this case it is arguable that the two disjoint regions are one and the same object.

[16]Again, this is only a problem if the classes in the partitions are required to be connected, which is often the case.

to the present. Further, some regions in the present may not correspond to any region in the past, and hence new labels may also appear this way.

The problem with time recursiveness using the described methods is that the number of regions tends to grow. This is caused because of illumination effects which often create new (artificial) regions, no matter how complex the region models are, and because regions seldom disappear, even when a stopping criterion based on the global approximation error is used. Hence, it is desirable to complement the segmentation steps explained above with a further step in which the segmentation no longer respects the seeds, some differently labeled regions being allowed to merge.

Coding

Though this chapter does not address directly the partition coding problem, it should be noted that information about the relation between the current and the past class labels should be sent along with the class shape information. This information may, for instance, state which classes in the past images ceased to exist, which new classes were created, and which classes were split or merged. The relation between current and past class labels may also be of help for the encoding of contours, of color (the inside of the regions), and even of motion.

4.7.2 Results

Figure 4.23 shows the results of segmenting the first image of the QCIF "Table Tennis" sequence using the flat RSST algorithm with a split step in which $t_s = 12$. The target root mean square color distance used was 22 (corresponding to a PSNR of 21.3 dB). A low PSNR target was chosen in order to obtain a final number of regions which would be meaningful on printed paper. The simple flat region model used accounts for the false contours in the background and also for the division of the sleeve into regions of different shading.

The time coherence of the TR-RSST segmentation algorithm can be seen in Figure 4.24, which shows the time evolution of ten classes of the segmented sequence corresponding to the arm, hand, and racket of the player. These ten classes correspond to only nine labels, since label ten (gray), used initially for the border of the racket, is later reused in the lower part of the arm. The algorithm introduces new regions when the approximation error is not good enough, as can be seen in the lower part of the arm from the eighth image on.

4.7.3 Conclusions

This section described an extension of the flat RSST algorithm which deals with sequences of images while maintaining coherence of the obtained partitions from image to image. The results obtained are reasonable and show that the method is interesting for use in second-generation video codecs.

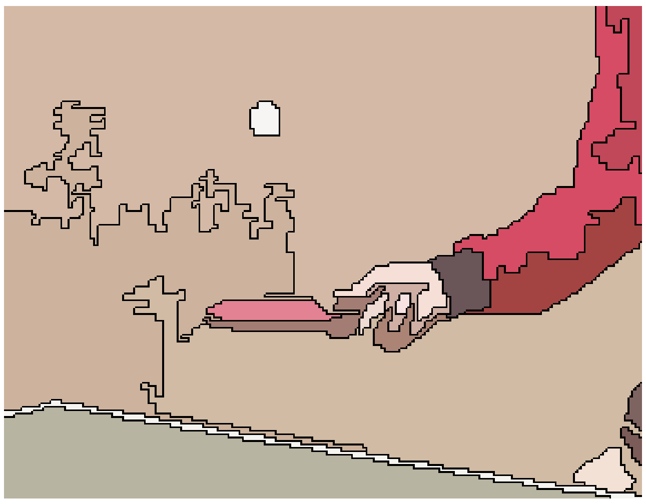


Figure 4.23: Segmentation of QCIF “Table Tennis” image 0 using flat RSST.

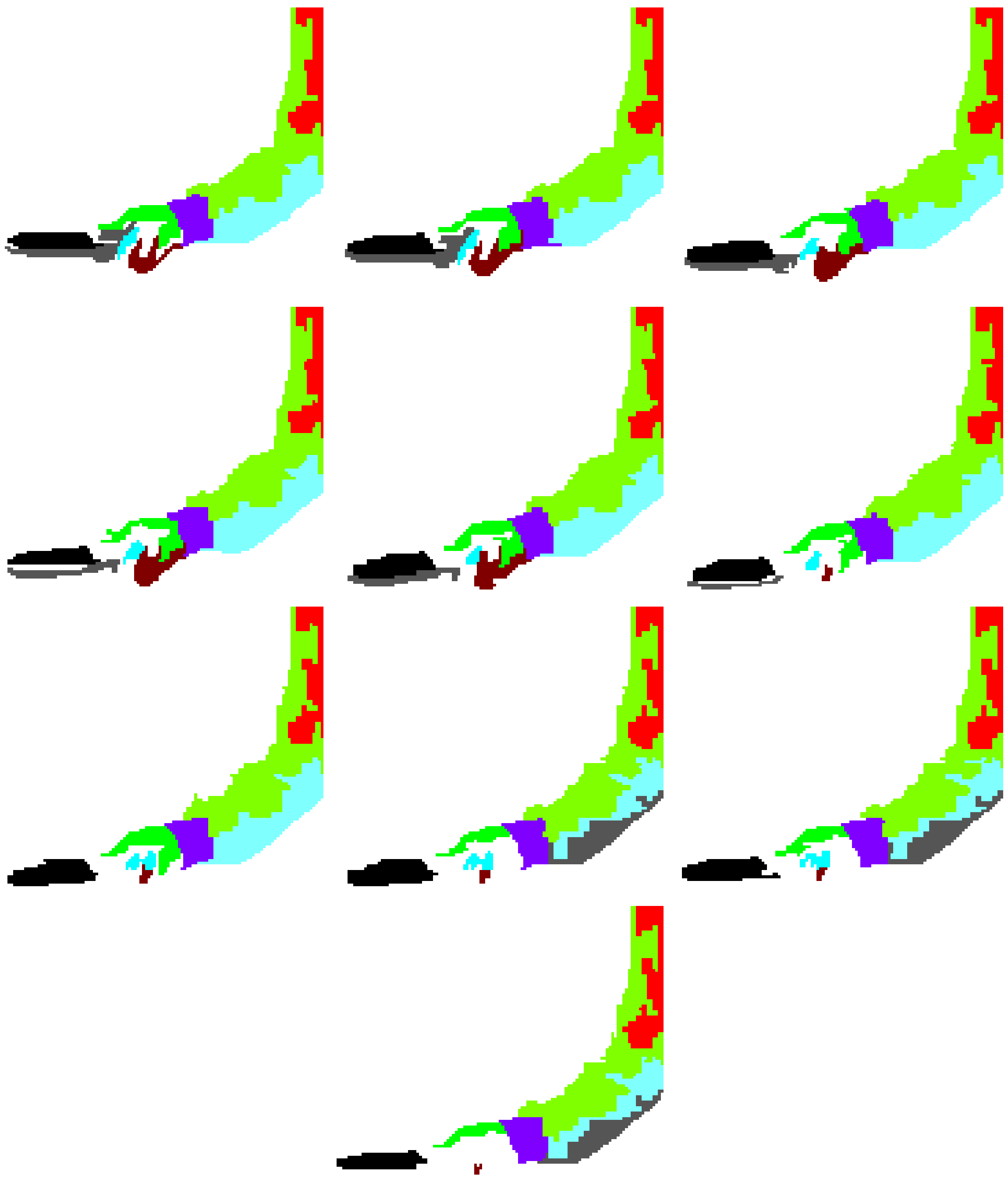


Figure 4.24: Segmentation of images 0 to 9 of QCIF "Table Tennis" with the TR-RSST algorithm. Ten classes are shown in different colors: hand (three classes, brown, dark blue, and dark green), racket (one and two classes, black and gray), sweater cuff (one class, violet), sleeve and shoulder (three and four classes, light blue, light green, red, and gray).

Chapter 5

Time analysis

Todo o Mundo é composto de mudança,
Tomando sempre novas qualidades.

Luís Vaz de Camões

There is a considerable interdependence between motion estimation and segmentation, i.e., between time and space analysis of moving images. Estimation of motion, even at pixel level, requires regularization, since otherwise the results will not be likely to correspond to the 2D projection of real 3D motion (due to the aperture problem [70]). On the other hand, if regularization is implicit in some motion model (with some physical meaning), then that motion model must be applied to individual regions: it is extremely unlikely that a single set of model parameters can describe the motion of all the pixels in a moving image: the real world scenes usually consist of rigid or semi-rigid objects with independent motion. Hence, segmentation into zones with different motions must be performed. The question that remains is which should be performed first: motion estimation (i.e., estimation of the model parameters) or segmentation? The obvious response seems to be to perform both simultaneously, but that is not an easy task. This was the issue of discussions in the MotSeg (Motion Segmentation) group of the MAVT (Mobile Audio-Visual Terminal) project (chaired by the author), see [135, 115]. It will not be further discussed here.

Camera movement introduces global motion, that is, motion which is independent (or nearly so) of the contents of the scene. Nevertheless, this motion may be superimposed into motion of the objects in the scene, so that segmentation is an important issue even in camera movement estimation. This chapter presents the camera movement estimation algorithm first published in [122] and also proposes an improvement based on the Hough transform which is similar to the one proposed in [4]. The improvement stems from the segmentation part of the algorithm, which is based on a Hough transform clustering segmentation technique, even if, in the sequel, it is interpreted, more in the light of robust estimation methods, as an outlier detection mechanism. Note that clustering segmentation techniques which make no use of topological restrictions, such as connectedness of the attained classes, do make sense in the framework of camera movement estimation, since these movements introduce global motion which is independent of the scene contents. The zone in the image whose motion corresponds to camera movement is that in which the scene is static from one image to the next. Modeling the shape or connectedness of this zone would probably not lead to improved camera movement estimation.

The proposed camera movement estimation algorithms deal with two types of movement: pan (which also encompasses tilt) and zoom. These algorithms, and especially the one based on the Hough transform, can be used for both compensation of camera movements, in the framework of video coding, or for image stabilization in the case of vibration or small pan movements originated in hand-held video cameras, for instance in mobile videotelephony. Both can be classified as transition to second-generation tools, since they make use of more structured information extracted from the image sequence. The first tool, camera movement compensation, will be dealt with in the next chapter, and the second one, image stabilization, in Section 5.5.

The algorithm based on the Hough transform is a natural evolution of the one in [122], which in turn stemmed from earlier work by the author [129, 127, 130, 128, 113]. It can also be seen as a simpler version of the algorithm in [4].

5.1 Camera movements

Two types of camera movement will be considered: panning and zooming. Panning corresponds to the usual movement used to capture a panoramic view. In this section, however, it will be taken to mean any rotation of the camera about an axis parallel to the image projection plan, so that it includes pan and tilt as particular cases. Rotations of the camera around the lens axis will not be considered. In a pure pan movement, the axis of rotation is taken to pass through the optical center of the camera objective.[1] The effect of a pure pan is approximately a translation of the projected image. The deviations from a true translation are due to several factors, of which the most important are that the in focus plan changes (objects which were in focus are now out of focus and vice versa) and that the perspective of the projected image also changes (formerly parallel projected lines are now convergent and vice versa). The first effect may be reduced by reducing the objective aperture, which results in a larger depth of field, i.e., it increases the maximum distance to the in focus plan where objects have an apparently sharp image in the projection plan by reducing the area of the corresponding circles of confusion. The approximation to a true translation, however, is good provided the rotation performed by the pan movement is small. Notice that, if two pan movements in objectives with different focal lengths (or the same zoom lens at two different focal lengths) result in translations of the image with the same magnitude, the approximation to a pure translation is better for the larger focal length. This happens because the corresponding camera rotation is smaller.

Zooming corresponds to the continuous change of the focal length of the camera while keeping the originally focused plane in focus, i.e., with a sharp image in the projection plan. In a pure zoom movement, the optical center of the lenses does not change its position. Even if the optical

[1] Actually, the axis of rotation is taken to pass through the principal point of the lens system closer to the object.

center does change, it is usually by a very small distance, especially when compared with the distance between the camera and the viewed objects. Zooming corresponds thus approximately to a proportional scaling, an isometric transformation of the projected image. If the zoom is not pure, then this is only partially true, since the change in the position of the optical center of the lens introduces perspective changes in the projected image, which cannot be described by a simple scaling. For the same aperture (or pupil), a change in the focal length which is accompanied by a corresponding change in the projection plan position so as to keep the in focus plane unchanged, results in a change of the blurriness (i.e., the area of the circle of confusion corresponding to a given point) by approximately the same factor as the projected sizes, and hence seems to corroborate that zooming corresponds to a simple scaling of the image. But, since increasing the focal length usually also implies augmenting the objective aperture, the circles of confusion are enlarged more that the image itself (i.e., the depth of field is reduced).

It will be assumed in this chapter that a rectangular sampling lattice is used to sample the moving images, and that the spatial active area of this lattice (corresponding to the domain of the digital image) is centered around the lens axis. Thus, zooming corresponds to an isometric scaling around the center of the active area. This assumption, however, is not fundamental: if the center is not on the lens axis, a fictitious pan movement will be estimated along with the zoom to compensate for the offset. This must be taken into account when analyzing the results, though.

The degree to which a pan or zoom movement results in approximate translations and scalings of the projected image is beyond the scope of this thesis. The reader is referred to any good textbook on optics dealing with lens systems, the usual optical aberrations, and camera technology (a simplified treatment can be found in [102]). However, it may be said that the non-linear effects of panning and zooming are small for small movements. Also, since these movements will be estimated (using the algorithms described in this chapter) from one image to the next in an image sequence, they correspond to fractions of movement lasting typically from $\frac{1}{60}$ to $\frac{1}{25}$ of a second. These intervals are generally short enough (or the camera operators slow enough) for the fractional movements to be small from image to image. However, these effects do become more evident when integrating the incremental movements over an entire sequence. Again, they will be neglected in this thesis, except as a (partial) justification for the cumulative errors of the estimated movement factors along a sequence.

Panning movements have an immediate equivalent in the HVS: changes of gaze direction through eye or, to a certain extent, head movements. Our eyes, unfortunately, have no zooming capabilities. The equivalent of zooming is performed by the upper levels of the HVS, by concentrating more or less the attention to the part of the image near the fovea. In the case of the HVS several position and attitude feedback mechanisms (from the eye and neck position, and from the equilibrium sensors in the inner ear) simplify the brain's task while performing a match between successive images.[2] The role of camera movement estimation then is to perform a similar function in the absence of position feedback. It should be noticed that zoom movements, being related directly to lens positions, might be sensed, quantized, digitized and stored or fed back for each image with little cost, thus rendering estimation useless. As to pan movements, the

[2]Images in the retina are formed in a continuous way, of course, but the changes in gaze direction are performed through saccadic eye movements, which render changes almost discrete. Besides, there is evidence that the visual function is effectively suppressed during these saccadic movements.

same process might be done, though the sensing devices would probably be costlier. In practice, none of these movements is captured along with the images, and hence has to be estimated.

5.1.1 Transformations on the digital image

Image and pixel coordinates

The notation defined in Sections 3.2.2 and 3.2.3 will be slightly extended. As before, s usually corresponds to a lattice site (i.e., a coordinate in the image plan), addressed by the digital image pixel v. That is,

$$s = \begin{bmatrix} u_0 & \dots & u_{m-1} \end{bmatrix} v.$$

This equation may be simplified by specifying a rectangular sampling lattice in $\mathbb{R}^2$ (i.e., $m = 2$, $u_0 = \begin{bmatrix} 0 & -b \end{bmatrix}^T$, and $u_1 = \begin{bmatrix} a & 0 \end{bmatrix}^T$)

$$s = \begin{bmatrix} 0 & a \\ -b & 0 \end{bmatrix} v.$$

Using the usual notation for the coordinates of lattice sites and pixels

$$s = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 & a \\ -b & 0 \end{bmatrix} v = \begin{bmatrix} 0 & a \\ -b & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}.$$

However, the real sizes of a and b (as well as the relation between image size and real size of the imaged objects), are immaterial for the purposes of compensation of camera movement or image stabilization. Hence, the coordinates of the sampling lattice sites will be expressed in pixel width units, whatever they are, i.e.,

$$s = \begin{bmatrix} 0 & 1 \\ -b/a & 0 \end{bmatrix} v = \begin{bmatrix} 0 & 1 \\ -\frac{1}{\alpha} & 0 \end{bmatrix} v,$$

where α is the pixel aspect ratio corresponding to the sampling lattice used.

The domain of digital images sampled with a rectangular lattice is usually taken to consist of L lines of C pixels, where line 0 is the topmost line (and line $L-1$ is the bottommost), and column 0 is the leftmost column. In this case it will be useful to center the corresponding lattice sites around the origin, which is taken to be the point where the lens axis intersects the image plan. Hence, the previous equation is changed to compensate for this offset

$$s = \begin{bmatrix} 0 & 1 \\ -\frac{1}{\alpha} & 0 \end{bmatrix} \left(v - \begin{bmatrix} \frac{L-1}{2} \\ \frac{C-1}{2} \end{bmatrix} \right) = \begin{bmatrix} 0 & 1 \\ -\frac{1}{\alpha} & 0 \end{bmatrix} \begin{bmatrix} i - \frac{L-1}{2} \\ j - \frac{C-1}{2} \end{bmatrix},$$

or

$$\begin{aligned} x &= j - \frac{C-1}{2}, \text{ and} \\ y &= -\frac{1}{\alpha} \left(i - \frac{L-1}{2} \right). \end{aligned} \tag{5.1}$$

Expressing the pixel coordinates v in terms of the image coordinates s, the following equation results

$$v = \begin{bmatrix} 0 & -\alpha \\ 1 & 0 \end{bmatrix} s + \begin{bmatrix} \frac{L-1}{2} \\ \frac{C-1}{2} \end{bmatrix},$$

or

$$\begin{aligned} i &= -\alpha y + \frac{L-1}{2}, \text{ and} \\ j &= x + \frac{C-1}{2}. \end{aligned} \tag{5.2}$$

According to the notation introduced in Section 3.2.3, in equations (5.1) and (5.2) $s = \begin{bmatrix} x & y \end{bmatrix}^T$ should be taken to belong to $\mathbb{R}^2$ and $v = \begin{bmatrix} i & j \end{bmatrix}^T$ to $\mathbf{Z} = \{0, \ldots, L-1\} \times \{0, \ldots, C-1\} \subset \mathbb{Z}^2$. By a slight abuse of notation, however, v will be taken to belong to $\mathbf{R} = [0, L-1] \times [0, C-1] \subset \mathbb{R}^2$, in which case they can be thought of as representing sites in the analog image expressed in the usual pixel coordinates. Hence, s coordinates are image centered, oriented as usual (x-axis rightwards and y-axis upwards), and isotropic (in the sense that they express real distances, in some unit, in both directions), while v coordinates have origin in the center of the top-leftmost pixel, the first coordinate growing downwards and the second rightwards, and are generally not isotropic, since the pixel aspect ratio is not taken into account. These last coordinates however, translate nicely into pixels.

Pan and zoom movements

Pan movement

A pan movement, as seen, corresponds to a translation of the image. Let $d^s = \begin{bmatrix} d^s_x & d^s_y \end{bmatrix}^T$ be the translation vector corresponding to the pan movement. Then, a site $s = \begin{bmatrix} x & y \end{bmatrix}^T$ will be taken into

$$s' = s - d^s = \begin{bmatrix} x - d^s_x \\ y - d^s_y \end{bmatrix}$$

after the pan movement. Notice the sign of the translation vector: its direction reflects the camera movement, since a rotation of the camera to the right will translate the image to the left.[3]

Zoom movement

A zoom movement, as seen, corresponds to a scaling of the image about the lens axis. Let $Z > 0 \in \mathbb{R}$ be the scaling factor corresponding to the zoom movement. Then, a site $s = \begin{bmatrix} x & y \end{bmatrix}^T$

[3]Of course, this is only true after reversing the directions in the image, since the lens projects an inverted image.

will be taken into

$$s' = Zs = \begin{bmatrix} Zx \\ Zy \end{bmatrix}$$

after the zoom movement, where $Z > 1$ means zoom in, $Z < 1$ means zoom out, and $Z = 1$ means no zoom.

Combined pan and zoom movements

If a scaling is performed before a translation, the result is reproducible by performing an appropriately scaled translation before the scaling. Hence, it is immaterial which of the pan and zoom movements is performed first (if they are not performed simultaneously). But the camera movement estimation equations will be rendered simpler if a combined movement is modeled as a scaling after a translation. Hence, after a translation by d^s and a scaling by Z, site $s = \begin{bmatrix} x & y \end{bmatrix}^T$ will be taken into

$$s' = Z(s - d^s) = \begin{bmatrix} Z(x - d^s_x) \\ Z(y - d^s_y) \end{bmatrix} . \tag{5.3}$$

If the opposite question is asked, viz. where did s' came from, the following equation results

$$s = \frac{s'}{Z} + d^s = \begin{bmatrix} \frac{x'}{Z} + d^s_x \\ \frac{y'}{Z} + d^s_y \end{bmatrix} .$$

The vector which takes from s' to s is given by

$$s - s' = \frac{s'}{Z} - s' + d^s = (\frac{1}{Z} - 1)s' + d^s = zs' + d^s = \begin{bmatrix} zx' + d^s_x \\ zy' + d^s_y \end{bmatrix} ,$$

where $z = 1/Z - 1$ will be known henceforth as the zoom factor.

Converting this equation to pixel coordinates

$$\begin{aligned} \mathcal{M}(z,d)[i,j] &= v - v' = \begin{bmatrix} 0 & -\alpha \\ 1 & 0 \end{bmatrix} (s - s') = \begin{bmatrix} 0 & -\alpha \\ 1 & 0 \end{bmatrix} (zs' + d^s) \\ &= z \left(v' - \begin{bmatrix} \frac{L-1}{2} \\ \frac{C-1}{2} \end{bmatrix} \right) + d = \begin{bmatrix} z\left(i - \frac{L-1}{2}\right) + d_i \\ z\left(j - \frac{C-1}{2}\right) + d_j \end{bmatrix} , \end{aligned} \tag{5.4}$$

where $d = \begin{bmatrix} d_i & d_j \end{bmatrix}^T$ is the translation vector expressed in pixel coordinates. The quantity $\mathcal{M}(z,d)[i,j]$ is usually known as the backward motion vector at $v' = \begin{bmatrix} i & j \end{bmatrix}^T$ corresponding to the camera movement factors z and d, since, if the camera movement occurred from image $n-1$ to image n in a sequence, $\begin{bmatrix} i & j \end{bmatrix}^T + \mathcal{M}(z,d)[i,j]$ tells where pixel $\begin{bmatrix} i & j \end{bmatrix}^T$ (of image n) was on image $n-1$.

Division of the image into blocks

Let $L = B_L L_b$ and $C = B_C C_b$, i.e., the digital images can be divided into $B_L \times B_C$ blocks of $L_b \times C_b$ pixels each. Let b_{mn} be the block at the mth line of blocks (topmost is 0) and at the nth column of blocks (leftmost is 0). Each block can be seen as a small image. The pixel at coordinates $\begin{bmatrix} i & j \end{bmatrix}^T$ of block b_{mn} has pixel coordinates

$$\begin{bmatrix} mL_b + i \\ nC_b + j \end{bmatrix}$$

in the overall image, as can be readily verified. Hence, its corresponding camera movement motion vector is

$$\mathcal{M}(z,d)[m,n,i,j] = \mathcal{M}(z,d)[mL_b + i, nC_b + j] = \begin{bmatrix} z\left(mL_b + i - \frac{L-1}{2}\right) + d_i \\ z\left(nC_b + j - \frac{C-1}{2}\right) + d_j \end{bmatrix},$$

from which the average camera movement motion vector of block b_{mn} can be calculated as

$$\mathcal{M}_b(z,d)[m,n] = \frac{1}{L_b C_b} \sum_{i=0}^{L_b-1} \sum_{j=0}^{C_b-1} \mathcal{M}(z,d)[m,n,i,j] = \begin{bmatrix} z\left(m - \frac{B_L-1}{2}\right) L_b + d_i \\ z\left(n - \frac{B_C-1}{2}\right) C_b + d_j \\ \end{bmatrix}. \tag{5.5}$$

5.2 Block matching estimation

Motion compensation was a breakthrough technology in video coding. It relies on two general facts: the projected scene tends to change little from image to image, and hence the previous image may be a good prediction for the current one, and changes are mostly due to camera movements and/or motion of the imaged objects. Both types of movements can be estimated, and the estimated motion, or the estimated motion model parameters can be used to improve the prediction of the current image given the previous one. Hence, motion estimation is of paramount importance to video coding. Despite the fact that numerous algorithms have been proposed for motion estimation in the literature [5, 190, 169], the block matching algorithm, and its variants, is still the most used and the one for which hardware implementations are more readily available.

The rationale for block matching is simple. Motion cannot be estimated in a purely local fashion: if the closest match to a pixel is searched in the previous image, the attained motion vector field, that is, the set of motion vectors obtained for all pixels, is extremely nonuniform, even if restrictions in the search range are introduced. This is undesirable for two reasons. Firstly, if motion analysis aimed at scene understanding, a highly nonuniform motion vector field is unlikely to correspond to the projection of the real 3D motion in the scene: real motion tends to be reasonably uniform. The physical world is mostly constituted of semi-rigid objects, and thus the usual motion models used assume that the motion vector field has few discontinuities, which typically occur at the boundaries of the imaged objects. This is related to the fact that estimating a motion vector field from a pair of images (i.e., computing the so-called optical flow) is an ill-posedness problem [9], and hence some type of regularization must be used, typically through the use of additional constraints forcing the motion vector field to be uniform almost everywhere. Secondly, if motion is to be used for encoding, then the savings obtained through a better prediction should not be smaller than the cost of encoding the motion vector field. This last reason, together with engineering concerns about the practicality of its implementation in real codecs, led to a compromise. In the now classical video codecs, motion is estimated at a much lower resolution than the image resolution: motion vectors are estimated for each $L_b \times C_b$ block of pixels, where L_b and C_b divide respectively L and C (the digital image size), as in the previous section. Typical values for the block sizes are 8×8 and 16×16.

The estimate $\tilde{\mathcal{M}}_b[m,n]$ of the motion vector of block $b[m,n]$ of image f_n relative to image f_{n-1} is calculated by minimizing some measurement of the prediction error such as (see [143])

$$E[m,n](\mathcal{M}_b[m,n]) = \sum_{v \in b[m,n]} D(f_n[v], f_{n-1}[v + \mathcal{M}_b[m,n]]), \tag{5.6}$$

where D is some distance on the color space, and where the minimization is performed over a restricted set of possible values for the motion vector. The motion vector is usually allowed only to take values on a small rectangular window centered on the null (no motion) vector, e.g., $\mathcal{M}_b[m,n] \in \{-d_{i_\text{max}}, \ldots, d_{i_\text{max}}\} \times \{-d_{j_\text{max}}, \ldots, d_{j_\text{max}}\}$, which restricts the range of image motion with which the algorithm can cope. This window is usually further reduced at the image borders so that the reference block stays within the previous image, though methods have been proposed which prefer to extrapolate the previous image out of its borders. The values used for d_{i_max} and d_{j_max} in this chapter are 32 and 30, which are a good compromise between the range of acceptable image motion and implementation efficiency, at least for full search algorithms (see below) over CIF images. The values also happily compensate the pixel aspect rate of the image sequences used, see Section A.1.1.

Regularization in this case is implicit, since all pixels of a block are assumed to share a single motion vector. In a sense, thus, the motion vector field is only allowed to have discontinuities at the block boundaries, which are rather unlikely to coincide with real object boundaries. However, when a block contains (parts of) objects with different motions or contains uncovered parts of objects, the minimum prediction error attained is likely to be large, and hence can be used as a rough indication of the validity of the motion vector. Sequences with pure pan and/or zoom camera movements do not suffer from these problems (except possibly at uncovered borders), but are extremely rare in practice (and uninteresting).

If the motion vectors $\mathcal{M}_b[m,n]$ in (5.6) are allowed to take values in $\mathbb{R}^2$ instead of in $\mathbb{Z}^2$, then motion estimation is said to have sub-pixel accuracy. Such estimations are rendered amenable to implementation by restricting the values of the motion vectors to submultiples of the pixel, typically to half or quarter pixels. In any case, methods must be devised to interpolate image f_{n-1} at such inter-pixel positions, which can be done by standard linear filtering methods (see [150]) or simply by using bilinear interpolation.

5.2.1 Error metrics

The color space metrics used in practice for motion estimation simply neglects the chromatic information in the image, relying thus only on the luma of the image. This approach is sound, in practice, because images are usually available with considerable blurred chromatic content. In most cases, the chromatic components are even subsampled relative to luma.

The usual Euclidean distance is not used in practice, because it requires multiplication, rendering its implementation more costly, and because the city block distance, i.e., the sum of absolute values, besides considerably simpler, in practice yields almost equivalent results [143].

5.2.2 Algorithms

Several algorithms have been proposed to find the motion vector yielding the minimum prediction error. The rationale for not using an exhaustive search (or full search) was the heavy computational requirements. Such algorithms search only an appropriately selected number of motion vectors, at the cost attaining non-optimal estimations, and are described in standard texts on image compression, e.g., [143].

The use of the full search algorithm has the advantage of, at the cost of some extra memory, being able to store the prediction errors of all possible motion vectors of a block (instead of just a small subset), which may be useful for motion vector field smoothing or for estimating the covariance matrix of the motion vector, as will be seen in the sequel. Notice, however, that the calculation of prediction errors for all tested motion vectors is incompatible with a standard acceleration method for block matching algorithms, which, for each motion vector, scans the block line by line and, at the end of each line, checks whether the sum exceeds the current minimum prediction error. If it does, then the current motion vector cannot possibly yield a lower prediction error, the sum is stopped, and the error is not fully calculated. This method, usually named short-circuited estimation, yields considerable savings in computational requirements.

Finally, it should be mentioned that while minimizing (5.6), when more than one motion vector leads to the same minimum prediction error, the smallest motion vector is preferred in the algorithms proposed in this chapter. Other choices, such as the motion vector closer to some already estimated neighbor, may help introduce more regularity in the motion vector field.

5.3 Estimating camera movement

Motion estimation, of which camera movement estimation can be seen as a particular case, relies heavily on robust estimation methods, that is, methods that remain reliable in the presence of several types of noise. Good reviews of such methods, as applied to computer vision in general and motion estimation in particular, can be found in [111, 57].

The basic requirement imposed to the camera movement estimator algorithms developed was that they should be robust in the presence of outliers, whose detection will be seen in Section 5.3.2. For reasons having to do with tradition and ease of computation [111], the more often used estimation method is least squares, which is a type of M-estimator. However, this method is not robust: it has a breakdown point of 0, meaning that a single outlier may force the estimates outside an arbitrary range [111].

Even though the algorithms proposed here are based on the least squares estimator, robustness is pursued in different ways: the first algorithm relies on iterative Case Deletion [57] to successively remove data points (motion vectors) deemed to correspond to outliers, while the second relies on the Hough transform to perform a clustering of the data points. This clustering can be seen either as a simple form of segmentation or as method for removal of outliers. This last method is essentially a simplification of the method proposed first in [4].[4] However, the proposed algorithms include an intermediate motion vector smoothing step in the iterations which reduces the number of outliers that stem from the aperture problem, thus improving the estimates by increasing the number of motion vectors on which the least squares camera movement estimation is based.

The algorithms proposed here for estimating camera movement were designed so that they would be easily implementable. As such, they rely on block matching to obtain an initial, low resolution, sparse motion vector field, which is then used to estimate the camera movement factors z and d. Camera motion is thus estimated from an estimated motion vector field. This is similar to the methods of motion estimation and segmentation which rely on the optical flow [4], though in this case the motion vector field is very sparse.

5.3.1 Least squares estimation

If the estimated block motion vectors $\tilde{\mathcal{M}}_b[m,n]$ are assumed to have an independent 2D Gaussian distribution around the motion vectors $\mathcal{M}_b[m,n](z,d)$ given by the model of (5.5), with covariance matrix $C[m,n]$, then the maximum likelihood estimated values of the camera movement factors minimize

$$\sum_{m=0}^{B_L-1}\sum_{n=0}^{B_C-1}(\tilde{\mathcal{M}}_b[m,n]-\mathcal{M}_b[m,n](z,d))^T C^{-1}[m,n](\tilde{\mathcal{M}}_b[m,n]-\mathcal{M}_b[m,n](z,d)), \qquad (5.7)$$

which is obtained by taking the logarithm of the probability of the estimated motion vectors according to the distribution model.

Since $\mathcal{M}_b[m,n](z,d)$, given by (5.5), is linear on z and d, the minimization actually corresponds to the least squares solution of a linear equation (which can be derived from (5.7) by using Kronecker operators), whose properties have been introduced when discussing the flat and affine region models in Chapter 4.

[4]Hough transform-based estimation algorithms are related to the LMedS (Least Median of Squares) robust estimation method, see [111].

Simplifying the estimation

The covariance matrix $C[m,n]$ of the motion vectors $\tilde{\mathcal{M}}_b[m,n]$, estimated in this case through block matching, may be important for obtaining accurate estimates of the camera movement factors. Here, however, the covariance matrix is simplified. The estimation and use of a full covariance matrix remains as an issue for future work.

Instead of estimating the covariance matrix, we build one which intuitively makes sense, and which gives appropriate results in practice. There are two sources of errors for the least squares estimator if the covariance matrix is simplified to $C[m,n] = \sigma^2 I_2$, with I_2 the identity matrix:

1. blocks which do not correspond to camera movement, i.e., blocks encompassing one or several moving objects, or blocks containing uncovered areas (without a correspondence in the previous image), are given the same weight as blocks corresponding to camera movements; and
2. badly estimated motion vectors, namely those where the minimum error is in a very flat valley (shallow or wide) of the prediction error surface, are given the same weight as blocks with good motion vectors, where the minimum error is in a deep trough of that surface (this is related to the aperture problem already mentioned).

The first problem is related to motion vectors which most definitely do not have the Gaussian distribution around the model, as taken in the previous section: they are outliers. Outlier removal will also be attempted in the complete algorithm, but some outliers are likely to remain even after such removal. Fortunately, the cases corresponding to the first problem, which are missed by the outlier detection mechanisms, usually result in a large prediction error, at least larger than for blocks with pure camera movement. Hence, if the prediction error is used in place of the variance, better estimates result. The covariance matrix will thus be $C[m,n] = \tilde{E}[m,n]I_2$, with $\tilde{E}[m,n] = E[m,n](\tilde{\mathcal{M}}_b[m,n])$ (see (5.6)). In practice, however, since $\tilde{E}[m,n]$ may be zero in occasions, and $C[m,n]$ cannot be singular, the covariance matrix is taken to be $C[m,n] = (1 + \tilde{E}[m,n])I_2$.

The second problem has not been addressed here directly. A numerically sound approach might be to fit a paraboloid (with ellipsoidal section, and axes in any direction) to the error surface, centered in the minimum error, and take the parameters of a section of the paraboloid at a fixed height as the elements of the covariance matrix,[5] possibly scaled up or down according to the minimum prediction error, as in the previous paragraph. This would allow the algorithm to cope appropriately with narrow valleys of the prediction error by allowing the model of the estimated motion vectors to include a covariance, off diagonal, element. This approach was not followed, for questions related to the computational cost of the algorithms, and remains as an issue for future work.

[5]Remember that the equation of an ellipse may be written as

$$[x \quad y]\, C^{-1} \begin{bmatrix} x \\ y \end{bmatrix} = r,$$

with C positive definite.

Finally, it must be remembered that, by assigning the same variance to both components of the estimated motion vectors, as in $C[m,n] = (1+\tilde{E}[m,n])I_2$, the pixel aspect ratio is not taken into account, so that, when the pixels are not square, one direction is given more weight in the minimization than the other. To solve this problem, the covariance matrix will be taken to be

$$C[m,n] = (1+\tilde{E}[m,n]) \begin{bmatrix} \alpha^2 & 0 \\ 0 & 1 \end{bmatrix}.$$

If, as proposed before, the full covariance matrix is estimated by fitting a paraboloid to the prediction error surface, this correction is obviously not necessary.

Since any positive definite correlation matrix C of a 2D random variable can be rendered to the form $C = \sigma^2 I_2$ by transforming the random variable by a rotation θ and a scaling s of its (say) y axis, and thus can be described by σ, θ, and s, the approximation used here corresponds to setting θ to 0, s to the inverse of the pixel aspect ratio, and σ^2 to the prediction error (plus one).

Masking the blocks

As said before, not all blocks correspond to camera movements, and as such an essential step is the removal of outliers. Such a step can be taken as producing a mask matrix M, with B_L lines and B_C columns, which is zero if block $b[m,n]$ is an outlier and one otherwise. Using this mask, together with the approximation of $C[m,n]$ proposed in the last section, the expression to minimize can be written as

$$\sum_{m=0}^{B_L-1}\sum_{n=0}^{B_C-1} \frac{M[m,n]}{1+\tilde{E}[m,n]} \left(\tilde{\mathcal{M}}_b[m,n] - \mathcal{M}_b[m,n](z,d)\right)^T \begin{bmatrix} \alpha^{-2} & 0 \\ 0 & 1 \end{bmatrix} \left(\tilde{\mathcal{M}}_b[m,n] - \mathcal{M}_b[m,n](z,d)\right). \tag{5.8}$$

Solution and its uniqueness

Let

$$
\begin{aligned}
A &= \sum_{m=0}^{B_L-1}\sum_{n=0}^{B_C-1} \frac{M[m,n]}{1+\tilde{E}[m,n]}\left(\alpha^{-2}\left((m-\frac{B_L-1}{2})L_b\right)^2 + \left((n-\frac{B_C-1}{2})C_b\right)^2\right),\\
B_i &= \sum_{m=0}^{B_L-1}\sum_{n=0}^{B_C-1} \frac{M[m,n]}{1+\tilde{E}[m,n]}(m-\frac{B_L-1}{2})L_b,\\
B_j &= \sum_{m=0}^{B_L-1}\sum_{n=0}^{B_C-1} \frac{M[m,n]}{1+\tilde{E}[m,n]}(n-\frac{B_C-1}{2})C_b,\\
C &= \sum_{m=0}^{B_L-1}\sum_{n=0}^{B_C-1} \frac{M[m,n]}{1+\tilde{E}[m,n]}\left(\alpha^{-2}(m-\frac{B_L-1}{2})L_b\tilde{\mathcal{M}}_{b_i}[m,n] + (n-\frac{B_C-1}{2})C_b\tilde{\mathcal{M}}_{b_j}[m,n]\right),\\
D_i &= \sum_{m=0}^{B_L-1}\sum_{n=0}^{B_C-1} \frac{M[m,n]}{1+\tilde{E}[m,n]}\tilde{\mathcal{M}}_{b_i}[m,n],\\
D_j &= \sum_{m=0}^{B_L-1}\sum_{n=0}^{B_C-1} \frac{M[m,n]}{1+\tilde{E}[m,n]}\tilde{\mathcal{M}}_{b_j}[m,n], \text{ and}\\
S &= \sum_{m=0}^{B_L-1}\sum_{n=0}^{B_C-1} \frac{M[m,n]}{1+\tilde{E}[m,n]},
\end{aligned}
$$

where $\tilde{\mathcal{M}}_b[m,n] = \left[\tilde{\mathcal{M}}_{b_i}[m,n] \quad \tilde{\mathcal{M}}_{b_j}[m,n]\right]^T$. Then, if $S(AS-\alpha^{-2}B_i^2-B_j^2)\neq 0$, the minimum of (5.8) is unique and found at

$$\tilde{z} = \frac{CS-\alpha^{-2}B_iD_i}{AS-\alpha^{-2}B_i^2-B_j^2}, \tag{5.9}$$

$$\tilde{d}_i = \frac{SAD_i+B_iB_jD_j-B_j^2D_i-SB_iC}{S(AS-\alpha^{-2}B_i^2-B_j^2)}, \text{ and} \tag{5.10}$$

$$\tilde{d}_j = \frac{SAD_j+\alpha^{-2}(B_iB_jD_i-B_i^2D_j)-SB_jC}{S(AS-\alpha^{-2}B_i^2-B_j^2)}. \tag{5.11}$$

If $S(AS-\alpha^{-2}B_i^2-B_j^2)\neq 0$, then either $S=0$ and/or $AS-\alpha^{-2}B_i^2-B_j^2=0$. But $S=0$ only if the mask M is all null (since $\tilde{E}[m,n]\geq 0$), in which case there is no data, only outliers, rendering estimation impossible. On the other hand, if $AS-\alpha^{-2}B_i^2-B_j^2=0$, the solution ceases to be unique. The standard least squares algorithms, in cases of non-uniqueness, often return the smallest possible solution. In this case, however, the factors z and d do not have the same units, and hence such a solution is meaningless. Two interesting (possible) solutions are as follows:

1. Choose z as zero (no zoom), and find the corresponding pan factor. In this case the

solution is

$$\begin{aligned}\tilde{z} &= 0,\\ \tilde{d}_i &= \frac{D_i}{S}, \text{ and}\\ \tilde{d}_j &= \frac{D_j}{S}.\end{aligned}$$

2. Set z to its extrapolation $\hat{z}$ from the estimates of the previous images and solve for d

$$\begin{aligned}\tilde{z} &= \hat{z},\\ \tilde{d}_i &= \frac{D_i - B_i\hat{z}}{S}, \text{ and}\\ \tilde{d}_j &= \frac{D_j - B_j\hat{z}}{S}.\end{aligned}$$

   This solution attempts to maintain zoom movements as smooth as possible, since in practice zoom movements are slower than pan movements, especially for hand-held cameras, where vibration can be quite strong and zooming is performed through a motor, which is typically quite slow.

The former solution is the one used in the proposed algorithm. The latter has been left for future work.

The next sections show how the least squares estimation may be improved by using appropriate outlier detection methods and motion vector field smoothing algorithms.

5.3.2 Outlier detection

As said before, the motion vector field of an image relative to the previous one may contain motion vectors which do not correspond to camera movement, either because they contain objects with independent motion, or because they contain areas with no correspondence in the previous image.

Uncovered areas

Given an initial estimate of the camera movement factors, it is easy to classify those blocks that contain uncovered zones *because* of the camera movement. Simply reconstruct the motion vectors of each block (and round them to integer coordinates), using the initial estimate of the camera movement factors, and mark as containing uncovered background those for which the motion vector takes the reference block outside of the previous image. Since those motion vectors are likely to be wrongly estimated, this is a reasonable step to take.

Local motion

As to the blocks containing objects (or parts thereof) with independent motion, two different methods will be compared. The first method relies on simple comparison of the estimated motion vectors with the ones predicted by the estimated camera motion factors, was already proposed in [122]. The second method is based in the concept of the Hough transform. While the first method may be seen as an iterative Case Deletion method for increasing the robustness of the least squares estimator [57], the second performs a clustering of the data points in the Hough transform parameter space, from which a rough segmentation of the data points into two disjoint sets (valid data points and outliers) can be obtained. It is a simple version of the method proposed in [4].

A distance based approach

This method uses a simple thresholding technique for classifying blocks as possessing local motion or not. Given the initial estimates $\tilde{z}$ and $\tilde{d}$ of the camera movement factors, the estimated motion vectors $\tilde{\mathcal{M}}_b[m,n]$ are compared with the ones predicted by the model, i.e., with $\mathcal{M}[m,n](\tilde{z},\tilde{d})$. If $\|\tilde{\mathcal{M}}_b[m,n] - \mathcal{M}[m,n](\tilde{z},\tilde{d})\|/\|\mathcal{M}[m,n](\tilde{z},\tilde{d})\| > t$, where t is a given threshold, then the block is deemed to possess local motion and classified as such. When $\|\mathcal{M}[m,n](\tilde{z},\tilde{d})\| = 0$, the test is performed not on the relative size of the difference between the motion vectors but on the absolute size of the estimated motion vector itself, i.e., if $\|\tilde{\mathcal{M}}_b[m,n]\| > t_2$, where t_2 is another threshold, the block is deemed to possess local motion. The threshold t_2 is taken, in the algorithm, to equal $10t$, so as to render the method dependent of single parameter. This relation between t_2 and t was chosen empirically and gives acceptable results in practice.

Finally, it should be mentioned that, in both cases, the norms are calculated on the motion vectors expressed in site coordinates, so as to take the pixel aspect ratio into account.

A Hough transform approach

This approach is very similar to the one in [4], even though much simpler and adapted to the camera movement model used in this thesis, and thus more efficient.

The algorithms for camera movement estimation proposed make a simple assumption about the movement, as will be seen: if more than 40% of the blocks in an image can be described adequately by a set of camera motion factors and no other larger group of blocks exists in the same circumstances, then those blocks are taken to represent static background and the estimated factors to represent the actual camera movement. This assumption can obviously fail at times, but it is simple and gives good results in practice. However, the method described previously can have some difficulties with this assumption, since the initial factors are estimated *before* outlier removal, and thus can fall midway between two different motions in a scene, probably resulting in the classification of nearly all blocks as outliers. In order to solve this problem, a different procedure is used here, which in fact intertwines a rough estimation with outlier detection. It uses the concept of Hough transform, which can be found on any image processing textbook [56].

Let the space of the possible camera movement factors be bounded and divided into accumulator cells, here called bins for simplicity. Since the pan factor components d_i and d_j are directly related with the motion vectors in case of a zoomless movement, they will be bounded to $[-d_{i_{\max}}, d_{i_{\max}}]$ and $[-d_{j_{\max}}, d_{j_{\max}}]$, respectively. As to the zoom factor, it will be bounded to $[-0.2, 0.2]$, based on empirical evidence about the range of typical zoom movements (and also because, for CIF images, used in this chapter, these values result in motion vectors for the outer blocks which exceed the maximum horizontal displacement of $d_{j_{\max}} = 30$ used). The number of bins was chosen so as to divide this bounded region into bins small enough to provide a rough estimate of the factors, and large enough to render the Hough transform approach useful, by allowing the Hough transform of estimated motion vectors corresponding to a single set of camera movement model parameters to concentrate in a single bin. The following empirical values where found to yield good results: 41 bins for z, and 42 bins for both d_i and d_j, totalling 72324 bins. If an initial estimate of the camera movement factors is available, these bins are offset so that the motion vectors generated by these initial factors all coincide in the center of a single bin. This contributes to avoid errors due to a dispersion of the Hough transformed motion vectors among two, four, or even among eight bins.

Let H be a 3D accumulator array with 41 planes of 42×42 bins. For each estimated motion vector $\tilde{\mathcal{M}}_b[m,n]$, the zoom factors z corresponding to the center of each bin are tried. Given the model (5.5) this results in

$$d_i = \tilde{\mathcal{M}}_{b_i}[m,n] - z\left(m - \frac{B_L - 1}{2}\right) L_b, \text{ and}$$
$$d_j = \tilde{\mathcal{M}}_{b_j}[m,n] - z\left(n - \frac{B_C - 1}{2}\right) C_b.$$

The bin in H corresponding to the zoom factor being currently tried and to the pan factor components calculated above is incremented. After processing all estimated motion vectors, the center of the bin containing the largest value (which can be tracked dynamically while filling H) is taken as a rough estimate of the camera movement factors. Blocks whose estimated motion vectors contributed to that bin or to a bin closer than a given threshold t_h (using a chess-board distance, i.e., the maximum of the absolute values, expressed in number of bins), are deemed to agree with the estimated camera movement factors. Even though a rough estimate is calculated by this method, it is discarded, since the mask of valid blocks will be later used to estimate, with the least squares estimator, more accurate values for those factors.

Those blocks which where deemed to contain uncovered areas are not affected by the procedure above.

Zones of the image, encompassing more than one block, and possessing motion which cannot be described by the assumed model will tend to disperse their contribution to the Hough transform through a large number of bins in H. On the contrary, zones whose motion is describable by the model will tend to concentrate on the bin corresponding to the model parameters. By selecting the maximal bin, the largest of these model compliant zones will be chosen. If no such zone exists, the chosen maximum will be small, leading to a mask with few valid blocks. Later parts of the algorithm will detect this and either attempt to relax the threshold t_h or quit the estimation, if further relaxation is not acceptable.

Finally, it should be noticed that if pan movements only are being estimated, this method reduces to selecting the maximum of the histogram of motion vectors, which was proposed in [110].

5.3.3 Smoothing

Whenever there are zones with a relatively uniform color or pattern, e.g. two colors separated by a straight boundary, the motion vectors estimated by the block matching algorithm will tend to be erroneous. This is the so called aperture problem. Such cases might be partially dealt with by estimating the covariance matrices $C[m, n]$ and using them in the least squares estimation. As this has not been done, for reasons of efficiency, some other method of dealing with these errors has to be used. It is true that part of these errors would probably be captured by the outlier detection part of the algorithm, but at the cost of estimates for the camera movement factors which would be based on a smaller number of data points. In some situations, that might even render estimation of camera movement impossible, for lack of enough blocks to perform the estimate (40% of the total).

In order to solve these problems, a simple method for regularizing the estimated motion vector field has been devised. Given estimates $\tilde{z}$ and $\tilde{d}$ of the camera movement factors, a corresponding motion vector field is constructed, as in equation (5.5), though rounded to integer coordinates. Let $\bar{\mathcal{M}}_b[m, n](\tilde{z}, \tilde{d})$ be that vector field. Then, of the set of possible motion vectors $\mathcal{M}_b[m, n]$ which are closer to $\bar{\mathcal{M}}_b[m, n](\tilde{z}, \tilde{d})$ than $\tilde{\mathcal{M}}_b[m, n]$ is, and whose prediction error is small enough, namely $E[m, n](\mathcal{M}_b[m, n]) \leq (\tilde{E}[m, n] + 1)(1 + t_s)$, the one which is closer to $\bar{\mathcal{M}}_b[m, n](\tilde{z}, \tilde{d})$ is chosen as the new, smoothed estimate. If there is a tie, the motion vector with the smaller prediction error is chosen. If again there is a tie, the one closer to the original estimate $\tilde{\mathcal{M}}_b[m, n]$ is chosen. If no such vector is found, the estimate is left as is.

The smoothing threshold t_s controls how much increase in the prediction error is allowable while manipulating the estimated motion vector. An empirical value of 6% has been used in this thesis. As to the sum of 1 to the minimum prediction error, it allows some smoothing to occur even if the minimum prediction error is zero.

A different approach to motion vector smoothing, using a Gibbs model, can be found in [185].

5.4 The complete algorithms

Two algorithms are proposed here. The first, see Algorithm 1, is based on the original proposal of [122]. It will be called the "old algorithm", and it uses the distance based approach to the detection of outliers. The second, see Algorithm 2, is an improvement which uses the Hough transform approach to outlier detection. It will be called "new algorithm".

Both algorithms use two loops. The inner loop performs camera motion estimation and outlier

Algorithm 1 Estimation of the camera motion factors using the old algorithm (based on [122]).

```
Require: $t_0 \geq 0$ {initial outlier detection threshold}
Require: $\Delta t \geq 0$ {outlier detection threshold increment}
Require: $t_{\max} \geq t_0$ {maximum outlier detection threshold}
Require: $t_s \geq 0$ {smoothing threshold}
Require: $0 \leq t_a \leq 1$ {camera movement detection threshold}
Require: $i_{\max} > 0$ {maximum inner iterations}
Require: $\alpha > 0$ {pixel aspect ratio}
Require: $L_b > 0$ {number of lines per block}
Require: $C_b > 0$ {number of columns per block}
Require: $B_L > 0$ {number of lines of blocks}
Require: $B_C > 0$ {number of columns of blocks}
Require: $\tilde{\mathcal{M}}_b$ has $B_L \times B_C$ motion vectors
Require: $|\tilde{\mathcal{M}}_{b_i}[m,n]| \leq d_{i_{\max}}, \forall m \in \{0,\ldots,B_L-1\}, \forall n \in \{0,\ldots,B_C-1\}$
Require: $|\tilde{\mathcal{M}}_{b_j}[m,n]| \leq d_{j_{\max}}, \forall m \in \{0,\ldots,B_L-1\}, \forall n \in \{0,\ldots,B_C-1\}$
Ensure: either "found" is false, and no camera movement has been found, or $\tilde{z}$ and $\tilde{d}$ are
  estimates of the camera movement factors
  $t = t_0$ {initialize outlier detection threshold}
  repeat
    $\mathcal{M}_b \leftarrow \tilde{\mathcal{M}}_b$ {copy estimated motion vector field}
    $E \leftarrow \tilde{E}$ {copy prediction errors corresponding to $\tilde{\mathcal{M}}_b$}
    $M \leftarrow 1$ {fill $M$ with ones (valid)}
    border($M$) {border blocks are set to zero (outlier) in $M$ (improves initial estimates in case
    of a zoom out or a non-null pan)}
    $i \leftarrow 0$ {initialize number of inner iterations}
    repeat
      $\tilde{z}, \tilde{d} \leftarrow$ lse($\mathcal{M}_b, E, M, L_b, C_b, \alpha$) {estimate zoom and pan factors using least squares}
      build $\bar{\mathcal{M}}_b(\tilde{z}, \tilde{d})$ {build rounded motion vector field from $\tilde{z}$ and $\tilde{d}$}
      $\mathcal{M}_b \leftarrow$ smooth($\tilde{\mathcal{M}}_b, \bar{\mathcal{M}}_b(\tilde{z}, \tilde{d}), t_s$) {smooth $\tilde{\mathcal{M}}_b$ towards $\bar{\mathcal{M}}_b(\tilde{z}, \tilde{d})$ with threshold $t_s$}
      changes, $M \leftarrow$ outliers($M, \bar{\mathcal{M}}_b(\tilde{z}, \tilde{d}), \alpha, t$) {mark uncovered blocks in $M$ and local motion
      blocks (distance based approach), set "changed" according to whether any block had its
      mask changed}
      $i \leftarrow i + 1$
    until not changed or $i = i_{\max}$
    found $\leftarrow$ (valid($M$) $> t_a B_L B_C$) {set "found" according to whether there are enough valid
    (non-outlier) blocks}
    $t \leftarrow t + \Delta t$ {increment outlier detection threshold}
  until found or $t > t_{\max}$
```

Algorithm 2 Estimation of the camera motion factors using the new algorithm.

```
Require: $t_{h_{\max}} \geq 0$ {maximum outlier detection threshold}
Require: $t_s \geq 0$ {smoothing threshold}
Require: $0 \leq t_a \leq 1$ {camera movement detection threshold}
Require: $i_{\max} > 0$ {maximum inner iterations}
Require: $\alpha > 0$ {pixel aspect ratio}
Require: $L_b > 0$ {number of lines per block}
Require: $C_b > 0$ {number of columns per block}
Require: $B_L > 0$ {number of lines of blocks}
Require: $B_C > 0$ {number of columns of blocks}
Require: $\tilde{\mathcal{M}}_b$ has $B_L \times B_C$ motion vectors
Require: $|\tilde{\mathcal{M}}_{b_i}[m,n]| \leq d_{i_{\max}}, \forall m \in \{0,\ldots,B_L-1\}, \forall n \in \{0,\ldots,B_C-1\}$
Require: $|\tilde{\mathcal{M}}_{b_j}[m,n]| \leq d_{j_{\max}}, \forall m \in \{0,\ldots,B_L-1\}, \forall n \in \{0,\ldots,B_C-1\}$
Ensure: either "found" is false, and no camera movement has been found, or $\tilde{z}$ and $\tilde{d}$ are
  estimates of the camera movement factors
  $t_h \leftarrow 0$ {initialize outlier detection threshold}
  $\tilde{z} \leftarrow 0$ {initial estimate: no zoom}
  $\tilde{d} \leftarrow 0$ {initial estimate: no pan}
  repeat
    $\mathcal{M}_b \leftarrow \tilde{\mathcal{M}}_b$ {copy estimated motion vector field}
    $E \leftarrow \tilde{E}$ {copy prediction errors corresponding to $\tilde{\mathcal{M}}_b$}
    $M \leftarrow 1$ {fill $M$ with ones (valid)}
    border($M$) {border blocks are set to zero (outlier) in $M$ (improves initial estimates in case
    of a zoom out or a non-null pan)}
    $i \leftarrow 0$ {initialize number of inner iterations}
    repeat
      changes, $M \leftarrow$ hough($\mathcal{M}_b, E, \tilde{z}, \tilde{d}, L_b, C_b, t_h$) {perform outlier detection, set "changed"
      according to whether any block had its mask changed}
      $\tilde{z}, \tilde{d} \leftarrow$ lse($\mathcal{M}_b, E, M, L_b, C_b, \alpha$) {estimate zoom and pan factors using least squares}
      build $\bar{\mathcal{M}}_b(\tilde{z}, \tilde{d})$ {build rounded motion vector field from $\tilde{z}$ and $\tilde{d}$}
      $\mathcal{M}_b \leftarrow$ smooth($\tilde{\mathcal{M}}_b, \bar{\mathcal{M}}_b(\tilde{z}, \tilde{d}), t_s$) {smooth $\tilde{\mathcal{M}}_b$ towards $\bar{\mathcal{M}}_b(\tilde{z}, \tilde{d})$ with threshold $t_s$}
      changes, $M \leftarrow$ uncovered($M, \bar{\mathcal{M}}_b(\tilde{z}, \tilde{d}), \alpha, t$) {mark uncovered area blocks (outliers) in $M$
      and set "changed" to true if any block had its mask changed}
      $i \leftarrow i + 1$
    until not changed or $i = i_{\max}$
    found $\leftarrow$ (valid($M$) $> t_a B_L B_C$) {set "found" according to whether there are enough valid
    (non-outlier) blocks}
    $t_h \leftarrow t_h + 1$ {increment outlier detection threshold}
  until found or $t_h > t_{h_{\max}}$
```

detection until the mask of valid blocks remains unchanged or until the limit number of inner iterations is attained. In both cases the mask is not guaranteed to converge, hence the need for a limited number of iterations. Notice that the order of estimation and outlier detection is inverted: in the old algorithm estimation is performed first, while in the new algorithm outlier detection is performed first. Of course, as noticed before, outlier detection in the new algorithm actually involves roughly estimating the movement parameters before detecting outliers, so that, for each inner iteration on the new algorithm, two estimations are performed. The advantage is that the outlier detection using the Hough transform is much more robust, since it explicitly selects the most probable bin. The use of the least squares estimation to obtain the final results is due to the lack of precision of the Hough estimator. This lack of precision is due to the fact that the parameter space of the Hough transform has been coarsely quantized to guarantee meaningful results for the relatively small number of data points used: the camera movement estimates are based on block matching results on a coarse 16×16 grid.

The outer loop checks whether the number of valid blocks on which the estimate is based is sufficient to consider that camera movements are present in the current image (relative to the previous). If not, it increments the threshold over which blocks are considered outliers in outlier detection (but only the part relative to local motion, not to uncovered areas), thereby relaxing the requirements until either camera motion is considered present or the maximum relaxation is attained. With this outer loop, it is possible to estimate camera motion factors as accurately as possible, by starting with stringent requirements and finishing with more relaxed ones. If the maximum relaxation is attained before camera movement is detected, the algorithms fails. Since even images possessing zero pan and zoom factors can be classified as having camera movement, the algorithm failure may mean, in both cases, that the scene is too complex for the algorithms or that a scene cut has been reached (the current image is unrelated with the previous).

Both algorithms suffer from an inherent limitation of accuracy: both are based on pixel level block motion vectors. The results are thus likely to suffer from errors, especially for very small pan movements without any zooming, which may in fact be estimated as no movement at all. The accuracy may be improved by using block matching methods with sub-pixel accuracy. This was not tried here, however.

5.4.1 Results

Experimental conditions

The algorithms have been tested with the parameters shown in Tables 5.1(a), 5.1(b) and 5.1(c).

Estimating zoom offset

The "Table Tennis" sequence consists of two shots. The second, from image 131 on, contains no camera movement, and thus will not be of great interest here. From images 0 to 130, though, there is a zoom movement. It starts slowly around image 20, increases its speed, then slowly

L	288	CIF lines
C	352	CIF columns, or pixels per line
L_b	16	pixels
C_b	16	pixels
B_L	18	blocks
B_C	22	blocks
$d_{i_{\max}}$	32	pixels
$d_{j_{\max}}$	30	pixels
α	1.0(6)	16/15 (see Section A.1.1)

(a) Common format parameters.

t_0	0.1	10%
Δt	0.1	10%
$t_{\max}$	0.5	50%
t_s	0.06	6%
t_a	0.4	40%
$i_{\max}$	15	iterations

(b) Old algorithm.

$t_{h_{\max}}$	1	Hough accumulator bins
t_s	0.06	6%
t_a	0.4	40%
$i_{\max}$	15	iterations

(c) New algorithm.

Table 5.1: Algorithms parameters.

decreases towards its end, around image 107. The zoom movement is only clear, though, from images 24 (23 to 24) to 106 (105 to 106). Note that this zoom movement is not accompanied by any pan movement, as can be readily seen by direct observation of the sequence.

Figures 5.1 and 5.2 show the estimated camera movement factors of the "Table Tennis" sequence using the old and the new algorithms, respectively.

In Figures 5.1(b) and 5.2(b) it can be seen that, despite the fact that there is no panning in the original sequence, non-zero pan factor components were detected. It should be noticed, however, that in the case of the new algorithm these factors are non-zero only when there is zooming in the sequence. In the case of the old algorithm, some spurious zoom and pan factors occur out of the 24 to 106 image range, but this is due to the lower quality of its estimation. Thus, from the results of the new algorithm, in Figure 5.2(b), it can be inferred that the center of scaling of the zoom is offset from the center of the image.

Let the center of zoom be offset from the center of the image. Let d_z^s be this offset. After a pure zoom with scaling factor Z offset by d_z^s, a site s in the image changes its position to s', where

$$s' = Z(s - d_z^s) + d_z^s = Z\left(s - \left(1 - \frac{1}{Z}\right)d_z^s\right). \tag{5.12}$$

Comparing with (5.3), it can be seen that this offset zoom is equivalent to a pan plus zoom with scaling Z and translation $d^s = (1 - 1/Z)d_z^s$. Converting to the usual camera movement factors z and d

$$d = -\begin{bmatrix} 0 & -\alpha \\ 1 & 0 \end{bmatrix} z d_z^s = -z d_z,$$

where d_z is the offset expressed in pixel coordinates.

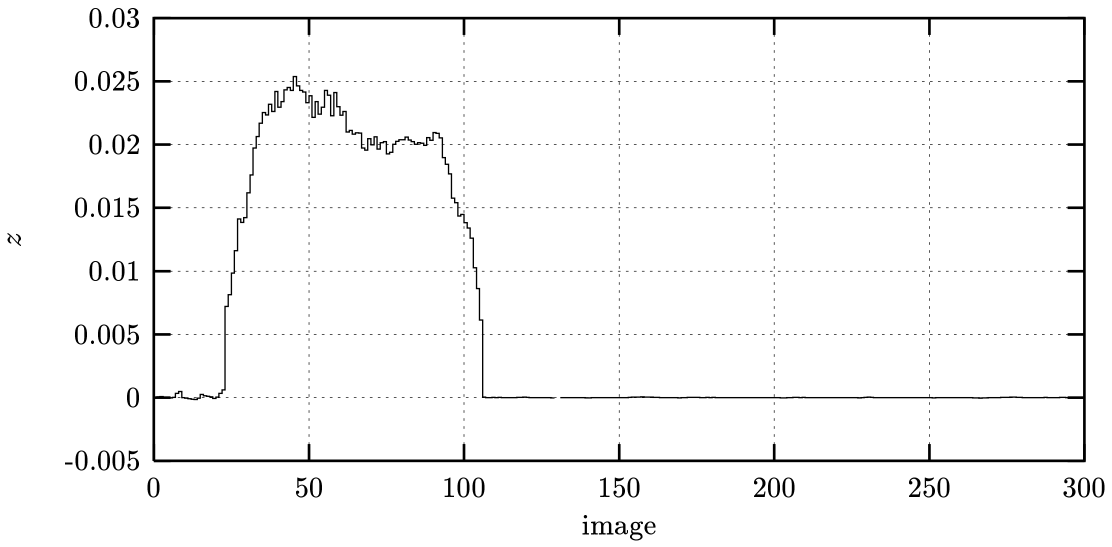
Hence, the estimated pan factor should vary linearly with the zoom factor. By observing Figures 5.2(a) and 5.2(b), it can be seen that this behavior is approximately true, especially for the d_i component of the pan factor, which attains larger values, and hence suffers less from estimation errors.

Using linear regression, it is possible to estimate the zoom offset in pixel coordinates d_z from the estimated values of z and d for images 24 to 106. The value obtained for d_z was

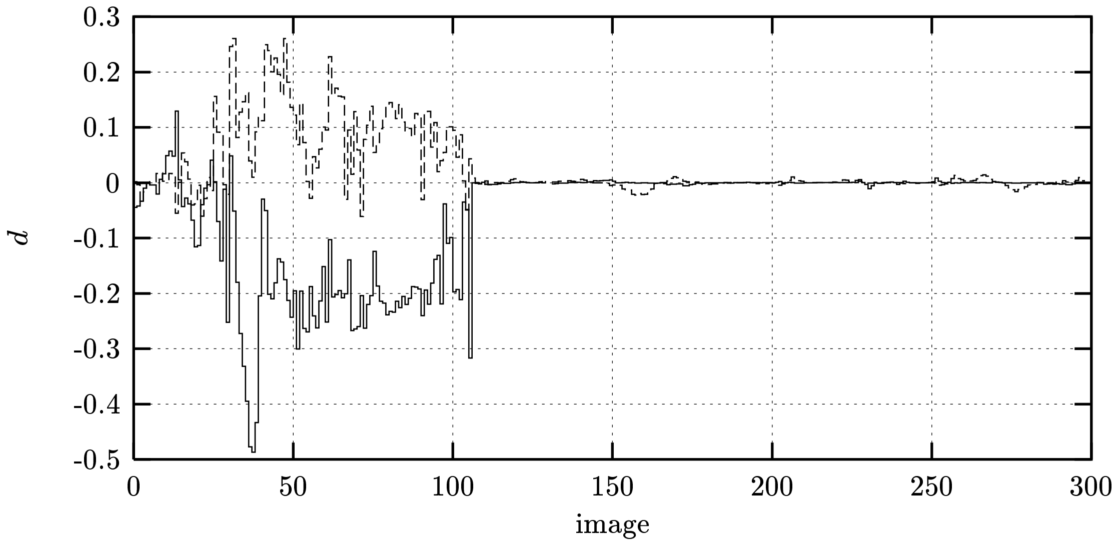
$$d_z = \begin{bmatrix} 10.3116 \\ -4.16089 \end{bmatrix},$$

whose estimated standard deviations (see [165, §15.2]) are 1.34175 and 1.03135, respectively. These values agree reasonably well with direct observation, which yielded approximately $d_z = \begin{bmatrix} 7 & -3.5 \end{bmatrix}^T$. It can be concluded that, even though camera movement estimation is based on block matching results, with pixel accuracy, it does yield estimation results with sub-pixel accuracy (at least when there are strong zoom movements). An issue which remained for future work is the quantification of the errors in the camera movement factors estimated.

Given the estimated zoom and pan factors and using (5.3) repeatedly, it is possible to estimate the position in each image of a point originally in the center of the image plan. Figure 5.3

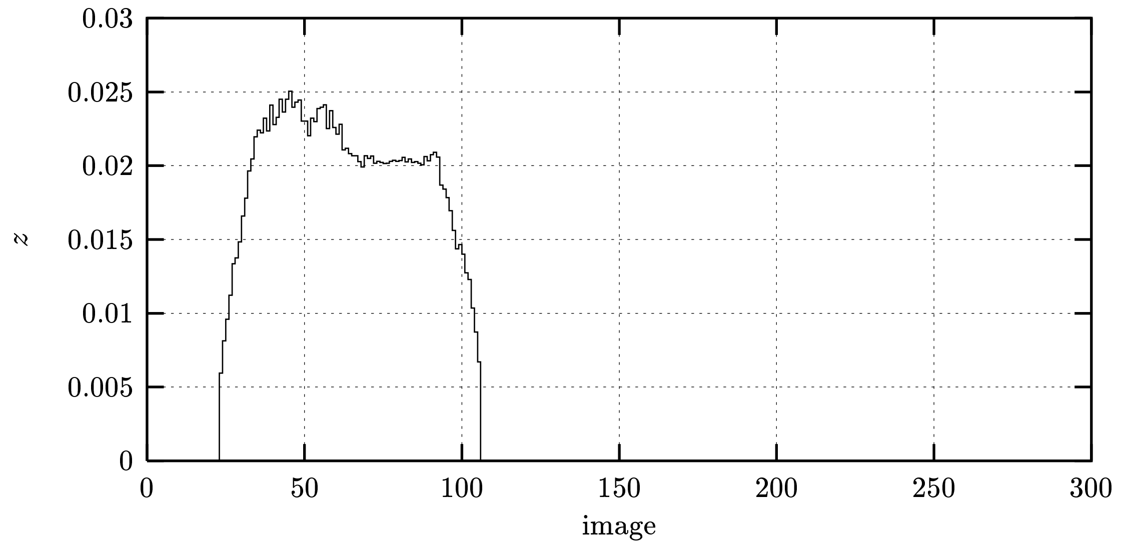


(a) Zoom factor.

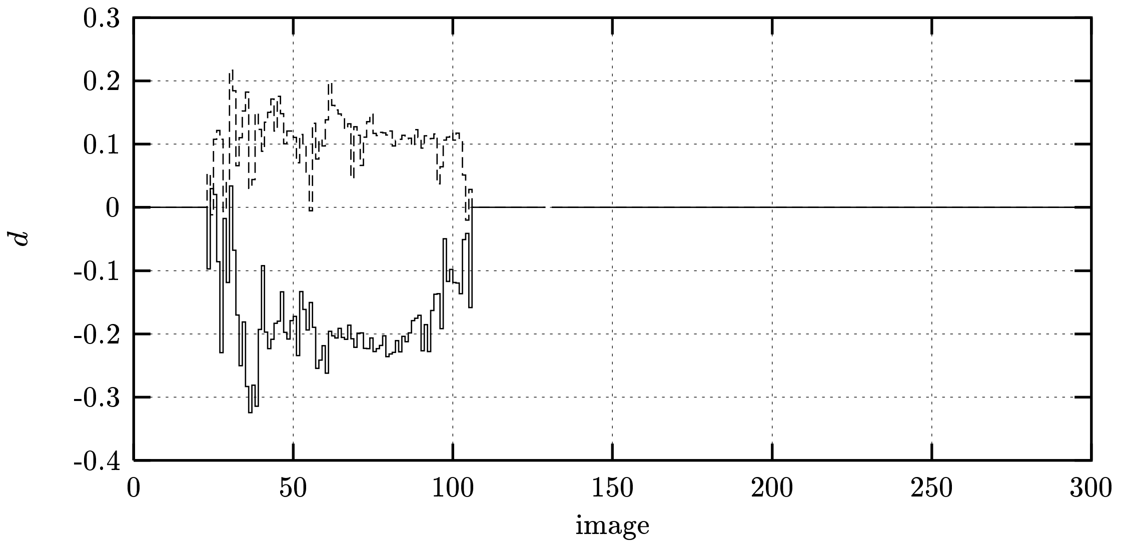


(b) Pan factor (d_i solid line, d_j broken line).

Figure 5.1: "Table Tennis": camera movement estimation results (old algorithm). Camera movement has not been detected in image 131, where a scene cut occurs.



(a) Zoom factor.



(b) Pan factor (d_i solid line, d_j broken line).

Figure 5.2: "Table Tennis": camera movement estimation results (new algorithm). Camera movement has not been detected in image 131, where a scene cut occurs.

shows this estimate, together with the same estimate using (5.12) with the value obtained for d_z by linear regression. It can be seen that the factors estimated do lead to an approximately linear evolution of the center, which further corroborates the hypothesis that the pan is due to an offset in the center of zoom.

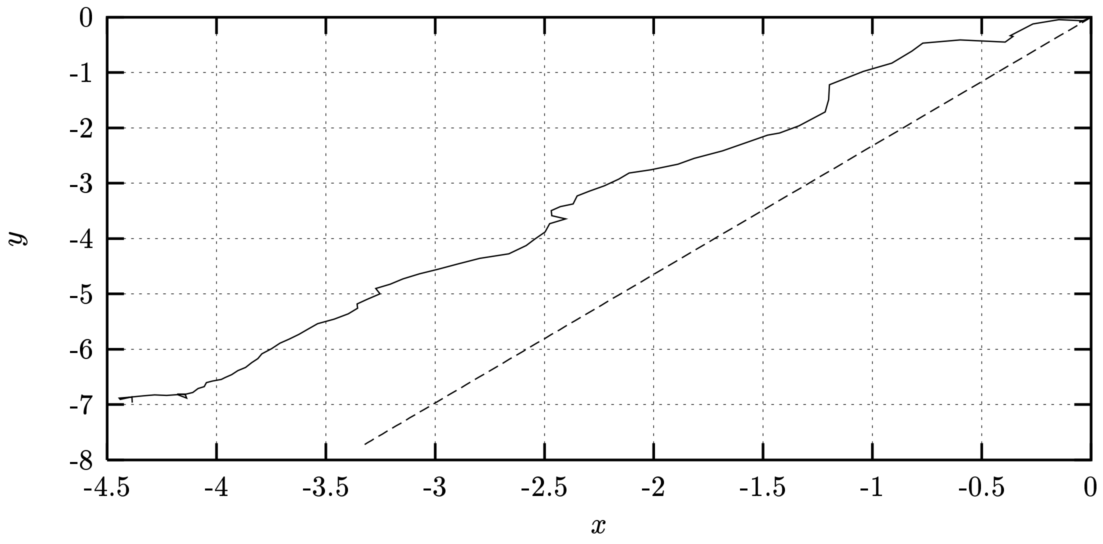


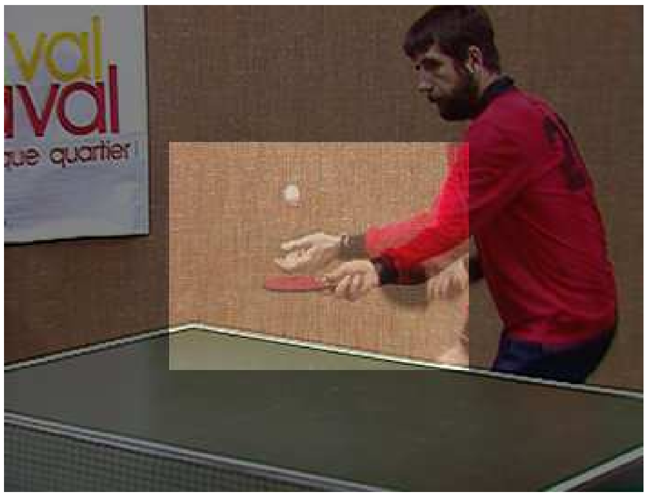
Figure 5.3: "Table Tennis": evolution of the center of the image plan from image 23 to image 106 using (5.3), solid line, and (5.12), broken line. Factors estimated using the new algorithm.

Figure 5.4 shows the result of transforming the first image of the "Table Tennis" sequence using the composition of all estimated camera movement factors up to images 60, in the middle of the zoom movement, and 124, well after the end of the zoom movement. The result of these two transformations of the first image is then overlapped with the original images 60 and 124. It is clear that the size of the transformed images seems to agree very well with the corresponding original image. There is, though, a slight displacement in its position, of the order of one or two pixels. The good agreement in image size seems to indicate that the zoom factors are being accurately estimated (it must be remembered that the transformations integrate about 36 and 80 fractional zoom factors, respectively). The slight offset in position can be seen to be of the order of the error between the two lines in Figure 5.3.

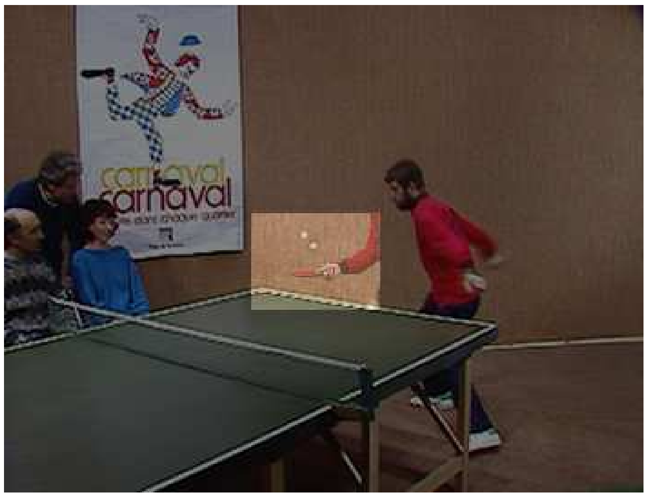
Comparison of the algorithms

Scene cuts

As said before, the failure of detection of camera movement can occur for several reasons, one of them being the occurrence of a scene cut. The robustness of the algorithms can be ascertained by checking their ability to detect real scene cuts, and by checking the number of false scene cut detections generated.



(a) Overlap with image 60.



(b) Overlap with image 124.

Figure 5.4: "Table Tennis": overlap of the original images with image 0 transformed according to the camera movement factors estimated by the new algorithm.

Several sequences have been tested, namely “Carphone”, “Coastguard”, “Flower Garden”, “Foreman”, “Stefan”, “Table Tennis”, and “VTPH”. Of these sequences, only “Table Tennis” and “VTPH” contain scene cuts. In “Table Tennis” two shots of a game are shown, the scene cut taking place from image 130 to image 131. In the case of “VTPH”, two scene cuts occur from image 79 to image 80 and from image 130 to image 131. In the following, scene cuts will be identified by the number of their second image.

Tables 5.2(a) and 5.2(b) show the occurrence of erroneous detections (i.e., detection of camera motion at scene cuts) or non-detections (i.e., failure to detect camera movement at normal, non-scene cut images) of camera movement in the test sequences.

	detections	non-detections
“Carphone”		203, 312, and 314
“Flower Garden”		62
“Foreman”		47, 49, and 153 to 157
‘Stefan”		2, 9, 65, 66, 85, 133, 141, 171, 172, 211, 214, 216, and 286
“VTPH”	80	86, 93, 102, 103, 110, and 111

(a) Old algorithm.

	detections	non-detections
“Foreman”		156, 191, and 192

(b) New algorithm.

Table 5.2: Camera movement erroneous detections and non-detections.

The superior performance of the new algorithm is clear. The old algorithm frequently fails to detect camera movement where there is no scene cut. But it also fails to detect a true scene cut at image 80 of the “VTPH” sequence. Hence, it can be said that its results would not be improved by allowing further relaxation on the outer loop, since this would lead to less erroneous non-detections but also to further erroneous detections. On the other hand, the new algorithm correctly classifies the three scene cuts and only fails to detect camera movement at three normal images of the “Foreman” sequence. In image 156, this failure is due to the movement of the hand in front of the camera. In images 191 and 192, it seems to stem from a badly estimated motion vector field, which is due to a strong pan movement with motion blurring.

Accuracy

Both algorithms are based on the same least squares estimator, so the accuracy of the result is strongly dependent on the outlier detection mechanism used. The previous results clearly show that the outlier mechanism of the old algorithm leads to frequent erroneous detections or non-detections of camera movement. But even in cases where camera movement is present and detected, the outlier detection mechanism can lead to poor quality estimates. Figures 5.5 and 5.6 show the camera movement factors estimated by both algorithms for the "Stefan" sequence.

The regularity in the time evolution of the factors for the new algorithm is not a coincidence. It is not imposed by the algorithm itself, since the algorithm operates independently for each image of the sequence. Hence, it can only stem from the regularity of the true camera movements (it is a TV sequence, and TV operators usually try to achieve smooth camera movement, especially when shooting sport images). The conclusion can only be that the results of the old algorithm are much worst, since the estimated factors have a quite rough time evolution.

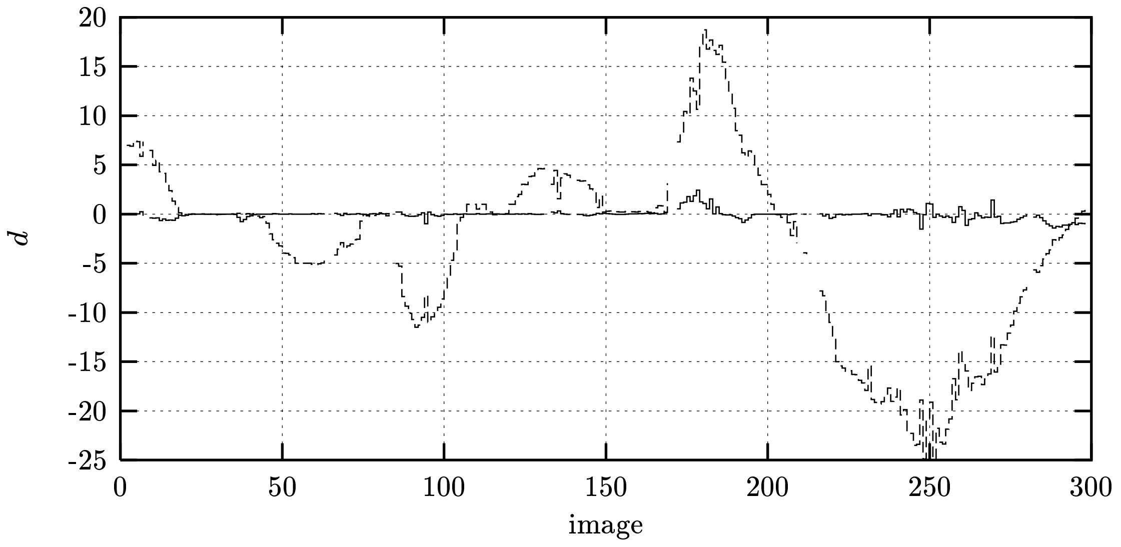
In order to ascertain the accuracy of the new algorithm, the estimated camera motion factors were integrated along time and the integrated values were used to move a box which was hand positioned over a particular feature of the scene present on the first image. Figures 5.7 and 5.8 show the box in the first image of the test sequences used (in this case "Coastguard", "Flower Garden", "Foreman", and "Stefan"), and the same box displaced and scaled according to the integration of the estimated camera movement factors in a posterior image (close to the last image where the feature is visible in the sequence and before any non-detections of camera movement). The positioning of the box relative to the corresponding feature can be used as a measure of the accuracy of the estimation, remembering that estimation errors are accumulated in the integration procedure.

For the "Coastguard" sequence, the box position offset is approximately 20 pixels horizontally and 10 pixels vertically. Taking into account that the box has been positioned according to the integration of the camera movement factors obtained for each image with reference to the previous, and also taking into account that this offset is obtained after 200 images, the result can only be classified as good. Besides, the sequence is not trivial, since it contains two objects with independent motion (the two boats), and the water with its random motion.

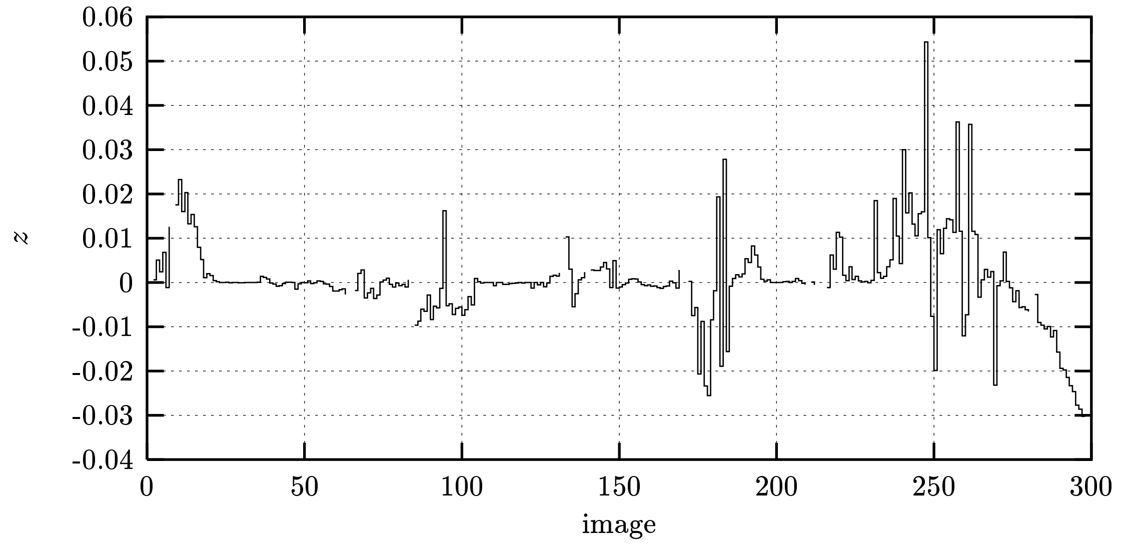
As to the "Flower Garden" sequence, the offset is of about 20 pixels in both directions for a range of 125 images. In this case, though, the camera movement is neither a pan nor a zoom: it is a traveling movement, in which the camera changes its positions. Hence, the motion in the projected image cannot be described by the model used and depends on the relative distance of the objects: objects which are closer move faster and objects which are farther move slower. This justifies the horizontal offset, since the algorithm takes the whole image into account. Also, some objects can have a projected vertical motion even if the camera moves horizontally. This is the case of window, and it justifies the vertical offset in its position. Nevertheless, it can be said that the algorithm performs reasonably well even when the true camera movement does not truly consist of zooming and panning.

In the "Foreman" sequence the offset is very small. The sequence is not simple since it can be seen to possess a composition of pan movements with small rotation movements about the lens axis which affects the estimation algorithm.

Finally, the "Stefan" sequence contains two difficult parts for the algorithms: small movements in the spectators, and the rather uniform game floor, for which block matching yields erroneous results. Nevertheless, the position offset after 244 images is small, of the order of 20 pixels horizontally, and negligible in the vertical direction.

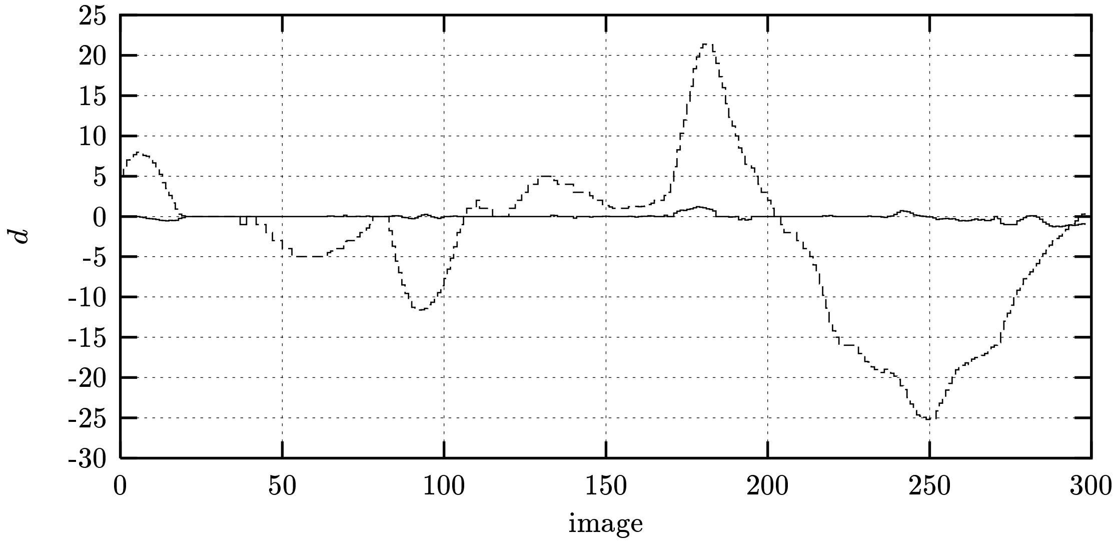


(a) Pan factor.

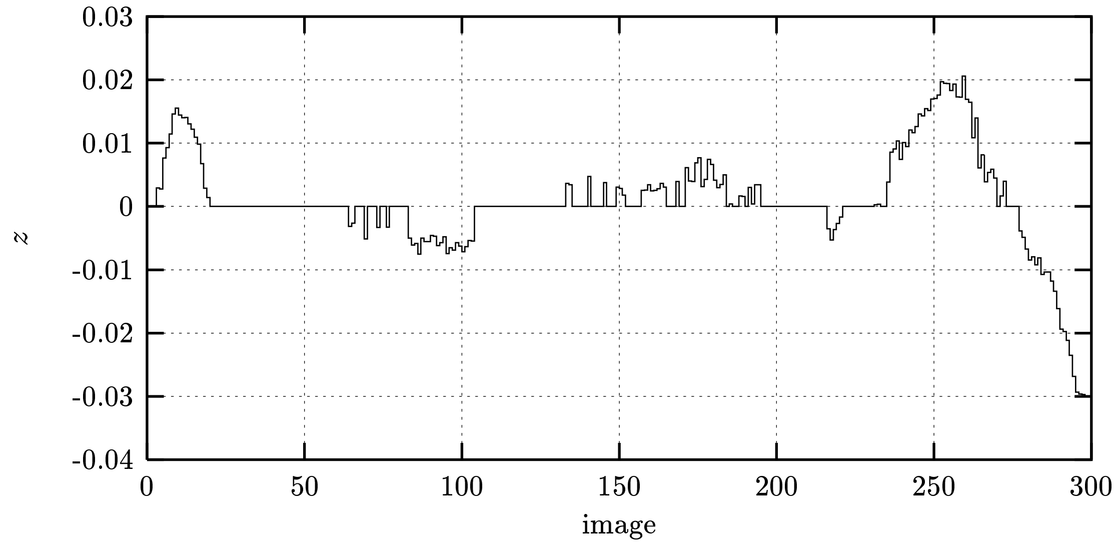


(b) Zoom factor.

Figure 5.5: "Stefan": camera movement factors estimated by the old algorithm (shown only for images with detected camera movement).

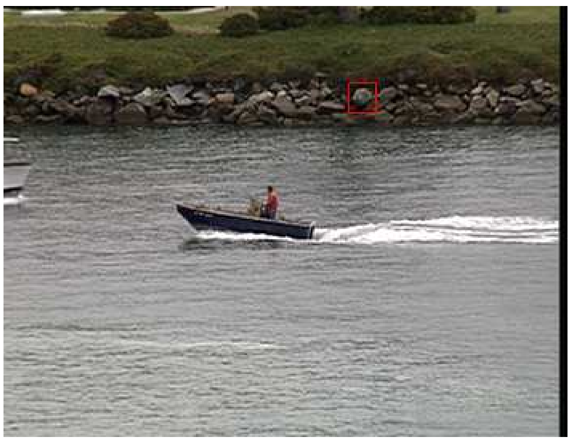


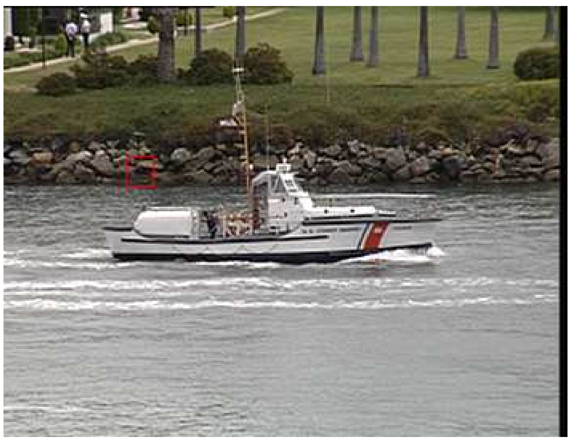
(a) Pan factor.

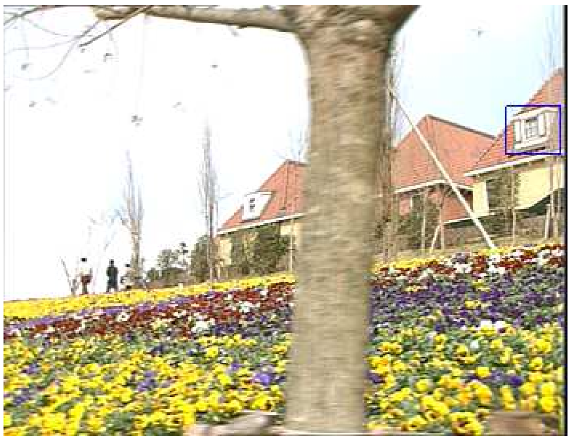


(b) Zoom factor.

Figure 5.6: "Stefan": camera movement factors estimated by the new algorithm.


(a) “Coastguard” image 0.


(b) “Coastguard” image 200.


(c) “Flower Garden” image 0.

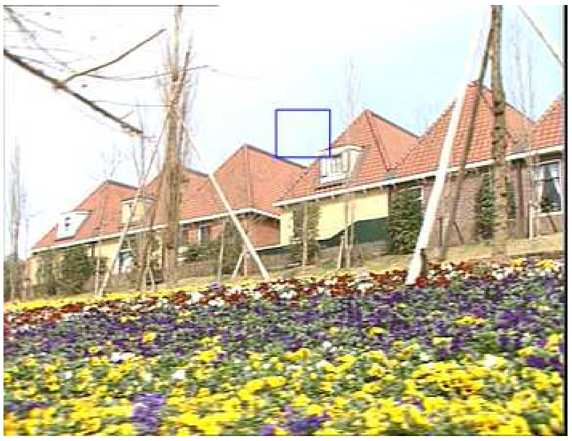
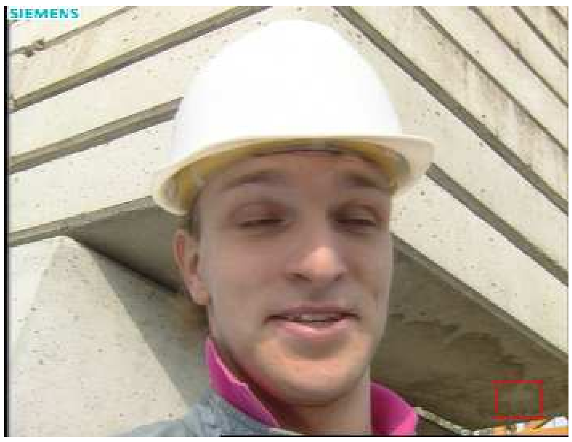
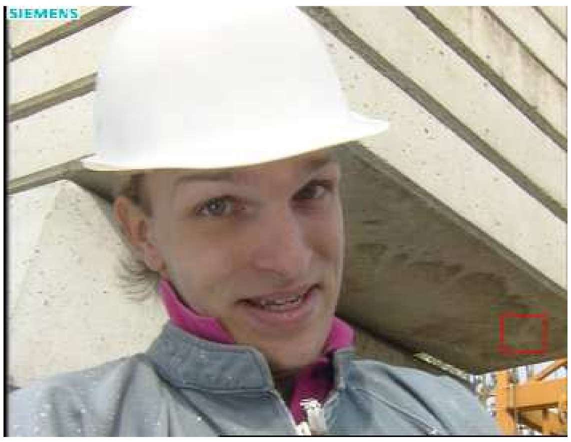

(d) “Flower Garden” image 124.

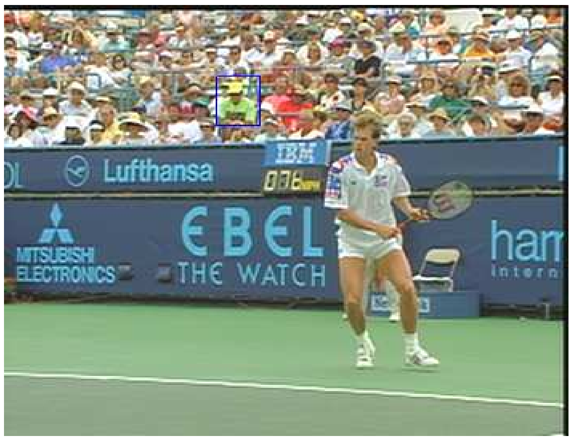
Figure 5.7: Pseudo-feature tracking capability of the new algorithm.



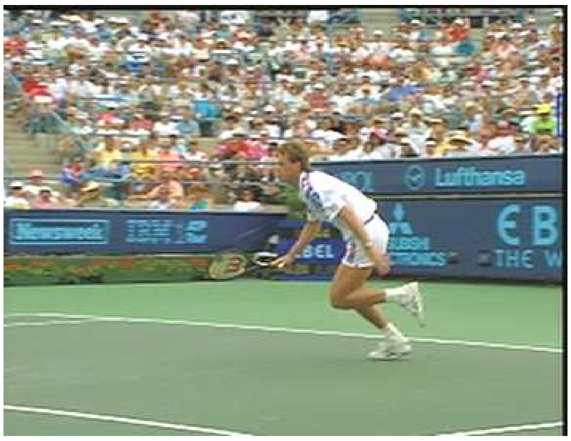
(a) "Foreman" image 0.



(b) "Foreman" image 150.



(c) "Stefan" image 0.



(d) "Stefan" image 243.

Figure 5.8: Pseudo-feature tracking capability of the new algorithm.

Computational requirements

The total running time for the estimation of camera movement for every image, except the first, of the "Stefan" sequence, which has 300 images, was of about 9500 seconds for both algorithms.[6] However, full search block matching accounts for about 98.5% of this time. Excluding full search, for which hopefully there are fast hardware implementations available, the old algorithm spent 133.07 seconds and the new algorithm 129.02 seconds estimating 299 camera movement factors. I.e., both algorithms take about half a second to run for each image. The algorithms are thus essentially equivalent in computation time, though the memory requirements are larger for the new algorithm, since it requires use of the matrix of Hough accumulator cells H for outlier detection.

5.5 Image stabilization

Image stabilization, be it mechanical or electronic, is an important feature of today's video cameras [102], whether they are hand-held or part of a videotelephone. It is important because it improves the quality of moving images by reducing disturbing image vibrations caused by an unsteady hand. Since image stabilization is performed before storage or transmission, the compression ratio is also improved, since images are more easily predicted from the previous ones.

5.5.1 Viewing window

Let the digital image available from a camera after sampling have $L \times C$ pixels. Let also the needed output image have $L^w \times C^w$, with $L^w \leq L$ and $C^w \leq C$, and the same pixel size. Let $\Delta_L = \frac{L-L^w}{2}$ and $\Delta_C = \frac{C-C^w}{2}$. The $L^w \times C^w$ image can be extracted from the camera image by placing its center, with site coordinates s^w, within a rectangle $[-\Delta_C, \Delta_C] \times [-\frac{1}{\alpha}\Delta_L, \frac{1}{\alpha}\Delta_L]$. The rectangle of width C^w and height $\frac{1}{\alpha}L^w$ centered at s^w will be called the viewing window. The output image can be obtained from the camera image by selecting the corresponding camera image pixels, if v^w (s^w in pixel coordinates) has integer coordinates, or by using some type of interpolation if the coordinates of v^w can take non-integer values. In this section bilinear interpolation will be used for simplicity. Bilinear interpolation introduces a linear filtering to the camera image whose frequency response depends on the fractional parts of the coordinates of v^w. For instance, for null fractional parts of the coordinates of v^w, no filtering is performed (flat frequency response), while for half-pixel coordinates (fractional parts of 0.5), the pixels of the output image are the average of four pixels in the camera image, corresponding to a low-pass filtering of the camera image. This is clearly not a desirable effect, but the use of more appropriate interpolation methods remained as an issue for future work.

[6]On a Pentium 200 MHz, with 64 Mbyte RAM, running RedHat Linux 5.0 (kernel 2.0.32), programs compiled with gcc 2.8.1 and full compiler optimization (-O3), times extracted with gprof 2.8.1.

5.5.2 Viewing window displacement

Small camera pan movements can be eliminated from the output image by displacing the viewing window, relative to its previous position, in the direction of the detected camera movement, thereby stabilizing the output image [192, 122]. Since the viewing window must always fit inside the camera image, the amount of movement that can be eliminated is limited, of course.

The objective of the displacements of the viewing window is to maintain the point which was at its center in one image also centered in the in the next image, even in the presence of camera movement. Let $s^w[n]$ be the position of the center of the viewing window at image n, and $Z[n]$ and $d^s[n]$ the camera movement factors (in site coordinates) from image $n-1$ to image n. Then a point at the center of the viewing window in image $n-1$ will be moved to $Z[n](s^w[n-1]-d^s[n])$ in image n, according to equation (5.3). Hence, if this movement is to be eliminated from the output image, the center of the viewing window has to be moved accordingly to

$$s^w[n] = Z[n](s^w[n-1] - d^s[n]). \tag{5.13}$$

If the coordinates of s^w ever fall outside the rectangle $[-\Delta_C, \Delta_C] \times [-\frac{1}{\alpha}\Delta_L, \frac{1}{\alpha}\Delta_L]$, then the camera movement cannot be fully canceled out, i.e., the image cannot be stabilized as required. The evolution of the center of the viewing window will thus be described by

$$\begin{bmatrix} s^w_x[n] \\ s^w_y[n] \end{bmatrix} = \begin{bmatrix} \min\Big(\max\big(Z[n](s^w_x[n-1] - d^s_x[n]), -\Delta_C\big), \Delta_C\Big) \\ \min\Big(\max\big(Z[n](s^w_y[n-1] - d^s_y[n]), -\frac{1}{\alpha}\Delta_L\big), \frac{1}{\alpha}\Delta_L\Big) \end{bmatrix}.$$

With this equation, after a large panning, say, to the right, the viewing window will be saturated at the left of the camera image. Even after the panning stops, it will remain there, effectively reducing the capabilities of image stabilization in that direction. Hence, some mechanism for returning the viewing window to its rest position, viz. the center of the camera image, must be devised. This is accomplished through the introduction of a loss factor $\beta(Z[n], d^s[\cdot])$ in the evolution of the center of the viewing window

$$\begin{bmatrix} s^w_x[n] \\ s^w_y[n] \end{bmatrix} = \begin{bmatrix} \min\Big(\max\big(Z[n]((1-\beta(Z[n], d^s[\cdot]))s^w_x[n-1] - d^s_x[n]), -\Delta_C\big), \Delta_C\Big) \\ \min\Big(\max\big(Z[n]((1-\beta(Z[n], d^s[\cdot]))s^w_y[n-1] - d^s_y[n]), -\frac{1}{\alpha}\Delta_L\big), \frac{1}{\alpha}\Delta_L\Big) \end{bmatrix}. \tag{5.14}$$

The loss factor is usually zero, but when the pan factor d^s is small enough for a given number of images, the loss will be set to a non-zero, small value, which will slowly revert s^w to the center of the camera image. When the zooming is very strong, the same thing happens, though with a larger loss, so as to reset the viewing window position faster. When camera movement estimation fails, which is likely to occur at scene cuts, the center of the viewing window is reset immediately. Hence, the evolution of s^w is governed by

$$\begin{bmatrix} s^w_x[n] \\ s^w_y[n] \end{bmatrix} = \begin{cases} \text{equation (5.14)} & \text{if camera movement was detected,} \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \text{if camera movement estimation failed;} \end{cases} \tag{5.15}$$

where

$$\beta(Z[n], d^s[\cdot]) = \begin{cases} 0.1 & \text{if } |z[n]| > 0.02 \text{ (where } z[n] = \frac{1}{Z[n]} - 1\text{)}, \\ 0.01 & \text{if } |z[n]| \leq 0.02 \text{ and } \|d^s[n-i]\| \leq 0.1 \text{ for } i = 0, \ldots, 4, \text{ and} \\ 0 & \text{otherwise.} \end{cases} \tag{5.16}$$

The values used in (5.16) were derived empirically. The loss factors of 0.1 and 0.01 were chosen because they reset the viewing window position in respectively 20 to 30 images (about 1 second for 25 Hz sequences) and in 220 to 230 images (about 9 seconds for 25 Hz sequences). Thus, if the camera stops, it takes 9 seconds for the viewing image to center, while if a large zoom in or out occurs, the viewing window is reset in 1 second. The use of a loss only after five small pan movements precludes it from affecting effective image stabilization in the presence of vibration.

5.5.3 Results

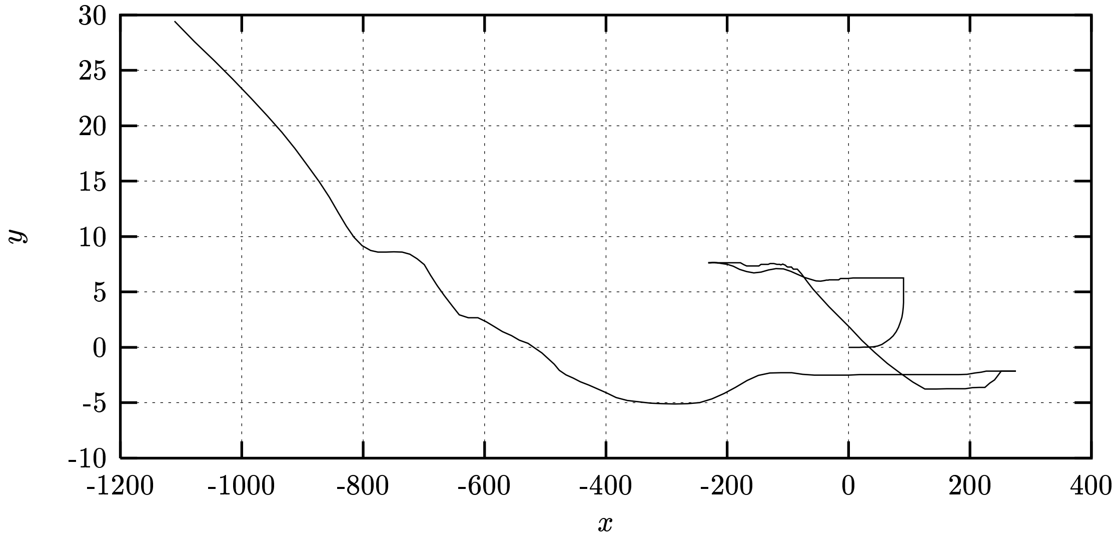
Figures 5.9, 5.10, 5.11 show the evolution of the center of the image as given by (5.13), for the CIF sequences "Stefan", "Foreman", and "Carphone". Actually, the coordinates have both been inverted so that the figures show the movement of the camera. The figures also show the residual camera movement, i.e., the camera movement that remains after adjustment of the viewing window position according to (5.15) and (5.16). The output image size used is 268×328, that is $L^w = 268$ (or $\Delta_L = 10$) and $C^w = 328$ (or $\Delta_C = 12$).

These results show that the method effectively stabilizes the output image, as can be seen by looking at the residual camera movement, provided the camera movement factors have been well estimated.

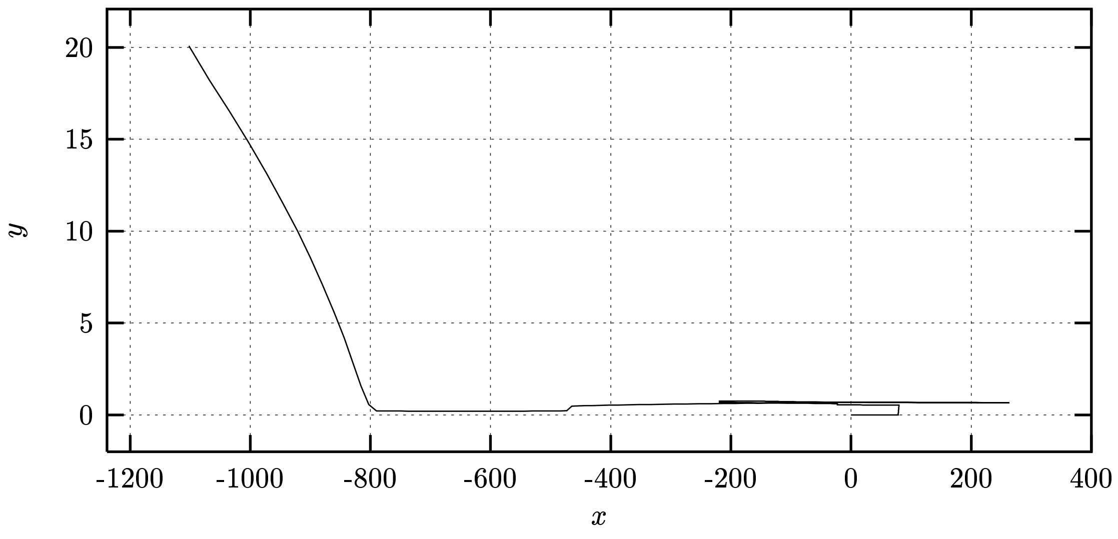
In order to ascertain the effectiveness of the stabilization, which depends on the quality of the camera movement estimation, the output image sequences have to be viewed in real time. The visualization of the results shows that the stabilization of the "Stefan" and "Foreman" sequences was very effective. In the case of the "Carphone" sequence, the results are not as good. This results essentially from the following:

1. the camera vibration is rather small, often smaller than one pixel;
2. the static background in rather uniform, which makes the block matching motion vectors less reliable; and
3. the speaker and the landscape seen through the window, both with motion independent of the camera, occupy a significant part of the images.

The net result is that sometimes the estimated factors reflect the motion of the speaker, not the one of the background. Figures 5.12 and 5.13 show the differences between images 5 and 6, and between images 26 and 27, with and without cancellation. The first case shows that the estimated camera movement has been "polluted" by the speaker, so that the differences in the window frame, to the right of the speaker, which are due to horizontal pan movements,

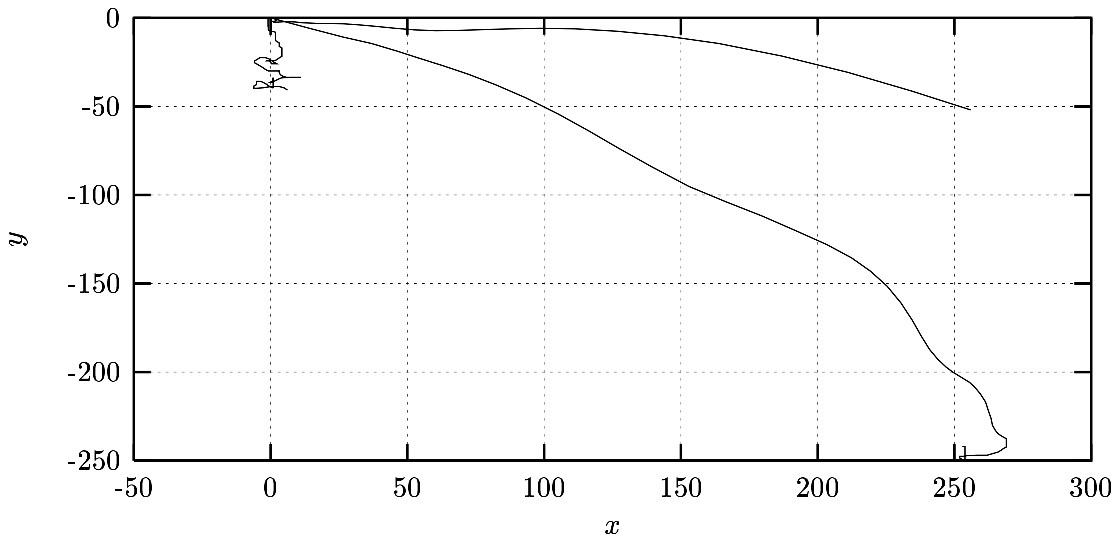


(a) Without stabilization.

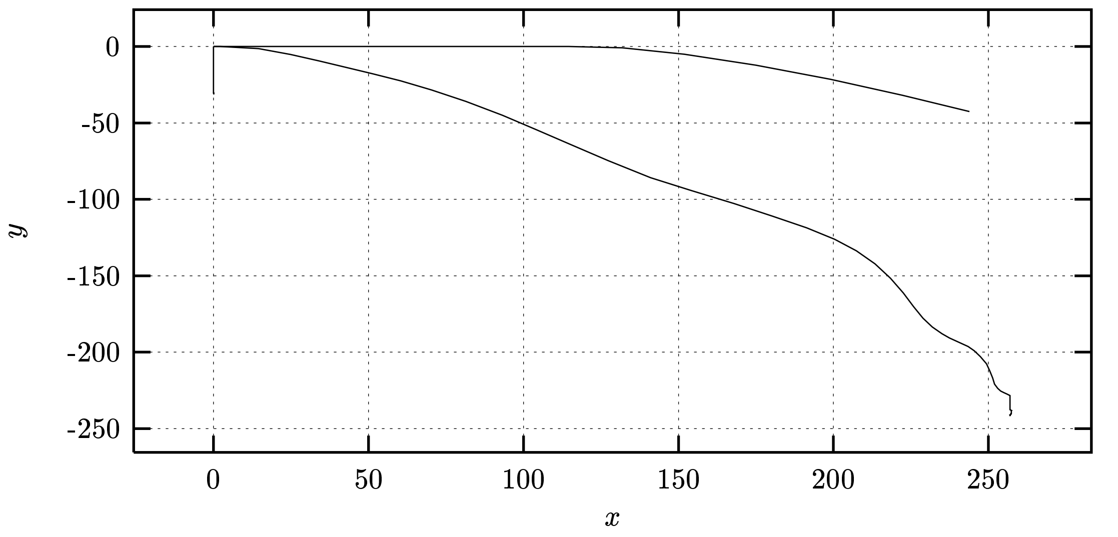


(b) With stabilization.

Figure 5.9: “Stefan”: camera movement.

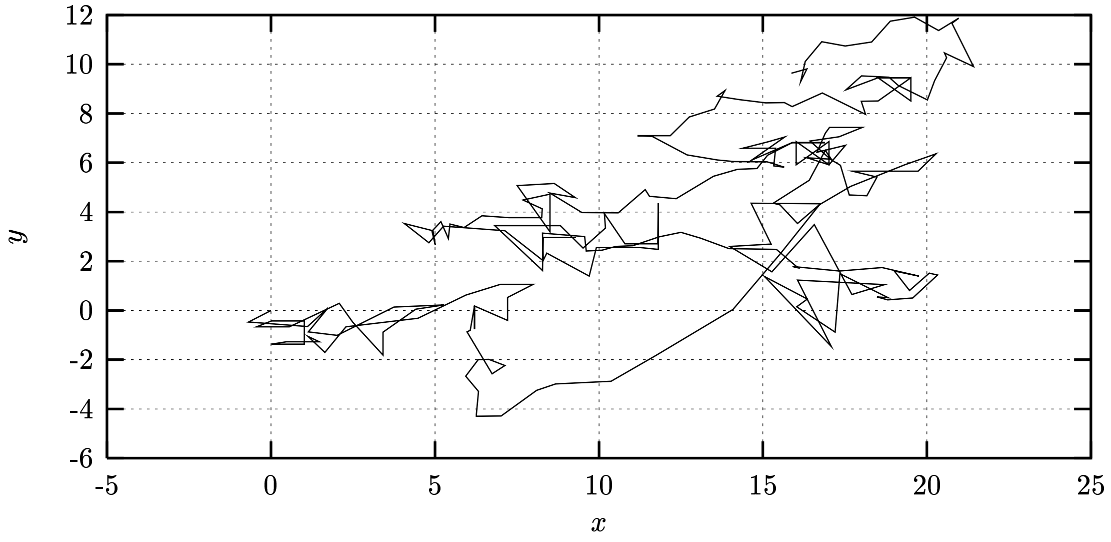


(a) Without stabilization.

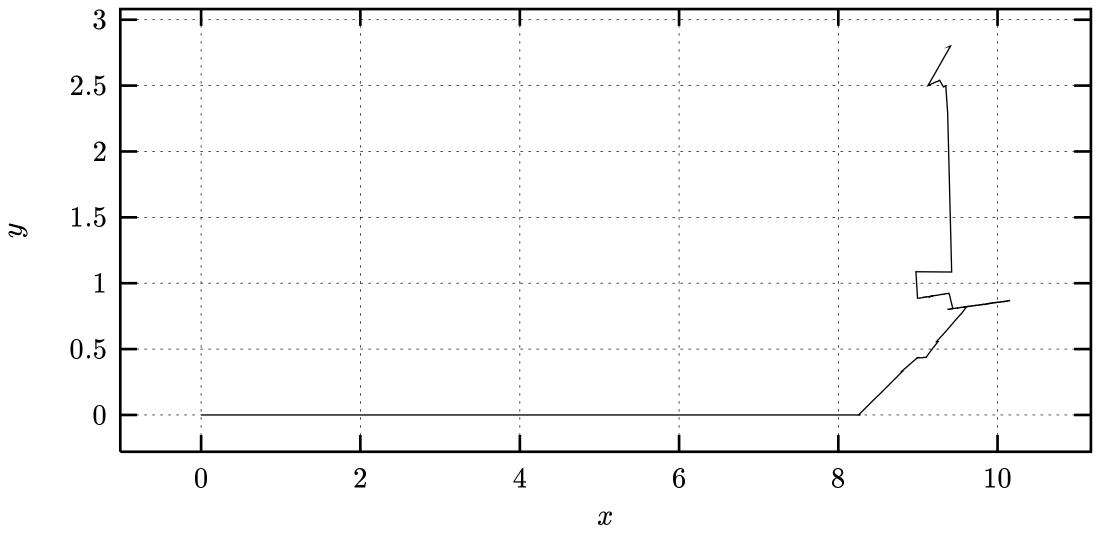


(b) With stabilization.

Figure 5.10: "Foreman": camera movement. Note that the position of the camera is reset whenever camera movement is not detected (at images 156, 191, and 192).



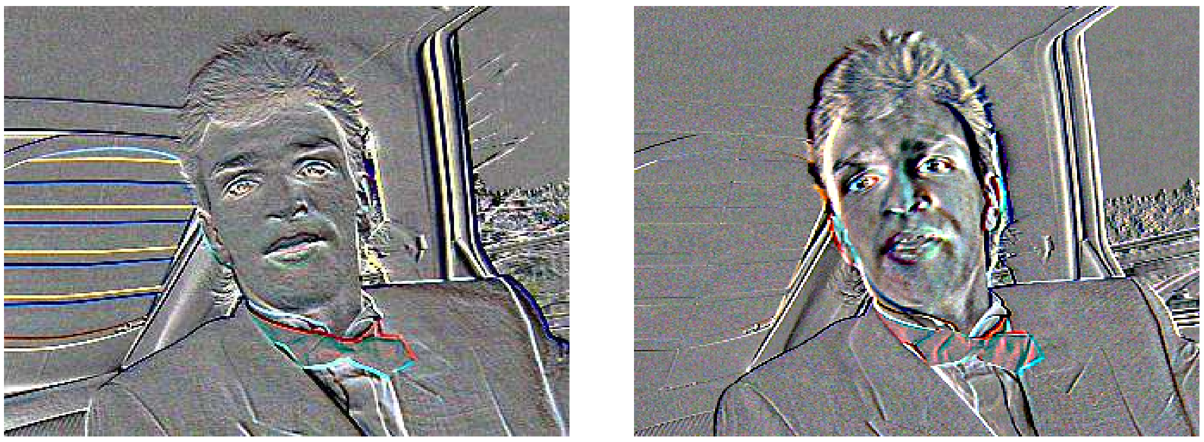
(a) Without stabilization.



(b) With stabilization.

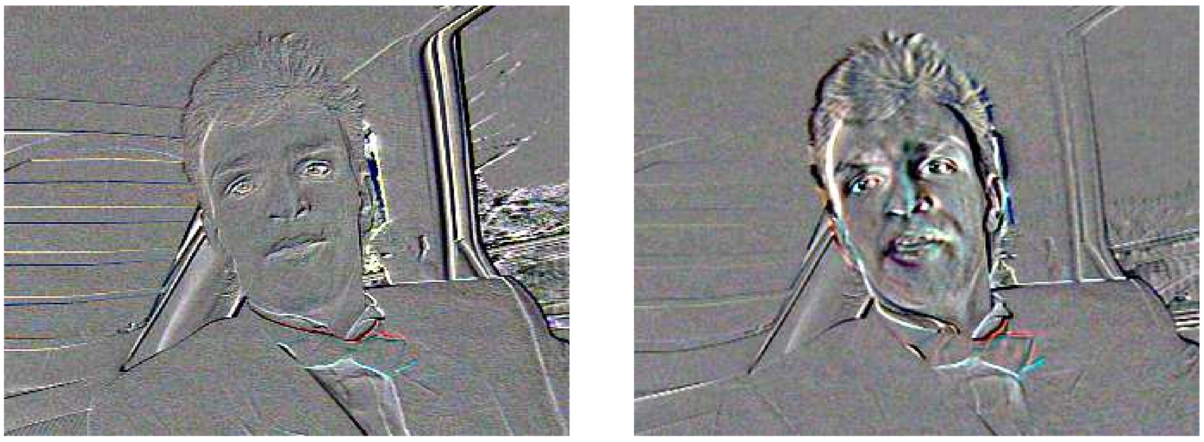
Figure 5.11: "Carphone": camera movement.

have not been canceled. The vertical estimate, however, is quite precise, as can be seen in the reduced differences obtained in to the left of the speaker. In the second case the estimated camera movement is correct, since the static background has been mostly canceled out. The overall results are quite acceptable.



(a) Images 5 and 6. (b) Images 26 and 27.

Figure 5.12: "Carphone": difference between successive images without stabilization. The luma differences are scaled by 5 and offset by 125.5 (half way the luma excursion in $Y'C_BC_R$), and the chroma differences are scaled by 5 and offset by 128 (zero chroma in $Y'C_BC_R$).



(a) Images 5 and 6. (b) Images 26 and 27.

Figure 5.13: "Carphone": difference between successive images with stabilization. See note in Figure 5.12.

Figures 5.14, 5.15 and 5.16 show the comparison between the PSNR of each image relative to the previous in the original sequences (but restricted to a centered window of 268×328) and in the stabilized sequences. The improvements in PSNR are quite good, even when the viewing window is saturated in one direction. When it is saturated in both directions, as around image 200 of the "Foreman" sequence, the results are quite similar, as expected.

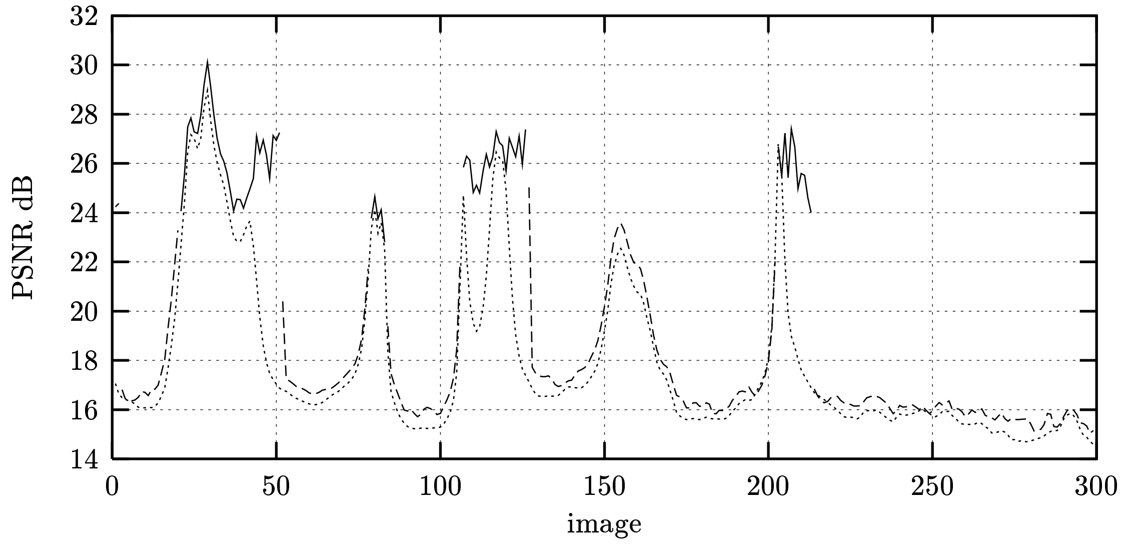


Figure 5.14: "Stefan": PSNR between each pair of successive images. PSNR without stabilization, dotted line, with stabilization saturated viewing window position, broken line, with stabilization and no saturation of window position, solid line.

Finally, it must be stressed here that the results of image stabilization on the "Stefan" sequence are valid for checking the validity of the proposed method, even though (electronic) image stabilization is of little or no use in professionally shot sequences, which often make use of specially constructed devices to guarantee stability of the image, and where the camera operator wants complete control of the sequence being shot.

5.6 Conclusions

Two camera movement estimation algorithms have been proposed, the second of which can be seen as a simplification of the method in [4]. However, both methods integrate a motion vector smoothing intermediate step which tends to reduce the number of outliers and thus to improve the camera movement estimate. A method for image stabilization has also been proposed, which is similar to the one in [122], but which includes the zoom factor into the equations for the viewing window position, thereby improving its performance in case of zoom movements. The camera movement estimation algorithm based on the Hough transform and the image stabilization algorithm have been shown to perform quite well by using several sequences with and without camera movement. This last Hough transform-based camera movement estimation algorithm was also shown to perform quite well as a scene cut detector.

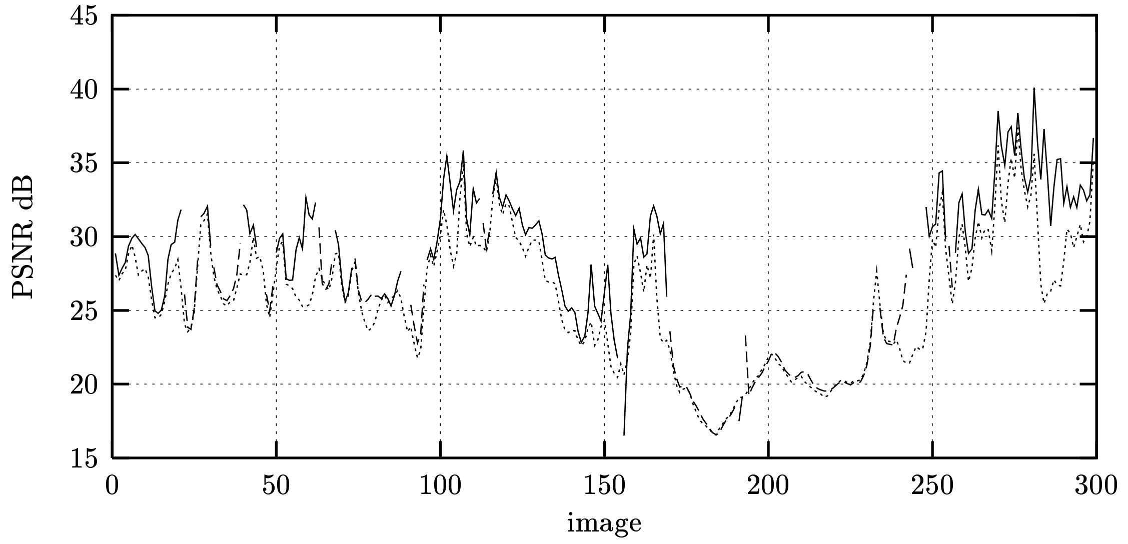


Figure 5.15: "Foreman": PSNR between each pair of successive images. The peaks at images 156, 191, and 192 are due to the non-detection of camera movement at those images, which leads to a reset of the viewing window position. PSNR without stabilization, dotted line, with stabilization saturated viewing window position, broken line, with stabilization and no saturation of window position, solid line.

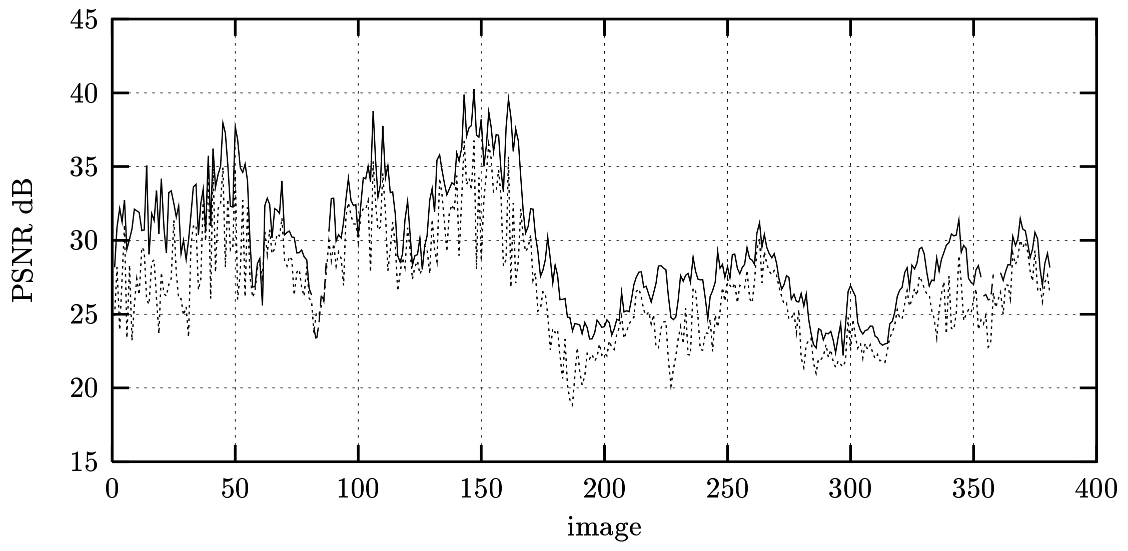


Figure 5.16: "Carphone": PSNR between each pair of successive images. PSNR without stabilization, dotted line, with stabilization saturated viewing window position, broken line, with stabilization and no saturation of window position, solid line.

Chapter 6

Coding

"Nicholas, I saw you wink at Elaine. What did you tell her?"

Nicholas Negroponte

Visual information is usually obtained in an unstructured way, by a sampling and quantization process. Several problems must be solved for effective transmission and/or storage of this unstructured information. Firstly, it must be fit into a more structured model, whose parameters represent the scene implicit in the original unstructured data as faithfully as necessary. The choice of an appropriate model or models is called modeling. Secondly, the parameters of the given model must be estimated. This estimation is called analysis. Finally, the model parameters must be encoded, so as to achieve the typical goals of coding: high compression ratio, high quality, low cost, and easy access to the structured content.

A codec contains analysis blocks and encoding blocks, as in Figure 2.1. The design of a complete codec encompasses choosing appropriate models, building the analysis algorithms, and constructing appropriate encoding methods. Analysis has been dealt with in the previous chapters. This chapter proposes encoding methods. Section 6.1 presents a camera movement encoding method for classical codecs (a transition to second-generation tool), and Section 6.2 discusses a taxonomy of partition types and representations which will be used in Section 6.3 to overview possible partition coding techniques. Finally, Section 6.4 develops a fast cubic spline implementation method with applications on parametric curve partition coding.

6.1 Camera movement compensation

The development of camera movement detection and estimation algorithms in [129, 127, 130, 128, 113, 122] led to a proposal for camera movement compensation in classical, first-generation codecs. In particular, a proposal was made for the extension of the H.261 standard which would take into account camera movements with minimum syntax changes. This proposal was published in [122], and is described briefly in this section.

It should be noticed that the ideas herewith presented can be applied without change to other video coding standards, such as H.263, MPEG-1, MPEG-2, and MPEG-4.

This section is divided in four parts. The first studies natural ways to quantize the zoom factor, which should be of generic use. The second presents the proposed H.261 extensions. The third part describes how an encoder may make use of the proposed extensions. The fourth and last part presents some results and discusses them.

6.1.1 Quantizing camera motion factors

Quantization is the process by which a continuous quantity is rendered discrete for storing or transmission. The zoom factor estimated by the algorithms in Chapter 5, equation (5.9), is a rational quantity. In all practical cases, it will already be "quantized" to fit into some fixed length floating point register. It is necessary to further quantize it so that it can be efficiently encoded.

Since most sequences do not have zoom and H.261 uses motion vectors detected at pixel level, it seems reasonable to round the pan factor components to integers. Besides, no movements larger than 15 pixels per MB are possible, as specified in the H.261 recommendation. So the quantization of the pan factor components, which are given by (5.10) and (5.11), is based simply on rounding to the nearest integer and limiting the result to the interval $[-15,\ 15]$. This results in five bits encoding with a FLC (Fixed Length Code).

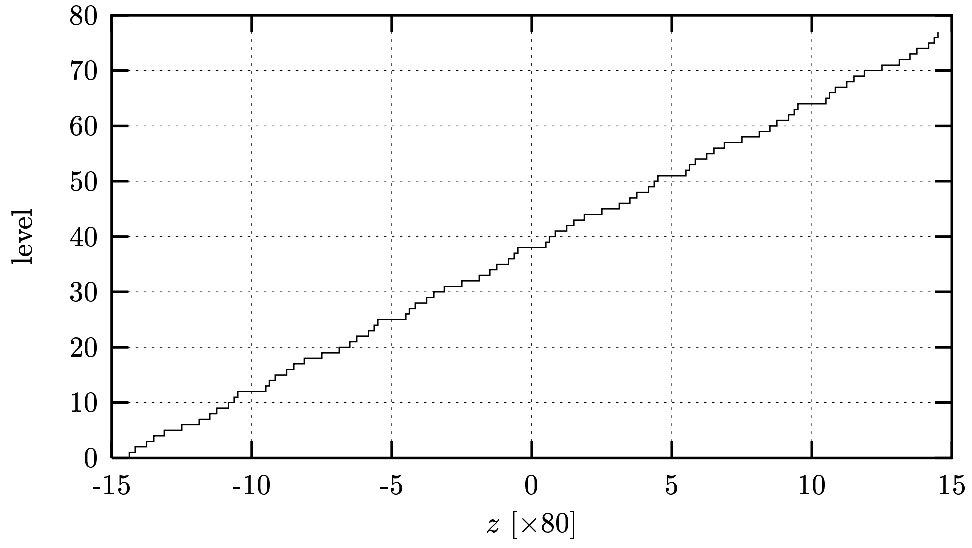
As to the zoom factor, it is assumed that the largest and smallest zooms allowed are the ones leading to -15 or 15 of either motion vector coordinates, calculated according to (5.4), in one of the MBs at the border of the image. This leads, after some simple calculations, to the following limits:

$$\begin{cases} -15/168 \leq z \leq 15/168 & \text{for CIF, and} \\ -15/80 \leq z \leq 15/80 & \text{for QCIF.} \end{cases}$$

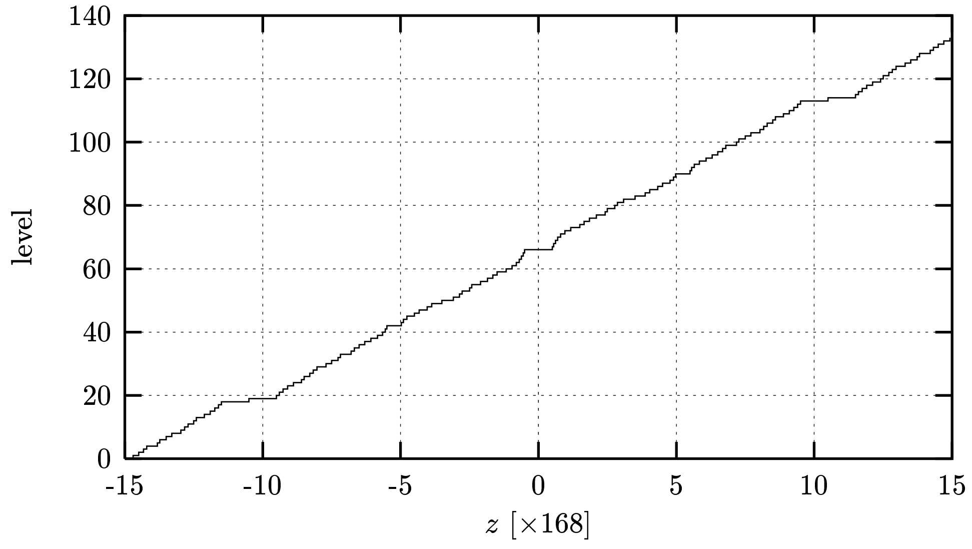
Notice that the maximum allowable zoom is larger in QCIF than in CIF, because the QCIF pixels are also larger than the corresponding ones in CIF.

Within the limits presented above for z, there is a finite number of distinct motion vector patterns in the rounded motion vector field $\bar{\mathcal{M}}_b[m,n](z,d)$ for any given d. Each of these patterns may be classified by an integer number, thus resulting in a natural quantization characteristic for z. The boundary values of z where the motion vector pattern changes have been calculated numerically. The results are presented in Figure 6.1 in the form of a quantization characteristic for z with 135 quantization levels for CIF resolution—8 bit encoding with a FLC—and 79 levels for QCIF resolution—7 bits with FLC.

Notice that several of the values for z around zero correspond to almost all motion vectors of zero length, except for a few on the boundaries of the image having small amplitudes. As such,



(a) QCIF.



(b) CIF.

Figure 6.1: The quantization characteristics of the zoom factor z.

it makes sense to include a dead zone in the z quantization characteristic. This may reduce the number of quantization levels below 129, thus allowing for a seven bit FLC for the quantized zoom factor for CIF images.

It should be noticed that similar quantization characteristics may be obtained for different image sizes, different maximum values of the corresponding motion vector field, and even if half- or quarter-pixel (or any fraction of the pixel, for that matter) motion vectors are admissible for motion compensation. The presented characteristic was developed with the H.261 extension in mind.

6.1.2 Extensions to the H.261 recommendation

The extensions of the H.261 recommendation so as to provide means for camera motion compensation are of two types: syntactical and semantical. The former changes are small, but the latter ones are more substantial. Both will be addressed below.

Syntactical extensions

The necessary syntactical extensions to the H.261 recommendation [62] are few:

1. Bit 5 of the PTYPE (Picture Type) means: 0 no camera movement is used, 1 camera movement is used.
2. When there is camera movement, the first three PEI (Picture Extra Insertion Information) bits will be set to 1 and PSPARE (Picture Spare Information) will contain the pan and zoom factors:

   First byte
   Horizontal component of pan factor (five bits).

   Second byte
   Vertical component of pan factor (five bits).

   Third byte
   Quantized zoom factor (eight bits).

The previous changes imply that camera movement compensated image will have an overhead of 27 bits. Appropriate VLCs (Variable Length Codes) for the pan and zoom factors may be developed in order to reduce this small overhead.

These extensions make use of reserved bits in the H.261 recommendation, and thus are totally compatible with existing H.261 compliant encoders and decoders.

Semantical extensions

The semantical extensions to the H.261 recommendation are the following:

1. Let there be a rounded camera movement motion vector field generated from the transmitted quantized pan and zoom factors, according to (5.5).

2. When there is camera movement present (fifth bit of PTYPE), the following interpretations will apply to inter MBs:

   - A MB is MC (Motion Compensated): this means that it is *local motion* compensated. Its MVD (Motion Vector Data) is obtained from the MB vector by subtracting the vector of the preceding MB or by subtracting the corresponding vector of the rounded camera movement vector field if:
     (a) evaluating MVD for MBs 1, 12 and 23 (left edge of GOB (Group of Blocks));
     (b) evaluating MVD for MBs in which MBA (MacroBlock Address) does not represent a difference of 1; or
     (c) the MTYPE (Macroblock Type) of the previous MB was not MC.
   - A MB is *not* MC: this means that it is camera movement compensated. It will be compensated using the corresponding motion vector from the camera movement motion vector field.

Thus, all MBs are predicted using the transmitted camera movement, except those with local motion. But even these will make use of the camera movement motion vector field by improved prediction of their motion vectors.

Similar schemes may be used in other standards, such as H.263. In H.263, where the prediction of motion vectors is more involved and efficient than in H.261, the motion vector of the camera movement field may be used to obtain improved prediction.

6.1.3 Encoding control

Given the extensions to the H.261 recommendation, there is still some information lacking about how to put them all to work in a fully functional extended H.261 codec.

A possible encoding procedure extends the one proposed in RM8 (Reference Model 8) [21]:

1. Take the current original image and the previous decoded image and perform full-search block matching motion estimation.

2. From the results of the previous item detect whether or not there is camera movement, and estimate it if there is. If no camera movement was estimated, proceed as in RM8. If camera movement was estimated, build a rounded camera movement motion vector field from the detected pan and zoom factors, according to (5.5).

3. Classify each MB as in RM8, except that non-motion compensated MBs should be predicted using the corresponding camera movement motion vector. Smoothing, as described in the previous chapter, may be used in motion compensated MBs to approximate the corresponding motion vector to the camera movement motion vector or to the preceding MB motion vector, whichever is more appropriate. This reduces the number of bits spent

transmitting motion vectors, though the prediction error may increase in a controlled fashion.

4. Proceed as in RM8, but encode according to the semantical extensions proposed.

6.1.4 Results and conclusions

Camera movement compensation, as presented, aims essentially at the reduction of redundancy in the field of motion vectors. This is somewhat misleading, firstly because H.261 already codes the motion vectors in a DPCM (Differential Pulse Code Modulation) way which removes part of the field redundancy, and secondly because, for very low bitrates, since some quality degradation must be accepted, the motion vector field shows little uniformity, particularly for mobile sequences. Figure 6.2 shows a typical vector field out of the sequence "Foreman" (encoded by RM8 at 24 kbit/s and 5 Hz image rate and using QCIF resolution) where this non-uniformity can be clearly seen.

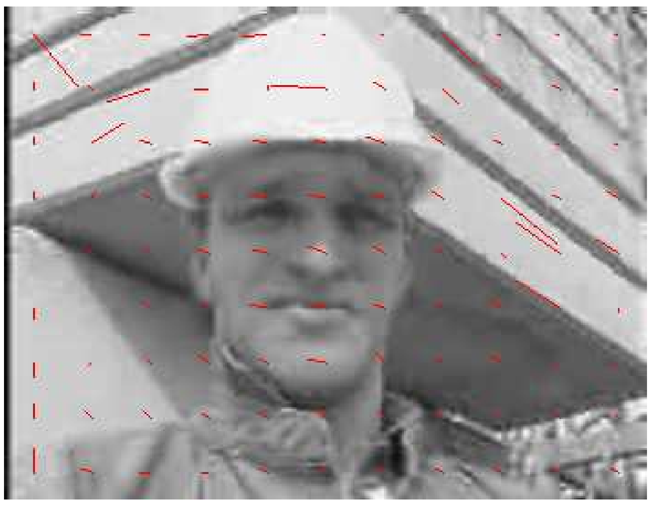


Figure 6.2: Typical motion vector pattern for "Foreman" at 24 kbit/s, 5 Hz image rate and QCIF resolution.

The overall weight of the motion vectors in terms of spent bits per image, though larger for smaller bitrates, is still small. For instance, using RM8 to encode the sequence "Foreman" at 24 kbit/s, with an image rate of 5 Hz and using QCIF resolution, the average number of bits per image spent encoding motion vectors and DCT coefficients is, respectively, 690 and 2866. The motion vectors account only for about 20% of the total. This means that, even if substantial reduction in the former were obtained, the gains in terms of quality, for the same target bitrate, would be somewhat small.

Two experiments will help to clarify matters. In the first, an ordinary RM8 coder was used. In the second, a modified coder which, when performing bitrate control, assumes that motion vectors are transmitted "magically", i.e., without spending any bits. Both experiments were performed though the encoding of "Foreman" with a target of 24 kbit/s, using 5 Hz image rate and at QCIF resolution. The resulting average luma PSNR obtained was respectively 23.94 dB and 24.38 dB. The gains in PSNR can be seen to be small. Using a realistic camera movement compensation approach, one which results in actual bits being spent, the results will forcibly be worst: an average PSNR of 24.01 dB was obtained using the camera movement compensation method proposed before. For this experiment, the average number of bits per image spent encoding the motion vectors and the DCT coefficients was, respectively, 649 and 2916. The transfer of bits from motion vector to DCT coefficients is small due to the non-uniformities of the motion vector field, which reduces the gain of camera movement compensation.

One way to attempt to solve the non-uniformity problem is through the use of smoothing. However, since smoothing introduces larger prediction errors, some, if not more, of the bits spared transmitting the motion vectors will be wasted compensating this worst prediction. After experimenting camera movement compensation with smoothing under the same conditions as before, an average luma PSNR of 23.99 dB was obtained, showing a little loss in terms of objective quality. In this case the bit distribution obtained was 607 for motion vectors and 2966 for DCT coefficients.

The results, though obtained for the H.261 standard, are expected to be valid also in more modern standards as H.263 and MPEG-4. This does not mean that the camera movement is useless in video coding. Its importance, as well as the importance of the scene cut detection, are considerable, for instance, if metadata is to be extracted from the sequences, i.e., if "data about the data" is necessary. This seems to be the case in video database indexing applications.

6.2 Taxonomy of partition types and representations

Image analysis algorithms usually produce partitions of the scenes into 2D (or 3D) regions. These partitions usually have to be coded during the image and video representation process. It has been recognized that partition information will account for a significant percentage of the bit stream (e.g., [61]). It is thus very important to develop efficient partition coding techniques.

The comparison of techniques proposed in the literature has often been haunted by the lack of systematization of the subject. This section attempts to fill this gap by proposing a taxonomy of partition types and representations. The proposals made in this section and the overview contained in Section 6.3 have already been published in [120, 121], and stem from preliminary work published in [31].

The two main levels of the taxonomy, partition type and partition representation, can be seen to correspond to the first steps taken when developing a partition coding technique: the identification of the problem to be solved corresponds to the identification of the partition type addressed by the coding technique, and the selection of the partition representation corresponds to selecting the kind of data the coding technique will manipulate. Thus, different representations, usually leading to different techniques, can be used for the same type of partitions.

During the description of the taxonomy tree levels, square brackets will be used to specify the codes representing the possible branches at each tree node.

6.2.1 Partition type

The partition types can be organized in a tree with the following levels:

1. **Space**
   Are partitions 2D [2D] or 3D [3D]?
2. **Lattice**
   What sort of sampling lattice was used for digitizing the images from which the partitions were obtained (e.g., rectangular [R] or hexagonal [H])?
3. **Graph**
   What kind of graph is super-imposed on the partition (usually a neighborhood system is specified [N_n])?
4. **Classes**
   Are partitions binary [B] or mosaic [M]?
5. **Connectivity**
   Are the classes connected [C] or can they be disconnected [D] (on the chosen image graph)?

Figure 6.3 shows the partition type levels of the taxonomy tree. The leaves of the taxonomy tree correspond to different types of partition. Each type of partition can be specified by answering the five questions listed above. For instance, the following answers: 1. 2D [2D], 2. hexagonal [H], 3. 6-neighborhood [N_6], 4. mosaic [M], and 5. connected [C], (or, with codes, 2DHN$_6$MC); define a type of partitions that lie in a 2D space, that correspond to digital images sampled according to an hexagonal lattice, that are structured according to the hexagonal graph, that can have more than two classes, and where all classes are connected (the concepts of class and region are equivalent in this case).

Notice that the branches under “3D” in the figure are not drawn, since 2D partitions are the focus of this section. At the partition representation level, however, 3D partitions will be considered in more detail (see the next section).

6.2.2 Partition representation

This section introduces more levels of detail into the taxonomy tree, related with the representation chosen for the partitions. 2D and 3D partitions will be dealt with separately.

2D partitions

The first important decision to be made regards mosaic partitions:

1. **Handling**
   Should the mosaic partitions be handled as such (a single mosaic partition) [M] or should

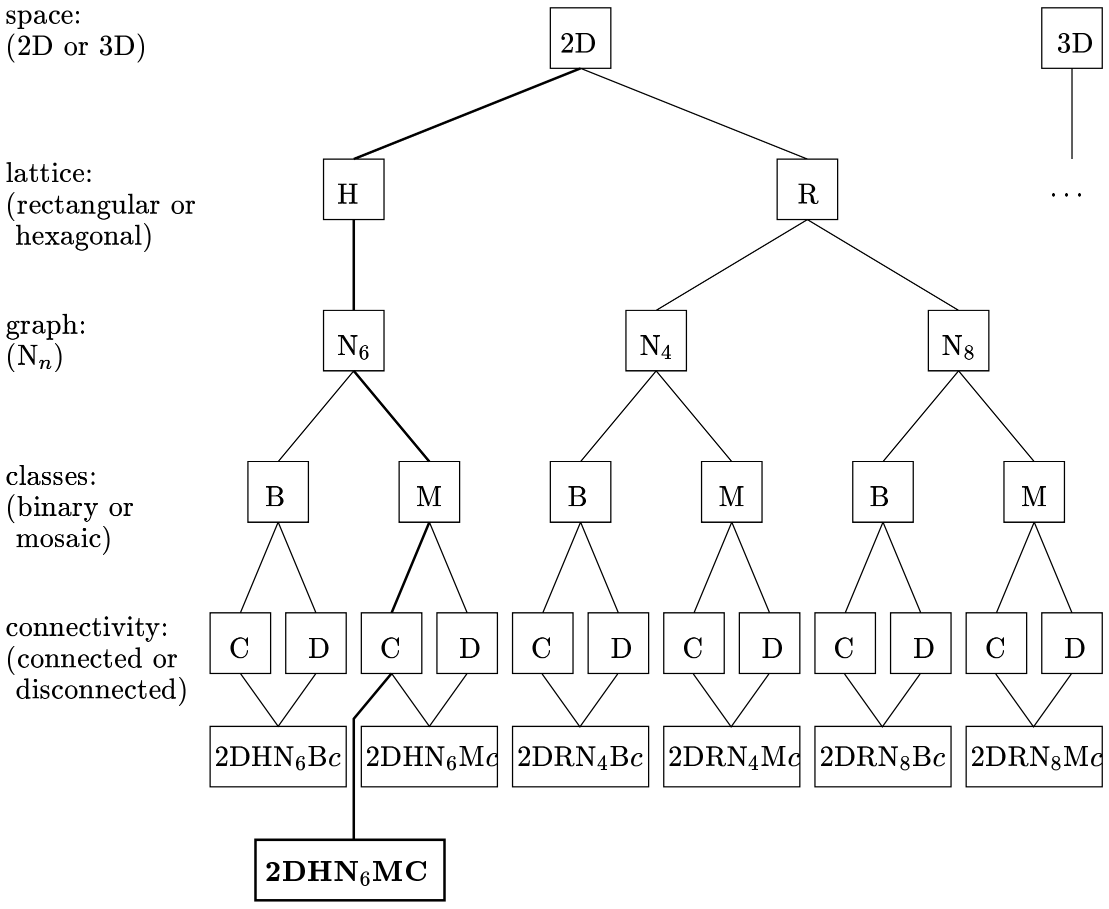


Figure 6.3: The partition type taxonomy tree (in bold, the example given in the text). *c* stands for either C (connected classes) or D (disconnected classes).

they be separated into a collection of binary partitions (each one corresponding to a different class in the original mosaic partition) [B]?

As will be discussed later, the handling of mosaic partitions as collections of binary partitions is often of paramount importance. For instance, when classes should be readily accessible from the coded bit stream, a collection of binary partitions may allow an easier access to the various objects in a scene than the original mosaic partition.

It has been seen that a partition can be represented in two different ways: either by the labels of each pixel, or by contour information plus region-class information.[1] When class equivalence is the aim, the latter representation provides information about the clustering of regions into a certain number of classes.

Hence, the next level in the taxonomy will be:

2. **How**
   How should the partition be represented? With pixel labels [L] or with contours [C]?

For the case of partitions represented with contours, other choices have to be made: How to represent the contours? What sort of neighborhood system has the contour graph? These questions lead to two other levels of partition representation in the taxonomy tree:

3. **Where**
   Where should contours be defined? On the image graph or on the line graph? That is, should the contour be defined on pixels [P] or on edges [E]?

4. **Graph**
   What is the kind of neighborhood system of the graph from which the contour is a subgraph [N_n]?

Figure 6.4 shows the partition representation levels of the taxonomy tree for the 2D case.

The $2DHN_6Mc$ partition type with a representation separated into binary class partitions, using contours defined on edges, which have a N_3 neighborhood system, is coded as $2DHN_6Mc$-$BCEN_3$ or:

Partition type
2D, hexagonal lattice, N_6 graph, mosaic, classes connected or disconnected according to whether c is C or D.

Partition representation
Mosaic treated as independent binary partitions, contours, edges, N_3 graph.

[1]Similarly, [38] divides shape representation methods into boundary-based, and area-based.

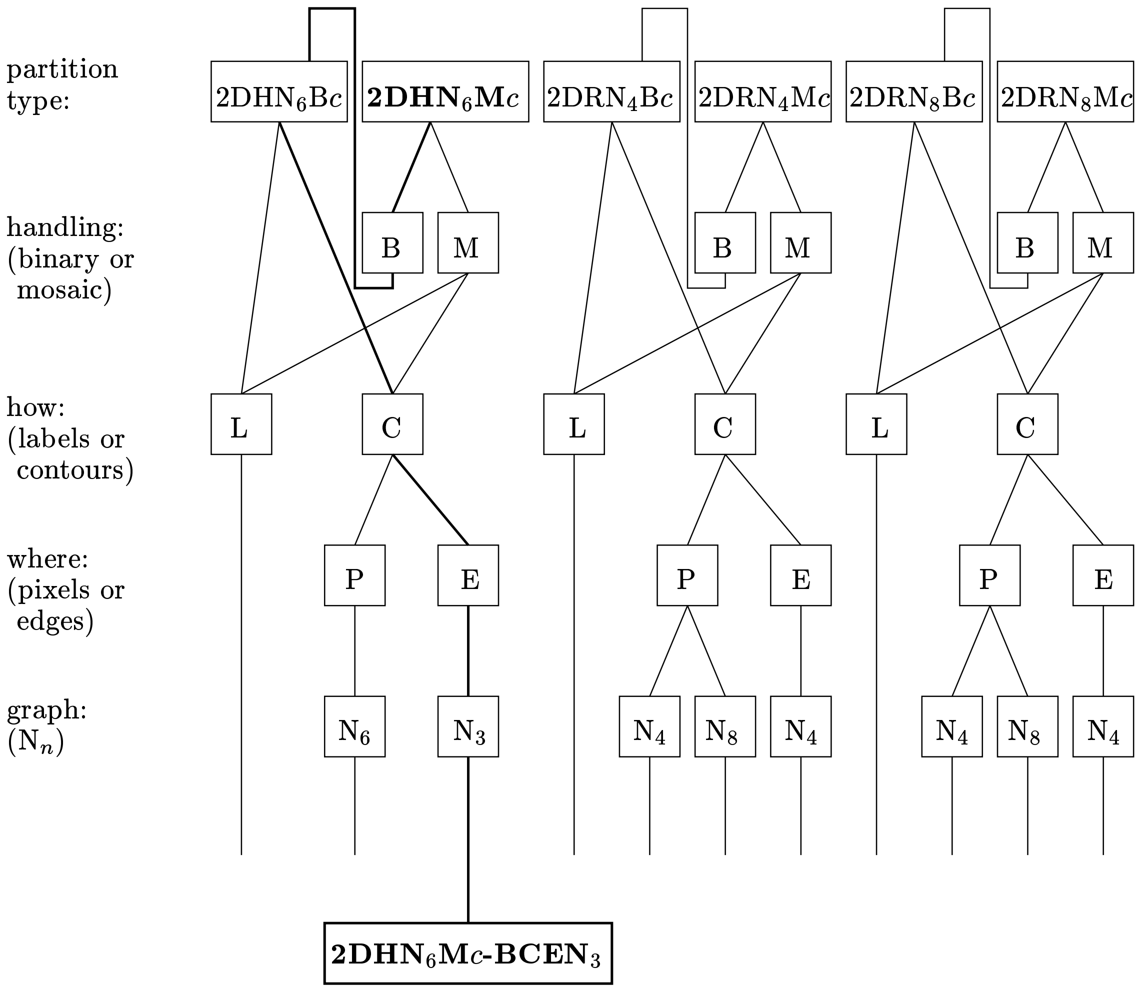


Figure 6.4: The partition representation taxonomy tree for the 2D case (in bold, the example given in the text). c stands for either C (connected classes) or D (disconnected classes).

3D partitions

As can be seen in Figure 6.5, for 3D partitions two representations may be considered: stick to three dimensions [3D], or slice the partition along the time domain and use 2D methods [2D]. Prediction of the 2D partition slices can be used [Inter], otherwise the 2D partitions are considered independent [Intra]. When prediction is used, it may [M] or may not [F] use motion compensation (the 'M' stands for "motion" while the 'F' stands for "fixed"). The motion information may be either estimated from the 3D partition [61] or input from external sources (e.g., from a motion estimator working with the original 3D image). Notice that the slicing to two dimensions establishes a connection to one of the 2D branches at the top of the representation taxonomy shown in Figure 6.4, depending on the type of the resulting 2D (possibly predicted) partitions.

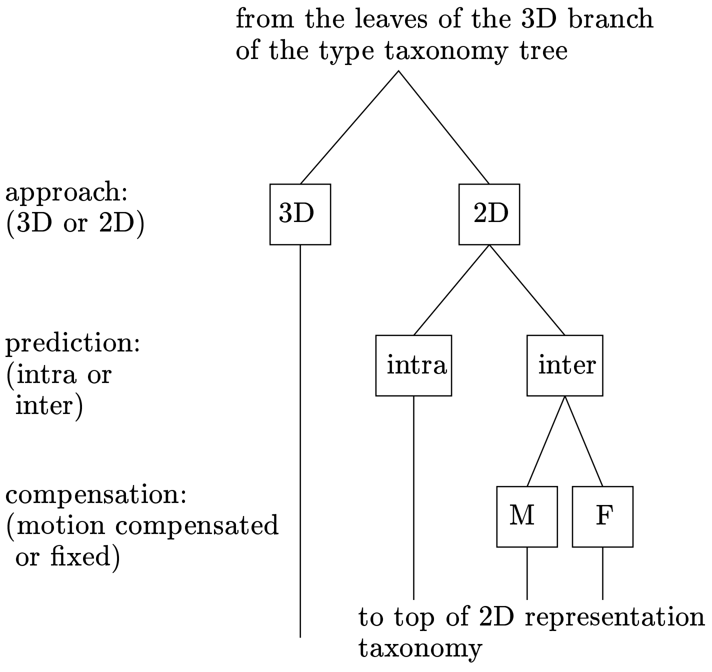

from the leaves of the 3D branch of the type taxonomy tree
approach: (3D or 2D)
3D
2D
prediction: (intra or inter)
intra
inter
compensation: (motion compensated or fixed)
M
F
to top of 2D representation taxonomy

Figure 6.5: The partition representation taxonomy tree for the 3D case.

6.2.3 Representation properties

Choosing the representation for the partitions (of a given type) depends on the properties of each representation and how adequate they are for the task at hand. Pros and cons related with some of the levels of the 2D partition representation taxonomy tree are listed below:

- Handling (only for mosaic partitions):

  Mosaic

  A single connected contour graph can separate several regions, which leads to coding efficiency when a contour representation is used; however, access to a single class shape is not easy, since the regions (and classes) are not represented individually.

Binary
The classes are represented independently, and thus easy access to each class is provided, though at the expense of less efficient coding efficiency.

- How:

Labels
In this case, the identification of the class to which each pixel in the partition belongs is very simple, though the shapes of the classes are not directly represented.

Contours
The shapes of the classes are directly represented, albeit at the expense of requiring somewhat involved algorithms to ascertain the class of a given pixel [155, 182, 6].

- Where:

Pixels
Representing contours on pixels poses a number of problems, especially in the case of mosaic partitions, since using all border pixels leads to unnecessary repetition at both sides of a border; when the problem is avoided by using only one side of each border, other problems arise: e.g., how should one pixel wide regions or parts of regions be distinguished from borders of thick regions. Although the problems associated with these representations have solutions, often somewhat involved, coding contours on pixels does not seem to achieve higher compression than coding contours on edges [31].

Edges
This is usually a more elegant way of representing contours, which in addition typically provides more compression than pixel based contours [31].

6.3 Overview of partition coding techniques

Once the type of partitions to code has been ascertained and a partition representation selected, according to the taxonomy defined in the previous section, there are usually a number of available coding techniques. This section overviews some of these techniques. Special attention will be payed to 2D partitions.

6.3.1 Lossless or lossy coding

The question of whether to use lossy partition coding techniques is an important one. It is true that some techniques that are inherently lossy, such as parametric curves, can yield good compression [73]. However, it may be difficult, for some applications, to establish sound partition coding quality criteria. Also, when the scene objects (corresponding possibly to classes or sets of classes) are to be manipulated individually, e.g., pasting an object into a different scene, the effects of lossy partition coding can be very important, since pieces of the real object may be lost, pieces of the background can be introduced, and even object deformation may occur. This seems to indicate that lossless partition coding techniques are preferable, and that simplifications should be introduced into the partitions carefully during the segmentation process, before partition coding.

However, if lossy coding is acceptable, the losses are usually constrained so that there is:

1. class topological equivalence, i.e., the classes should be maintained in number and adjacency relations—the CAG should not be altered (a stronger constraint can be imposed if the RAG, or even the RAMG, is not allowed to change); and/or

2. small displacement of borders, i.e., the borders between the regions should change as little as possible, according to some error criterion (other constraints may be imposed, for instance on errors associated with the area and position of the regions).

6.3.2 Mosaic vs. binary partitions

When easy access to the contents of the video sequence is required, the shapes of the various objects (e.g., a class or a set of classes in a partition) will have to be coded independently. This requirement can be imposed even if the segmentation process resulted in a mosaic partition, reducing the problem to the coding of a series of binary partitions (see the handling level in Figure 6.4).

The independent coding of binary partitions also arises naturally when a layered scene representation, as proposed by Wang and Adelson [195], is used. Layered representations of the scenes are also used in MPEG-4 [77]: each layer corresponds to a 2D object of arbitrary shape, whose time snapshots are called VOP (Video Object Plane). The shape of the objects represented by VOPs can be associated to binary partitions.[2] However, if the content of the VOPs were coded through region based techniques (as would happen if the Sesame [30] proposal had been included in MPEG-4), then mosaic partitions would also be necessary within each VOP.

Thus, both coding of binary and mosaic partitions may be important issues when easy access to the contents of the video sequences is required.

6.3.3 Partition models

The coding efficiency always depends on the characteristics of the partitions being coded. Most of the techniques aim at genericness, though this is a somewhat hard to define property. By genericness it is often meant that the techniques perform well on average. The problem with this definition is that often little is known about the statistics of the partitions which need to be

[2] Actually the shapes of the VOPs can be specified in MPEG-4 using "binary shape," i.e., a binary partition, or "grey scale shape," which is an alpha plane specifying the transparency of each pixel.

coded. This is a general problem in image processing: is there a statistical model for the images to process? In the case of partition coding, the statistical characterization of input partitions depends both on the original images and on the segmentation algorithm used upstream. Hence, most techniques do not address a specific model of input partitions, making only some general assumptions such as:[3]

1. the regions tend to contain a significant amount of pixels, i.e., small regions are improbable;
2. the classes tend to contain a small amount of regions;
3. the contours (borders between regions) tend to be simple (not ragged); and
4. the region interiors tend not to contain too many small holes.

6.3.4 Class coding

Class coding is necessary when:

1. class equivalence is enough;
2. the partitions used have disconnected classes (see connectivity level in Figure 6.3); and
3. the explicit labels of the partition pixels have not (yet) been coded (it is the case after contour coding techniques and some label coding techniques).

The objective of class coding is to establish which regions are grouped in the same class. This issue will not be discussed at length here. However, note that the coding methods used should take into account that:

1. the explicit class labels are not required, since class equivalence is enough; and
2. adjacent regions cannot belong to the same class, for otherwise they would be a single region (this can help reduce the amount of data to transmit).

If partition equality is required, then the class labels should be coded explicitly for each region in the partition. When the classes are connected, the fact that a given label appears only once can be used to reduce the amount of data to transmit, since the degrees of freedom keep reducing until zero when the next-to-last label is transmitted.

6.3.5 Label coding

Label coding techniques code partitions whose representation is based on pixel labels. The cases of binary and mosaic partitions will be addressed separately in the following.

[3]See for instance Chapter 10 of [68].

Binary partitions

Binary partitions can be seen as binary (or two-tone) images. Therefore, the techniques available for coding binary images are good candidates for coding binary partitions. While lossless techniques can be applied without any problems, lossy techniques often do pose some problems, since the type of losses they allow does not generally take into account the constraints typically used for lossy partition coding.

Reviews on binary image coding can be found in [90, 74] and, specifically for fax, in [76]. The lossless coding standards ITU-T T.4 and T.6 (Group 3 and Group 4 facsimile) [45, 46] and ITU-T T.82 JBIG [82] use techniques with increasing compression efficiency:

T.4 uses one-dimensional RLE (Run-Length Encoding) and, optionally, also the 2D MREAD (Modified Relative Element Address Designate) codes, both followed by VLC. In the 2D mode, each k line is coded with RLE (k is set to 2 for low resolution images and to 4 for high resolution images), while all the other lines are coded with MREAD.

T.6 is similar to ITU-T T.4, though the 2D mode is always used and k is set to infinite, so that only MREAD is used. The resulting codes are called MMREAD (Modified MREAD).

T.82 uses the arithmetic Q-Coder [159] to code the pixel values. The probabilities for the Q-Coder are estimated using a local context (a template) for the current pixel. Since JBIG uses resolution layers for progressive coding, two types of templates exist: the first is used in the lowest resolution layer and includes only pixels already transmitted in that layer, while the second is used for all the other layers and includes not only pixels from the current layer but also from the layer immediately below in resolution.

A technique based on a modified MMREAD code, on 16×16 blocks, has been proposed for the coding of binary alpha maps in the framework of MPEG-4 [188]. This technique has been adopted in VM3 [2] after a round of core experiments on binary shape coding [140]. Two techniques with relations to JBIG [11, 12] have also been evaluated during the core experiments. Both use arithmetic codes with probabilities estimated from a local context around the pixel to be coded. The technique which was later approved for inclusion in the MPEG-4 CD (Committee Draft) [77] is of this latter type.

Among all the other techniques that have been proposed for binary partition coding, morphological skeletons [103] (and more recently [83]) are especially relevant, mainly because this technique has evolved lately to efficiently cover also mosaic partitions [14]. This technique represents the shape of a region by a set of skeleton points and a so-called quench function: the region is the union of structuring elements (of a certain shape) centered on the skeleton points and scaled according to the value of the quench function at that point.

Since binary partitions are a special case of mosaic partitions, techniques developed for the latter may also be applied to the former, either directly or with simplifying changes, despite the fact that they do not take into account the special characteristics of binary partitions.

Mosaic partitions

The case of mosaic partitions is more complex. The coding of mosaic partitions has received less attention than the coding of binary partitions (however, see [14, 13, 191]). It is possible, nevertheless, to use binary partition coding techniques by first converting the mosaic partitions into bit planes. For instance, using the FCT (see Section 3.4.3), the regions in a partition can be perfectly identified by painting them with only four colors. Hence, each region can be identified by a two-bit label, and thus two bit-planes are sufficient for representing the partition. Each of the two bit-planes can be coded independently using (lossless) binary partition coding techniques. Notice that some borders are present in both bit-planes, so this method cannot yield optimal results.

A technique using the concept of geodesic skeleton, where the regions are described by a set of skeleton points and a quench function [14], was recently proposed. This technique is, in a sense, an extension of the technique proposed in [103] for binary partitions. The authors claim that "the geodesic skeleton is preferable to chain code whenever there are many isolated and short contour arcs to be coded," which seems to be the case when 3D...-2DInterM (motion predicted 2D partitions corresponding to time slices of a 3D partition) partition representations are used.

A method which is also related to geodesic skeletons has been proposed in [191, 171]. It represents regions as a union of structuring elements with appropriate translations and scalings. Both techniques ([14, 191]) allow the structuring elements to overlap already coded regions, thus avoiding duplicate coding of borders and reducing the required bitrate. Both techniques are lossy and, again, can be used for mosaic and binary partitions. Incidentally, it may be noted here that the problem of finding the minimum number of rectangles covering a given set of elements in a matrix can be show to be NP-complete, see [51, SR25, p.232].

Another interesting technique, based on Johnson-Mehl tessellations, has been proposed in [13] (which contains a good review of partition coding techniques). The idea is to find germs (and their germinating time) for each region such that the original partition is reproduced well when the germs are allowed to grow until reaching other growing germs. Though the technique proposed is lossy, it can easily be made lossless. According to the authors, the technique performed worse than the other techniques studied (straight line and polygonal approximation, chain codes, and geodesic skeletons).

6.3.6 Contour coding

At least three breeds of contour coding techniques can be distinguished:

Chain codes
The contour graph is coded by a string of symbols representing the direction of the "chain" connecting a vertex to the next vertex on the contour. Each of these strings is called a chain code. Symbols may also represent direction changes, which makes the chain codes differential.

Parametric curves
The contours are approximated by parametric curves, whose coefficients are then coded; the most common examples are approximations by straight lines and by splines (in general, by polynomials).

Transform codes
The contours are represented as parametric curves which are coded using transform methods, in a one-dimensional equivalent of transform coding for images.

All these techniques involve two steps: first the representation is changed by transforming the contours into strings of symbols (e.g., changes in chain direction, spline parameters, control points or transform coefficients—possibly quantized) and then these symbols are entropy coded.

For contours defined on pixels, it is also possible to use techniques developed for binary image coding. The idea is to paint black, against a white background, all the border pixels in the partition and then use one of the binary label coding techniques already discussed. Notice, however, that lossless techniques should in general be used, since lossy techniques were not usually developed with partition coding in mind.

Chain codes

The contour graph is a subgraph of either the line graph (for contours defined on edges) or the image graph (for contours defined on pixels), and usually consists of a collection of trails on the original graph. Each contour trail can thus be represented by a string of symbols representing which of the neighbors of the current graph vertex belongs to the contour trail or, which is the same, the direction of the "chain" connecting it to the next vertex on the contour trail: these strings are called chain codes [49, 50, 201]. When the symbols represent direction changes, the chain codes are said to be differential [42, 58]. The simplest partitions are those for which the contour graph is constituted of disconnected closed trails.

Binary partitions are generally simpler to code than mosaic partitions. The main difference stems from the fact that, for binary partitions, all vertices in the contour graph (at least for edge contours graphs) have an even number of neighbors: two vertices for the N_3 line graph corresponding to the N_6 image graph (used for hexagonal sampling lattices), and two or four vertices for the N_4 line graph corresponding to the N_4 image graph (used for rectangular sampling lattices). That is, the connected components of such graphs have Euler trails, i.e., they can be "drawn without lifting the pencil".

Mosaic partitions with contours defined on edges require special treatment, since the existence of junction vertices (vertices with degree 3, see Figure 3.12) precludes the definition of contours as disconnected closed trails. There are at least two ways of dealing with this problem:

1. Ignore junctions and crossings. Select one of the exits and leave the others for coding as separate contours; since initial contour points are costly to code, this solution is not optimal.

2. Code junctions and crossings explicitly [106]. Select one of the exits but code also information about the junction or crossing so that later one can "return" and continue following the remaining exits (one in the case of a junction, two in the case of a crossing).

When junctions and crossings are explicitly coded, or when retracing of contour segments is allowed, the compression obtained when coding a connected component of a contour depends strongly on the way the connected component is followed: where to start, which exit to follow first at each junction or crossing, etc. The problem of coding can then be seen as a problem of minimizing the bitrate given a certain syntax of representation. This problem is similar to Chinese postman problem, i.e., to the problem of making a line drawing without lifting the pencil and minimizing the length of the redrawn lines, which is solvable in polynomial time. Its solution can be accelerated if the drawing schedule is determined in the RBPG of the partition where each arc has an appropriate weight. The advantage stems from the fact that the RBPG has a smaller number of vertices and arcs. Each arc in the RBPG, which corresponds to a complete border on the partition, should have a weight which is proportional to the number of bits required to encode it using chain codes. A rough approximation would be to make the weight proportional to the number of edges in the border. Otherwise the weight might be estimated from statistics obtained of previous encodings.

When contours are defined on pixels, the concepts of junction and crossing require a more involved definition and treatment [101, 31]. In the case of binary partitions, the problem may be solved by again ignoring the presence of vertices of degree larger than two in the pixel contour graph. Another problem of contours defined on pixels is posed by one pixel wide regions or parts of regions, which make it difficult to use a stopping condition as simple as "Stop when the initial vertex of the contour is attained", which is often used when coding contours defined on edges. Such regions may also require the existence of a turning back (180°) direction in the chain codes, rarely used, which may cause some VLCs to be inefficient (for instance Huffman).[4]

In general, chain codes correspond to the specification of a subgraph, consisting of a set of trails, in the underlying image or line graph. A contour connected component consists of a set of trails linked at junctions and crossings. Each trail can be represented by:

1. a position for the first vertex of the trail, maybe implicitly indicated in a previous crossing or junction information; and
2. a string of symbols, the chain codes, which may include crossings and junctions information.

Both the first vertex position and the chain codes are then entropy coded. The construction of the chain codes may also include contour simplification procedures.

Several techniques have been proposed in the literature for entropy coding the initial vertices and the chain codes:

[4]Consider an alphabet consisting of two symbols A and B with equal probabilities 0.5: the corresponding Huffman code will have one bit per symbol. If a third, improbable but possible, symbol C is added, and the probabilities are $p(A) = 0.495$, $p(B) = 0.495$, $p(C) = 0.01$, the number of bits per code word will be 1, 2, and 2, respectively. The average number of bits per symbol will be 1.505, 40% worst than the minimum of 1.071.

1. zero order Huffman and arithmetic coding (adaptive or not) [133, 106], which tend to be inefficient, since region borders are usually very different from a Brownian random walk through the image or line graph;
2. nth order Huffman and arithmetic coding (adaptive or not) [31, 133, 42];
3. Ziv-Lempel (or Ziv-Lempel and Welsh) coding [202, 197], which is a form of "dictionary-based coding" [90]; and
4. run-length coding, which groups chain codes into runs of related symbols [86, 133], usually corresponding to straight line segments [93, 10, 133] (and hence constituted either of a single symbol or of two symbols, with adjacent directions, which verify the conditions defined by Rosenfeld in [173]).

In the framework of the MPEG-4 core experiments on binary shape coding [140], extensions to basic or differential chain codes have been proposed. In [52, 140] a lossy multi-grid chain code is proposed which, according to the authors, reduces by an average of 25% the coding cost with respect to differential chain codes. In [196] a method is proposed which decomposes a (differential) chain code into two chain codes with half the resolution, plus additional codes if lossless coding is desired.

Parametric curves

These techniques approximate contours (or contour segments) by parametric functions, usually polynomials. The functions can usually be represented by either a set of coefficients or a set of control points [175, 43]. The coefficients or the coordinates of the control points are quantized and then entropy coded. Notice that when polynomials of degree one are used (with rectangular coordinates), the contours are approximated by polygons. The use of control points [152] simplifies the quantization process, since it is simpler to control the errors introduced by quantizing the coordinates of control points than the errors introduced by quantizing the coefficients of a polynomial. In the case of mosaic partitions, the crossings and junctions of contours are frequently selected as control points [43, 97].

One of the most important problems in parametric curve representation of contours is error control. Iterative techniques are commonly used which successively split the contour until a sufficiently small approximation error is obtained for each resulting segment [43, 97]. The error is frequently calculated from the geometrical distance between the parametric curves and the real contours [97, 53], but some researchers propose the use of the contrast across the contours, assuming it is available [43]. Methods have also been proposed which follow the split phase by a merge phase [157, 154]. Such split & merge methods for polygonal approximation of contours were the precursors of similar methods used later in image segmentation.

When control points are used, their differences along the contour graph are usually entropy coded. These methods deal with junctions and crossings in a very similar way to chain coding techniques.

As part of the MPEG-4 core experiments on binary shape coding [140], parametric curve techniques have also been evaluated [53, 148, 89, 25] (some of these techniques stem from the earlier [73]). These techniques approximate the contours with polygons or splines using a set of control points chosen again with a split algorithm. The selection of which approximation method to use is either done for each contour segment (between control points) or for each object. The proposed techniques also take advantage of time redundancy between control points along the successive partitions. One-dimensional transform coding methods, some of which multi-resolution, are proposed to compensate the residual error between the parametric curve approximation and the actual contours (see the next section).

Transform codes

The contours are represented first as parametric curves taking values in $\mathbb{R}$, if the contour (or contour segment) being coded can be represented by a polar function centered somewhere in the image, or in $\mathbb{R}^2$ for other kinds of contour (or contour segments). These parametric curves (still a lossless representation) are then coded using transform methods [22], in a one-dimensional equivalent of the transform coding used in image coding (e.g., DCT), i.e., the parametric curves are transformed and the resulting coefficients are quantized and entropy coded.

Transform codes have also been under scrutiny in the MPEG-4 core experiments on binary shape coding [140], both for contour coding proper and for coding the residual error after using parametric curve methods.

The first technique considered in the core experiments uses a polar representation of the contour [24]. The contour is represented by a function of the polar angle, whose value is the distance between the centroid and the contour in the direction defined by the angle.[5] The one-dimensional DCT of the distance function is calculated and then its coefficients are quantized and VLC coded. Some contours cannot be properly represented by a parametric function of the polar angle (since more than one contour point may occur for a single angle). Hence, parts of the contour may have to be left out. These parts are handled separately using chain codes. This technique can also take advantage of the temporal redundancy between successive partitions.

The other transform coding techniques tested on the MPEG-4 core experiments use either the one-dimensional DST or DCT to code not the contour itself, but the residual error (distance) between a parametric curve approximation and the actual contour [148, 89, 25]. In [25] the distance between the approximate and actual contours is calculated either horizontally or vertically, depending on the slope of the line between the control points of the contour segment being encoded. This substantially reduces the calculations relative to the usual orthogonal distance method. In [148] a multi-resolution version of the DST is used, so as to provide contour (object) scalability.

[5]The centroid is the point whose coordinates are the average of the coordinates of all the pixels in the region enclosed by the contour.

6.4 A quick cubic spline implementation

Splines are a form of parametric approximation of contours. In the framework of the COST 211ter project, the SIMOC1 reference model [39], made use of a mixture of polygonal and cubic spline approximation [26] to the contours of a binary partition (a change detection mask). Among several proposals made by the author related with that reference model, namely [116, 114, 79, 78], an improvement on the original specification of cubic splines and a fast implementation method are especially relevant.

The improvement stems from considering continuity of first and second derivatives at all nodes of the spline, since the spline is being used to approximate a closed curve (a contour), given a number of control points. The determination of the spline coefficients is similar to the usual cubic spline, except that the matrix which needs to be inverted as part of the algorithm has a type of symmetry which makes it suitable for efficient implementations.

6.4.1 2D closed spline definition

Given a sequence of n control points in $\mathbb{R}^2$

$$s_0, s_1, \ldots, s_{n-1}, \text{ with } s_j = \begin{bmatrix} x_j & y_j \end{bmatrix}^T,$$

the aim is to find n cubic polynomials $P_j(t) = \begin{bmatrix} p_{x_j}(t) & p_{y_j}(t) \end{bmatrix}^T$ from $t \in [j, j+1[$ (uniform parameterization) to $\mathbb{R}^2$, with $j = 0, \cdots, n-1$, which result in a smooth interpolation.

To achieve the desired smoothness, the same set of constraints (6.2) is imposed for each component of the polynomials, $p_{x_j}(\cdot)$ and $p_{y_j}(\cdot)$, in the sequel referred to generically as $p_j(\cdot)$. Let also f_j be x_j or y_j according to whether $p_j(\cdot) = p_{x_j}(\cdot)$ or $p_j(\cdot) = p_{y_j}(\cdot)$.

Each polynomial is defined by the four parameters a_j, b_j, c_j, and d_j

$$p_j(t) = a_j + b_j(t-j) + c_j(t-j)^2 + d_j(t-j)^3 \text{ with } j = 0, \ldots, n-1. \tag{6.1}$$

The constraints impose continuity up to the second derivative at each node[6]

$$\begin{array}{rcl} p_j(j) & = & f_j \\ p_j(j+1) & = & f_{\overline{j+1}} \\ p'_j(j+1) & = & p'_{\overline{j+1}}(j+1) \\ p''_j(j+1) & = & p''_{\overline{j+1}}(j+1) \end{array} \text{ with } j = 0, \ldots, n-1, \tag{6.2}$$

where $\overline{j} = j \bmod n$ and $p'_j(\cdot)$ and $p''_j(\cdot)$ are respectively the first and second derivative of polynomial $p_j(\cdot)$.

From (6.1) it can be seen that $4n$ coefficients must be found by the spline algorithm, using the $4n$ restrictions given by (6.2).

[6]Notice that the selected restrictions do not generally assure smoothness of the final 2D curve, only of the individual parametric curves!

6.4.2 Determination of the spline coefficients

Substituting (6.1) in (6.2), one obtains

$$\left.\begin{array}{rcl} a_j &=& f_j \\ a_j + b_j + c_j + d_j &=& f_{\overline{j+1}} \\ b_j + 2c_j + 3d_j &=& b_{\overline{j+1}} \\ 2c_j + 6d_j &=& 2c_{\overline{j+1}} \end{array}\right. \quad \text{with } j = 0, \dots, n-1.$$

Simple algebraic manipulations (see [26]) lead to the following solution

$$\left.\begin{array}{rcl} a_j &=& f_j \\ d_j &=& \frac{c_{\overline{j+1}} - c_j}{3} \\ b_j &=& a_{\overline{j+1}} - a_j - \frac{2c_j + c_{\overline{j+1}}}{3} \end{array}\right. \quad \text{with } j = 0, \dots, n-1,$$

and

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_j \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{bmatrix} = A_n^{-1} 3 \begin{bmatrix} a_1 - 2a_0 + a_{n-1} \\ a_2 - 2a_1 + a_0 \\ \vdots \\ a_{\overline{j+1}} - 2a_j + a_{\overline{j-1}} \\ \vdots \\ a_{n-1} - 2a_{n-2} + a_{n-3} \\ a_0 - 2a_{n-1} + a_{n-2} \end{bmatrix},$$

where A_n is the $n \times n$ matrix

$$A_n = \begin{bmatrix} 4 & 1 & & & 1 \\ 1 & 4 & 1 & & \\ & & \ddots & & \\ & & 1 & 4 & 1 \\ 1 & & & 1 & 4 \end{bmatrix}.$$

6.4.3 Approximate algorithm

The main part of the spline algorithm is the inversion of matrix A_n $(n \times n)$. If observed carefully, matrix A_n has a special kind of structure, originated in the imposed spline restrictions which treat all nodes equally: each line (column) of A_n may be obtained by rotating the previous one to the right (down), the last element of the line (column) being rotated back to the beginning. Another interesting characteristic of A_n is that each line (column) is symmetric around its diagonal element.

It turns out, as can be proved analytically, that the inverse of A_n has exactly the same structure. This means that A_n^{-1} can be constructed from the knowledge of the first half of its first line, viz. $n///2 + 1$ elements where $///$ stands for truncating integer division. Let those elements

		i					
		0	1	2	3	4	5
	3	0.2777777778	-0.0555555556				
	4	0.2916666667	-0.0833333333	0.0416666667			
	5	0.2878787879	-0.0757575758	0.0151515152			
	6	0.2888888889	-0.0777777778	0.0222222222	-0.0111111111		
	7	0.2886178862	-0.0772357724	0.0203252033	-0.0040650407		
n	8	0.2886904762	-0.0773809524	0.0208333333	-0.0059523810	0.0029761905	
	9	0.2886710240	-0.0773420479	0.0206971678	-0.0054466231	0.0010893246	
	10	0.2886762360	-0.0773524721	0.0207336523	-0.0055821372	0.0015948963	-0.0007974482
	11	0.2886748395	-0.0773496789	0.0207238762	-0.0055458260	0.0014594279	-0.0002918856
	12	0.2886752137	-0.0773504274	0.0207264957	-0.0055555556	0.0014957265	-0.0004273504
	13	0.2886751134	-0.0773502268	0.0207257938	-0.0055529485	0.0014860003	-0.0003910527

Table 6.1: The first 6 $q_i(n)$ for $n = 3, \ldots, 13$.

be $q_0(n), \ldots, q_{n///2}(n)$ (the value of the elements depends on n, of course). Table 6.1 shows the evolution of these elements. Notice that it is assumed that $n \geq 3$, since $n = 1$ and $n = 2$ conduce to degenerate splines. Notice also that, for $i \geq 2$, element $q_i(n)$ only exists if $2i \leq n$.

An example may clarify the structure of the matrices. Suppose the inverse of A_6 is to be calculated. The first step is to copy the elements in row 6 of Table 6.1 to the first half of the first line of A_6^{-1}

$$A_6^{-1} = \begin{bmatrix} 0.28889 & -0.07778 & 0.02222 & -0.01111 & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix}.$$

The second step is to reflect the elements of the first line around the diagonal element

$$A_6^{-1} = \begin{bmatrix} 0.28889 & -0.07778 & 0.02222 & -0.01111 & 0.02222 & -0.07778 \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix}.$$

Finally, the remaining lines are filled by successive rotation of the first line

$$A_6^{-1} = \begin{bmatrix} 0.28889 & -0.07778 & 0.02222 & -0.01111 & 0.02222 & -0.07778 \\ -0.07778 & 0.28889 & -0.07778 & 0.02222 & -0.01111 & 0.02222 \\ 0.02222 & -0.07778 & 0.28889 & -0.07778 & 0.02222 & -0.01111 \\ -0.01111 & 0.02222 & -0.07778 & 0.28889 & -0.07778 & 0.02222 \\ 0.02222 & -0.01111 & 0.02222 & -0.07778 & 0.28889 & -0.07778 \\ -0.07778 & 0.02222 & -0.01111 & 0.02222 & -0.07778 & 0.28889 \end{bmatrix}.$$

	i			
n	0	1	2	3
3	0.2777777778	-0.0555555556		
4	0.2916666667	-0.0833333333	0.0416666667	
5	0.2878787879	-0.0757575758	0.0151515152	
6	0.2888888889	-0.0777777778	0.0222222222	-0.0111111111
7	0.2886762360	-0.0773524721	0.0207336523	-0.0055821372
8	0.2886762360	-0.0773524721	0.0207336523	-0.0055821372
9	0.2886762360	-0.0773524721	0.0207336523	-0.0055821372
10	0.2886762360	-0.0773524721	0.0207336523	-0.0055821372
11	0.2886762360	-0.0773524721	0.0207336523	-0.0055821372
12	0.2886762360	-0.0773524721	0.0207336523	-0.0055821372
13	0.2886762360	-0.0773524721	0.0207336523	-0.0055821372

Table 6.2: The approximate elements $\hat{q}_i(n)$ for $n = 3, \ldots, 13$.

Hence, one method for inversion of matrices A_n corresponds to storing the first half of the first row of A_n^{-1} in a lookup table for several values of n and to apply the steps above. Should this method be too memory demanding for the given application, an approximation can be used as derived below.

One interesting fact about the elements $q_i(n)$ (see Table 6.1) is that they are negligible for high enough i. In particular for $i \geq 4$, since $|q_i(n)| < 0.003$ ($\forall n$ and $i \geq 4$) which is very small compared to $|q_0(n)|$ (always about 0.29). It is thus possible to approximate the inversion of A_n by considering all $q_i(n)$ elements to be zero for $i \geq L$ (e.g., $L = 4$).

The elements $q_i(n)$ form a rapidly convergent succession with n, for each i. Hence, a good approximation to all $q_i(n)$ with $i < L$ and $n > N$ is $q_i(M)$, provided M ($\geq N$) and N are large enough numbers.

Having the described behavior in mind, the following approximation was implemented:

$$\hat{q}_i(n) = \begin{cases} q_i(n) & \text{if } n \leq N, \\ q_i(M) & \text{if } n > N \text{ and } i < L, \text{ and} \\ 0 & \text{if } n > N \text{ and } i \geq L; \end{cases}$$

with L set to 4, M to 10 and N to 6. The approximate element values $\hat{q}_i(n)$ can be seen in Table 6.2.

The approximation presented is quite good, though the degree of accuracy of the algorithm may be further improved by increasing L, N and/or M.

Implementation

The inverse of a matrix can be calculated as its transposed adjoint divided by its determinant

$$A^{-1} = \frac{\mathrm{adj}^T(A)}{\det(A)}.$$

The calculation of the adjoint and the determinant of a matrix involves only sums and multiplications, hence, if the elements of A_n have integer values, $\det(A)$ and the elements of $\mathrm{adj}(A)$ will also have integer values. This is the case for the particular A_n considered in this work. Since the values f_j also have integer values, the spline algorithm can be implemented using solely integer arithmetic. See Algorithm 3.

6.4.4 Exact algorithm

The matrix to invert as part of the spline algorithm is a special case of a more generic class of matrices. Matrices whose columns can be obtained by successively rotating the left column upwards (or downwards) are called column circulant matrices. Matrices whose rows can be obtained by successively rotating the top row rightwards (or leftwards) are called row circulant matrices. Row circulant matrices can be transformed into column circulant matrices simply by transposition. If a matrix is both row and column circulant, then it is also symmetric.

An efficient way of inverting a circulant matrix can be easily devised using the DFS (Discrete Fourier Series), or its fast algorithmic version, the FFT (Fast Fourier Transform).

Let A be a $n \times n$ non-singular downwards column circulant matrix with first column a ($\downarrow$ is the downwards rotation operator)

$$A = \begin{bmatrix} a & a{\downarrow_1} & \cdots & a{\downarrow_{n-1}} \end{bmatrix},$$

with

$$a = \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}.$$

Let $b = \mathcal{F}(a)$, where $\mathcal{F}(\cdot)$ is the DFS, and also let

$$c = \begin{bmatrix} c_0 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} \frac{1}{b_0} \\ \vdots \\ \frac{1}{b_{n-1}} \end{bmatrix},$$

then A^{-1} can be calculated as

$$A^{-1} = \begin{bmatrix} d & d{\downarrow_1} & \cdots & d{\downarrow_{n-1}} \end{bmatrix},$$

where $d = \mathcal{F}^{-1}(c)$, with $\mathcal{F}^{-1}(\cdot)$ the inverse DFS.

In the case of the spline algorithm,

$$a = \begin{bmatrix} 4 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix},$$

Algorithm 3 Fast closed cubic spline algorithm (in C).

```
/*
 * word and dword are signed integer types with 16 and 32 bits.
 * af corresponds simultaneously to the f vector of points to interpolate
 * and to the a vector of spline parameters.
 * n is the number of nodes in the spline.
 * det3 is det(A)/3.
 * b, c, and d are scaled output vectors of spline parameters.
 * b and d should be divided by det(A)=det3*3 and c by det3 to obtain the
 * corresponding spline parameters.
 */
void spline(dword *b, dword *c, dword *d, dword *af, dword n, dword *det3)
{
    word j;
    dword a0, a1, a2, a3, a4, a5, b0, b1, b2, b3, b4, b5, det;

    /* d will temporarily store the RHS of the matrix equation: */
    d[0] = af[1] - (af[0] << 1) + af[n-1];
    for(j = 1; j < n-1; j++)
        d[j] = af[j+1] - (af[j] << 1) + af[j-1];
    d[n-1] = af[0] - (af[n-1] << 1) + af[n-2];

    /* Efficient and approximate matrix inversion, calculation of c: */
    if(n == 3) {
        *det3 = 6;
        c[0] = d[0] * 5 - d[1] - d[2];
        c[1] = d[1] * 5 - d[2] - d[0];
        c[2] = d[2] * 5 - d[1] - d[0];
    }
    else if(n == 4) {
        *det3 = 64;
        a0 = d[0] << 4; a1 = d[1] << 4; a2 = d[2] << 4; a3 = d[3] << 4;
        c[0] = d[0] * 56 - a1 + (d[2] << 3) - a3;
        c[1] = d[1] * 56 - a2 + (d[3] << 3) - a0;
        c[2] = d[2] * 56 - a3 + (d[0] << 3) - a1;
        c[3] = d[3] * 56 - a0 + (d[1] << 3) - a2;
    }
    else if(n == 5) {
        *det3 = 22;
        a0 = d[0] * 5; a1 = d[1] * 5; a2 = d[2] * 5; a3 = d[3] * 5;
        a4 = d[4] * 5;
        c[0] = d[3] - a4 + d[0] * 19 - a1 + d[2];
        c[1] = d[4] - a0 + d[1] * 19 - a2 + d[3];
        c[2] = d[0] - a1 + d[2] * 19 - a3 + d[4];
        c[3] = d[1] - a2 + d[3] * 19 - a4 + d[0];
        c[4] = d[2] - a3 + d[4] * 19 - a0 + d[1];
    }
    else if(n == 6) {
        *det3 = 30;
        a0 = d[0] * 7; a1 = d[1] * 7; a2 = d[2] * 7; a3 = d[3] * 7;
        a4 = d[4] * 7; a5 = d[5] * 7;
        b0 = d[0] << 1; b1 = d[1] << 1; b2 = d[2] << 1; b3 = d[3] << 1;
        b4 = d[4] << 1; b5 = d[5] << 1;
        c[0] = b4 - a5 + d[0] * 26 - a1 + b2 - d[3];
        c[1] = b5 - a0 + d[1] * 26 - a2 + b3 - d[4];
        c[2] = b0 - a1 + d[2] * 26 - a3 + b4 - d[5];
        c[3] = b1 - a2 + d[3] * 26 - a4 + b5 - d[0];
        c[4] = b2 - a3 + d[4] * 26 - a5 + b0 - d[1];
        c[5] = b3 - a4 + d[5] * 26 - a0 + b1 - d[2];
    }
    else {
        *det3 = 418;
        c[0] =     d[n-3] * -7 + d[n-2] * 26 - d[n-1] * 97 + d[0]   * 362 - d[1]   * 97 + d[2]   * 26 - d[3]   * 7;
        c[1] =     d[n-2] * -7 + d[n-1] * 26 - d[0]   * 97 + d[1]   * 362 - d[2]   * 97 + d[3]   * 26 - d[4]   * 7;
        c[2] =     d[n-1] * -7 + d[0]   * 26 - d[1]   * 97 + d[2]   * 362 - d[3]   * 97 + d[4]   * 26 - d[5]   * 7;
        for(j = 3; j < n - 3; j++)
            c[j] = d[j-3] * -7 + d[j-2] * 26 - d[j-1] * 97 + d[j]   * 362 - d[j+1] * 97 + d[j+2] * 26 - d[j+3] * 7;
        c[n-3] =   d[n-6] * -7 + d[n-5] * 26 - d[n-4] * 97 + d[n-3] * 362 - d[n-2] * 97 + d[n-1] * 26 - d[0]   * 7;
        c[n-2] =   d[n-5] * -7 + d[n-4] * 26 - d[n-3] * 97 + d[n-2] * 362 - d[n-1] * 97 + d[0]   * 26 - d[1]   * 7;
        c[n-1] =   d[n-4] * -7 + d[n-3] * 26 - d[n-2] * 97 + d[n-1] * 362 - d[0]   * 97 + d[1]   * 26 - d[2]   * 7;
    }
    det = 3 * *det3;

    /* Calculation of b and d: */
    for(j = 0; j < n-1; j++) {
        b[j] = (af[j+1] - af[j]) * det - (c[j] << 1) - c[j+1];
        d[j] = c[j+1] - c[j];
    }
    b[n-1] = (af[0] - af[n-1]) * det - (c[n-1] << 1) - c[0];
    d[n-1] = c[0] - c[n-1];
}
```

so c_i is

$$c_i = \frac{1}{4 + 2\cos(i\frac{2\pi}{n})}.$$

Hence, the exact computation requires only the calculation of one FFT, whose run time is $O(n \ln n)$.

6.4.5 Results

In order to assess the accuracy of the approximate algorithm, a few simple experiments were done:

1. Generate 100 sets of n random control points each fitting into a window slightly smaller than QCIF: $x_j \in [10, 165]$ and $y_j \in [10, 133]$.
2. Solve the spline problem for each of the 100 sets of nodes using the approximate algorithm and a non-approximate algorithm.
3. For each set, calculate the maximum error between the two versions of the spline. This error is calculated independently in x (e_x) and y (e_y).
4. Calculate an upper bound for the error between the two lines[7]: $\sqrt{e_x^2 + e_y^2}$.
5. Calculate the maximum of the upper bounds calculated for each set.

The above experiment was repeated for different numbers of nodes. Since the algorithm used is exact for $n \leq 6$, the experiments were done only for $n > 6$, viz. $n = 7$, 8, 9, 10, 15, 20, 40, and 100. The maxima of the upper bounds (which can be considered as estimates of the error upper bound for each number of nodes) are shown in Table 6.3.

Observation of the error table leads to the conclusion that the approximate algorithm produces errors of at most 1/3 of a pixel when working on control points located in a window of about QCIF size. Hence, it is a good candidate for efficient implementation.

6.5 Conclusions

A camera movement compensation method was presented in Section 6.1. The method has been shown to lead to small improvements in classical codecs such as H.261. As discussed, this does not render camera movement estimation useless, since it is important information to be around

[7]This is an upper bound for two reasons. The first is that the maximum errors in x and y usually do not occur for the same t. The second is that the error between two lines should actually be calculated as the maximum of the distance between any point in one of them to the nearest point in the other.

n	error upper bound
7	0.101775
8	0.304761
9	0.116038
10	0.230298
15	0.270953
20	0.28581
40	0.283622
100	0.299161

Table 6.3: Table showing the evolution with n of the estimated error upper bound.

for the end user to use in its manipulations of the contents of video sequences, besides being of use for indexing and camera stabilization.

A systematization of the field of partition coding has been proposed in Section 6.2. It has the form of a taxonomy tree which is divided in two main levels: partition type and partition representation. The proposed systematization is believed to simplify the comparison between partition coding techniques, by establishing clearly which type of partitions a given partition coding technique addresses, and which partition representation that technique is based on.

An overview of the partition coding techniques available for each partition type and the corresponding partition representations has been presented in Section 6.3.

Finally, some suggestions for efficient approximate implementations of closed cubic splines have been proposed in Section 6.4, which may be of use in parametric curve partition coding techniques.

Chapter 7

Conclusions: Proposal for a new codec architecture

In the previous chapters a series of analysis and coding tools were developed. Spatial analysis tools were developed in Chapter 4 and discussed in the unified framework of SSF and related graph theoretical concepts. Time analysis tools were developed in Chapter 5, where camera movement estimation algorithms and image stabilization methods have been proposed. Coding tools were developed in Chapter 6, which additionally proposed a systematization of the field of partition coding. The logical conclusion for this thesis is a proposal for a codec architecture that might integrate most of these tools. The next section contains such a proposal, which is followed by suggestions for future work and by the list of the thesis contributions.

7.1 Proposal for a second-generation codec architecture

This section proposes an architecture for a four-criteria (bitrate, distortion, cost, and content access effort) second-generation video codec. This work as already been published in [117], and it owes much to the fruitful discussions between the author and the Image Processing Group of the UPC (Universitat Politècnica de Catalunya), which later made their own contribution through the Sesame verification model proposal to MPEG-4 [30].

Possible source models for codecs with the proposed structure are discussed in this section. The main blocks of the codec, namely image and motion analysis, are also discussed and some possible solutions proposed.

From the point of view of this proposal, the MPEG-4 functionalities [139] considered are those addressing content access and improved coding efficiency. The objective is thus to minimize rate, distortion, and cost (i.e., maximize coding efficiency) as well as to minimize the content access effort (the "fourth criterion" [162]). Notice, however, that content manipulation is not addressed here. A simplified objective was deemed to be the provision for means to access easily the content of a video stream.

The main blocks of the proposed architecture are described from a functional point of view, and the requirements each block must fulfill are presented.

7.1.1 Source model

The source model selected is not new [40]: flexible 2D+1/2 objects. That is, objects are understood as 2D regions that can change shape over time (flexibility), and that can overlap each other. This is the same model used in Sesame. Each object's motion can be described by a single set of motion model parameters. The motion model used can be translational, affine, or more complex. As to textures, no explicit assumptions are made, except that motion boundaries should coincide with texture boundaries (except in pathological cases).

According to the adopted source model, objects are defined as a set of regions with coherent motion. Hence, image analysis, within this framework, is based essentially on motion analysis. This means that objects can be, and usually will be, inhomogeneous in terms of texture.

The approach taken by MPEG-4 has been different [77]. Essentially, MPEG-4 is an extension to previous MPEG standards providing "sequences" with arbitrary shapes, viz. VOs (Video Objects). Usually the VOs correspond to objects with some semantical meaning. MPEG-4 also provides a structured scene description as an acyclic graph of nodes specifying both synthetic and natural objects. In the later sense, MPEG-4 is also an extension of the VRML standard. Nevertheless, the inside (color or texture) of the VO, and its time evolution, is specified with techniques which stem directly from the previous MPEG standards and H.261 and H.263 [136, 137, 62, 63], even if some more modern approaches, such as sprites, meshes and face objects, are used. Notice, however, that the insides of sprites are also still encoded with techniques which can be classified as low-level vision, and that face objects, which correspond to a 3D scene description, can hardly be considered generic, since they apply only to a very particular type of scene. It can be said that MPEG-4 video is classical texture coding on arbitrarily shaped objects. Even if it indeed can be classified as a good step towards second-generation video coding, it is still in the transition. The Sesame [30] proposal to MPEG-4, which was not accepted due to its weaker performance, could, on the other hand, be classified as truly second-generation.

It is expected that MPEG-4 version 2, through its provision for programmable terminals, will allow more sophisticated architectures, such as the Sesame one or the one proposed here, to be developed independently and blended into the tool set of MPEG-4 to provide extended functionalities or capabilities.

In the following, the word image may be replaced by VOP, thus allowing the proposed architecture to be used for encoding of arbitrarily shaped sequences.

7.1.2 Codec architecture

Figures 7.1 and 7.2 show the proposed architecture for the codec. The main block is "Image analysis", which partitions each image into its composing objects, each with an estimated set of motion parameters. Motion parameters are (differentially) encoded by the "Motion parameter encoding" block. The "Partition encoding" block encodes the image partition differentially using the estimated motion for compensating the partition of the previous image. After "Motion compensation" of the previous decoded image, new objects and uncovered areas of the image are encoded using an intra mode. The "Detection of MF (Model Failure) areas" block detects areas in the image where the underlying source model fails. Those areas are then encoded by the "Encoding of MF partition" and "Encoding of MF texture" blocks.

The interface signals

Some interface signals have been defined in the block structure:

P, E Denote the image partition and the extra partition parameters. These signals represent the current partition and extra parameters related to its spatial and temporal structure. P may be simply a partition image, where each class label corresponds to an object, some special values being used to identify uncovered parts of objects. Notice that no constraint on the connectivity of objects was mentioned: an object can consist of a collection of disjoint connected regions (disconnected classes). E consists of some of the following extra information about the partition: number of class labels in use, maximum label in use, graph of occlusions, that is a graph specifying which regions overlap which, and information regarding the temporal evolution of objects, that is which classes are new and which no longer exist, or which classes where split or joined together.

P_{MF} Denotes the MF partition (see discussion below).

M Denotes the motion model parameters. A vector of motion model parameters for each object in the scene (new objects may lack this information, though). These parameters may be translational parameters, affine motion parameters, or parameters of some other more complex motion model. It may also include parameters of a given camera movement model used, used to compensate global motion in the sequence.

I Denotes an image.

Image analysis

The purpose of the "Image analysis" block is to describe the scene in terms of its component objects, according to the source model defined. As a first approach to the problem, this analysis is supposed to be done in a causal way and by looking at no more than two images at a time. This approach has been selected because real time and low delay video communications are envisaged. One should keep in mind, however, that for applications such as storage, these restrictions do not necessarily apply.

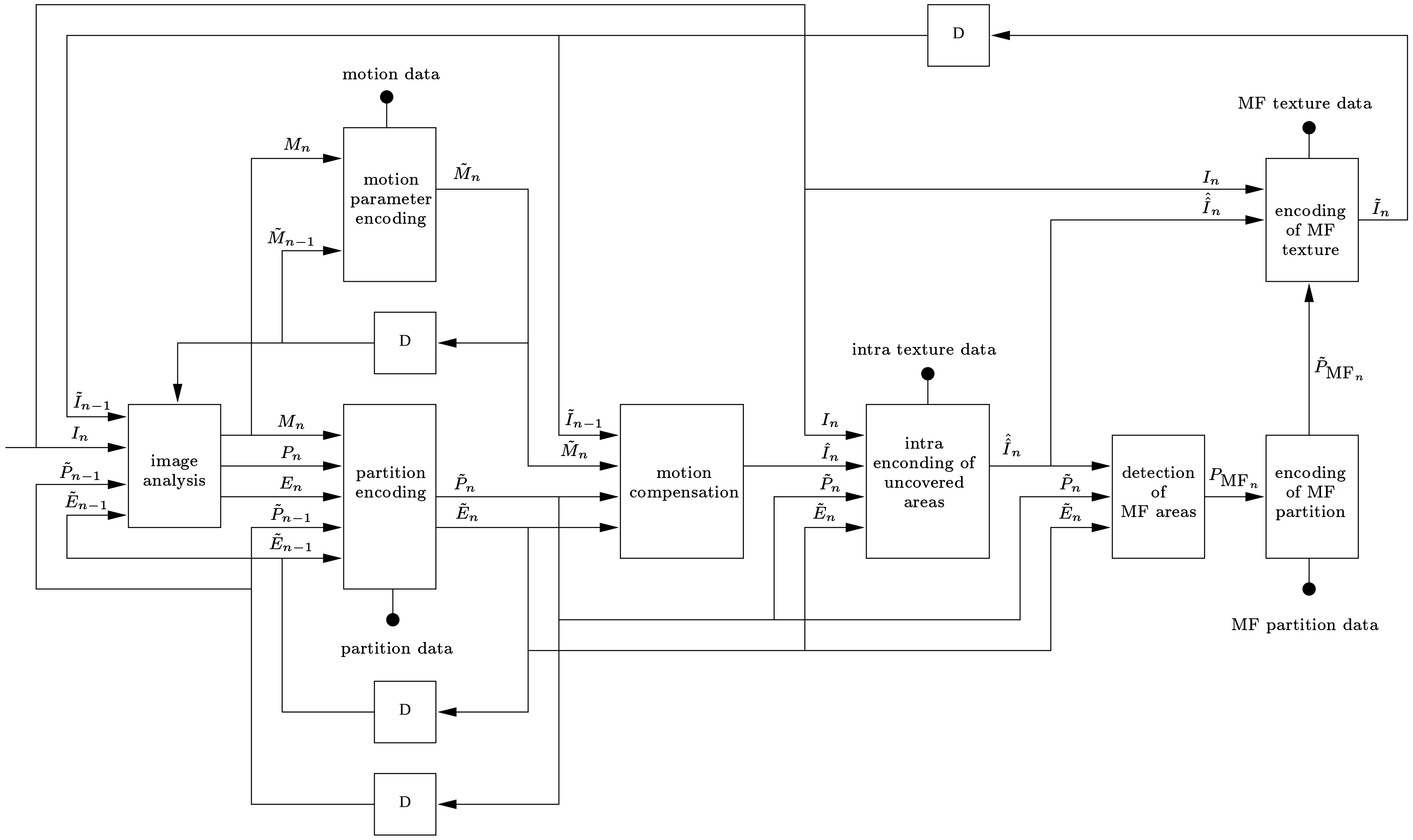


Figure 7.1: Proposed encoder architecture. See text for the meaning of the interface signals. n means current image and $n-1$ means the previous image. Blocks labeled D are (delay) memories. A tilde ˜ means an encoded/decoded (quantized) signal, while a hat ˆ means a compensated signal. The black circles • are output connections to a multiplexer/channel coder.

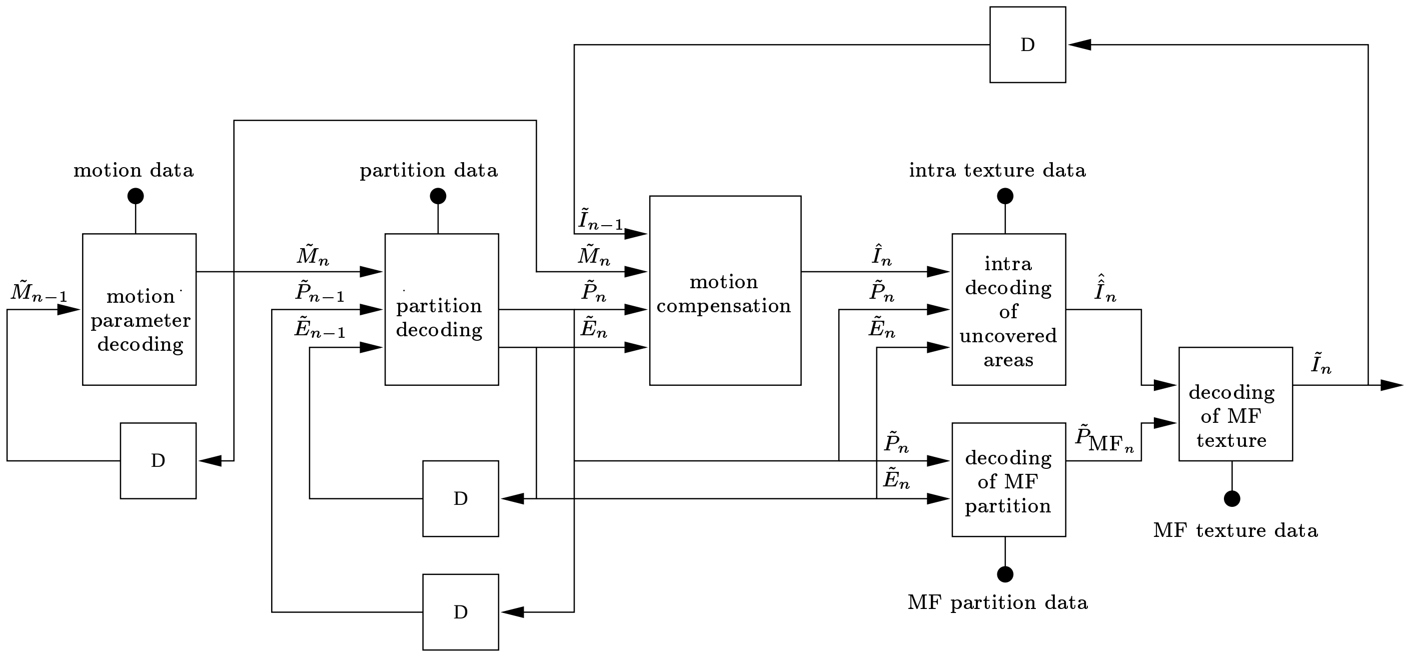


Figure 7.2: Proposed decoder architecture. See text for the meaning of the interface signals and Figure 7.1 for the notation.

Image analysis is required to segment the current image (using also the previous image, the previous partition information, and the previous motion parameters) into a set of connected regions, grouped into (not necessarily connected) objects: a partition. Often the detected objects will consist only of parts of the real objects in the scene, e.g., in the case of occlusion by other objects. These objects will mostly be related to objects present in previous images, though new objects are likely to appear from time to time. This partition is additionally required to contain information about uncovered regions. Uncovered regions should be labeled as belonging to one of the adjacent regions. For all practical purposes, new objects are uncovered areas that are not deemed to belong to any of the adjacent objects.

Objects should have coherent motion, i.e., all visible (non-uncovered) parts of an object should be represented by a single set of (backward) motion model parameters, also obtained by this block. The partition should desirably be coherent with the underlying texture. For that, spatial analysis may have to be used, as discussed in Chapter 4.

If there is camera movement in the scene, the image analysis block should estimate its parameters according to a given camera movement model, as discussed in Chapter 5.

Finally, the image analysis block is required to take into account the evolution of the objects in the scene. That is, segmentation should also be coherent in time. This is why the previous partition information and the previous motion parameters are fed back into the image analysis block.

Partition encoding

The "Partition encoding" block should encode the current image partition and the extra parameters as efficiently as possible. It seems reasonable to expect that considerable savings in bitrate may result from encoding partitions differentially using motion compensation. Notice, however, the following problems:

1. in order to motion compensate (project) objects from the previous image into the current image, forward motion must be available; and
2. the motion parameters of an object are usually encoded with respect to the object's shape and position, and the object's shape and position are predictively encoded using the object's motion.

Hence, apparently, problem 1 makes it necessary to invert motion estimation from backward to forward, which would create additional problems when compensating the inside (texture) of regions, viz. the possibility of gaps appearing, while problem 2 creates a chicken and egg dilemma.

However, if motion models such as affine are used, the problems are easy to solve. As to problem 1, affine motion is almost always invertible, so it is simple to obtain forward motion from the backward motion parameters. Regarding problem 2, one can always encode the affine motion parameters with regard to the object shape and position in the *previous* image.

The graph of occlusions from the previous partition image is used here to decide which object takes precedence over which in case several compensated objects overlap.

As to the graph of occlusions itself, its topology often does not need to be encoded, since it is implicit in the decoded partition. Hence, both coder and decoder are able to build the same undirected graph, given the current image partition. By making the graph directed, occlusions may be represented. If an arc is undirected, the regions do not occlude each other. If it is directed, the direction specifies which region occludes which. Direction information must be encoded for each arc in the implicit undirected graph. If mutual occlusions occur, then multigraphs may have to be used, each parallel arc specifying a segment of boundary between two regions with given occlusion characteristics. This extra information may also be built on top of the implicit undirected (simple) graph.

The prediction error of the motion compensated partition can be encoded using the techniques discussed in Chapter 6.

Finally, some means of encoding the temporal evolution of objects, in terms of split and merged objects and of disappeared and newly appeared objects, must be devised.

Motion parameters encoding

Motion model parameters are a very sensitive type of information. Motion encoding errors can have considerable repercussion in the quality of motion compensation and hence on the quality of the predicted image obtained by motion compensation. Thus, motion parameters should probably be encoded losslessly or at least with high accuracy. Some scheme for differentially encoding the motion parameters for each object will probably be of use.

Often the best way of encoding the parameters of some model (of texture or of motion) in a given region is to send samples of the region values which are sufficient for the decoder to estimate accurately the original model parameters. Often the location of these samples can be inferred by the decoder with the available information, which may or may not include the region shape. The advantage of this scheme over schemes where model parameters are encoded directly stems from the fact that, if the model is well behaved (and typical models are), the effects of quantization on the decoded region are more clearly understandable if this operation is performed on the function samples than if it is performed directly on the model parameters. A typical example is affine motion models, which are very useful for representing region motion. For 2D images, the parameters of affine motion of a region can be represented by three non-collinear sample motion vectors.[1] The errors of the motion vectors within the triangle limited by the three samples will always be smaller than the quantization errors of the quantized sample motion vectors, for each motion vector component. A reasonable choice for the sample positions seems to be the vertices of a triangulation of the object (e.g., the enveloping triangle with the smallest area). Since this triangulation may be done both at encoder and decoder, one needs only to encode the values of the sample motion vectors of all regions, maybe in raster order, using prediction as in H.263 [63].

[1]If the region itself consists of collinear pixels, then this restriction may be relaxed. Two non-coincident samples are sufficient. Similarly for the trivial one-pixel region.

As already mentioned in the previous section, motion model parameters will be encoded before encoding the current partition, so that the partition of the previous image can be motion compensated. The above referred triangulation is thus based on the previous position and shape of the objects.

Motion compensation

Once the image partition is known and motion model parameters are available for each object, motion compensation consists simply in projecting the previous image into the current one according to the motion vector field recreated from the motion parameters. If references from the future are allowed, as in MPEG-2 and MPEG-4, projection can also be performed from the future, and thus also for new objects. This requires, nevertheless, anchor intra images, where objects are encoded without any reference, or anchor images predicted only from the past. Since interpolation may be needed to obtain the corresponding pixel in the reference (decoded) image, progressive smoothing of the objects' texture may result. This may be solved by building an object store, and by compositing the motions between successive images in order to fetch the texture from this store. This method was originally proposed in COST211ter's SIMOC1 [39], and is similar to the one used for sprite update in MPEG-4.

Intra encoding of uncovered texture

Uncovered areas are those with no correspondence in the previous (decoded) image. They may correspond to new objects or to uncovered areas of existing objects. These areas have to be coded in intra mode. There are several possibilities, from shape adaptive DCT and related methods [184, 84, 54, 88, 87] to VQ (Vector Quantization) [59]. However, other methods may be attempted, such as VQ using the part of the object that was *not* uncovered (if it exists) as a codebook, in the case of textured areas, or some kind of extrapolation plus prediction errors, in the case of smooth areas.

For new objects, it might be useful to use texture based segmentation coding schemes, since it would create a smooth evolution path towards hierarchical source models. In that case, the spatial analysis tools proposed in Chapter 4 may prove useful, as well as the partition coding tools discussed in Chapter 6.

Model Failure blocks

No matter how complex the motion models, there will always be some objects (or parts thereof) undergoing movements which do not fit them. In such cases either these objects are split into smaller, and thus easier to model, objects, which may be extremely inconvenient from the point of view of content-based functionalities, or motion compensation errors, i.e., MFs [40], will have to be allowed.

The "Detection of MF areas" block uses criteria based on motion compensation errors. This block should take into account the current image partition in order to produce a coherent MF partition. This MF partition is then encoded by the "Encoding of MF partition" block.

Finally, the "Encoding of MF texture" block encodes the texture of the MF areas using shape-adaptive techniques.

7.1.3 Conclusions

A new second-generation codec architecture has been proposed which may use some of the tools developed in the previous chapters. The next section discusses how this architecture and the tools proposed may be improved.

7.2 Suggestions for further work

7.2.1 Codec architecture

Regarding the proposed codec architecture, the first issue that remained for future work was its full implementation making use of the analysis and coding tools proposed in the previous chapters.

Source model

The selected source model may be improved in several ways. One of the possibilities is to allow the objects to have memory [167]. That is, objects can be successively filled from the partial information available at each image. This would correspond to the layered approach of Adelson et al. [3]. In his model, each object is represented by a mask, specifying the known shape of the object, and by the object's texture. One difficulty with this scheme is the representation of mutually overlapping objects, though this might be solved by adding discriminating depth values to the object's mask (at the expense of some extra bitrate).

Other approaches include more realistic 3D models of the scene, though experience has shown that this task is tremendous, except when a priori knowledge about the scene is available (e.g., face objects in MPEG-4).

Finally, since facilitating the manipulation of video content is one of the aims of modern codecs, it seems that the definition of objects as zones with coherent motion may not provide enough content access discrimination: for instance, a static background will be considered as a single object, though the user might be interested in manipulating individual objects (such as a painting on a wall). This may lead to the definition of a hierarchy of objects: at the lowest level objects are defined by homogeneous texture and at the highest level by homogeneous motion. This has been the step taken by Sesame [30].

Image analysis

Several techniques can be used for image analysis:

Shape from texture (and motion from shape)
: Firstly segmentation is carried out using texture information only (though with provisions to keep temporal coherency), as in Chapter 4. Then motion is estimated for each region. Finally objects are built from regions with motion describable by a single set of motion parameters. Hence, time analysis is performed after a first step of spatial analysis.

Shape from motion
: Optical flow (2D projection of the real 3D motion) is estimated first. Then the obtained vector field is segmented (maybe using texture for accurate boundary localization). Finally, the motion model parameters corresponding to each region are computed. Notice that some optical flow algorithms detect motion boundaries that can be used to help segmentation. In this case time analysis is performed first. In order to obtain a hierarchy of objects, the partition may then be refined based solely on spatial analysis. Hence, spatial analysis is performed only after a first step in time analysis.

Simultaneous shape and motion
: Simultaneous motion estimation and image segmentation are attempted. These techniques sometimes correspond to iterative versions of the **shape from motion** and **motion from shape** approaches. In this case, it is also possible to use texture information for accurate boundary localization.

The most promising of these approaches is the last one, simultaneous analysis of shape and motion, since it takes into account a basic contradiction in image analysis: a good motion estimation requires accurate segmentation of the image into objects with different motion and, simultaneously, an accurate segmentation requires a good motion estimation.

Apart from the desired coherency of motion segmentation results with the underlying texture boundaries, another important point for investigation is the maintenance of temporal coherency along successive image partitions. Some interesting ideas can be found in [153], which have already been applied to the spatial analysis tools proposed in Chapter 4.

Motion models

As to the motion models used, experience has shown that translational motion is clearly insufficient, since it cannot describe but the simplest types of projected 3D motion. However, affine motion models, though certainly not enough to model the perspective projection of the motion of all rigid 3D surfaces, have shown to be reasonably good in most situations and relatively simple to manage [41] (6 parameters per object, instead of 2 for translational models).

Content access effort

Though the proposed codec architecture and source model seem more appropriate for minimizing the content access effort than classical video coding algorithms, the improvement must still be quantified: standard ways of measuring the content access effort must be devised.

Other issues requiring attention are how to manipulate objects whose description is spread along the encoded video stream and how to periodically refresh object descriptions.

7.2.2 Graph theoretic foundations for image analysis

An issue which remained for future work was the study of the theory of cell complexes and the reformulation of the SST foundations of image analysis in their framework.

An interesting question which also remained for future work is the checking of whether any spanning k-tree such that each of its connected components is a SST of the corresponding subgraph corresponds to a SSSkT of the graph for some set of seeds. If this is true, it would be interesting to relate the result to the skeletons in mathematical morphology. The assertion is obviously true if each seed can consist of more than one vertex: simply select as a seed all vertices of the corresponding connected component. In this case one may ask what is the minimal number of vertex seeds with the required property.

7.2.3 Spatial analysis

Region- and contour-oriented segmentation algorithms

The extension of all the notions to 3D, whose treatment in this thesis is only partial, and the study of segmentation techniques, also with a graph theoretic framework, but now using the concepts of flow on graphs [200], remained for future work.

Another issue requiring further work is the study of multiple solutions to the SSF or SST problems and their impact in the multiple solutions of the region growing, region merging, and contour closing segmentation algorithms.

Finally, the development of faster implementations of the globalized segmentation algorithms using SST-based concepts also remained for future work.

A new knowledge-based segmentation algorithm

The classification of sequences can be improved. A refinement of Class 4 is possible by introducing information about the kind of movement in the background:

Class 4A
Uniformly moving background, i.e., the whole background suffers the same motion (ac-

cording to a given motion model) from one image to the next (e.g., "Foreman" has this kind of background movement). Usually the background motion is due to camera movements.

Class 4B

Non-uniformly moving background (e.g., "Carphone" has a non-uniformly moving background, since the landscape seen through the windows moves independently of the background).

Assuming a translation plus isometric scaling motion model, class 4A corresponds to sequences where movement in the background is due to camera operations such as pan, tilt and zoom (camera vibrations can be considered to consist of small pan and tilt movements). The segmentation of this class of sequences might be attempted using the same algorithms as for Class 3 if image stabilization is performed first (see Section 5.5). Image stabilization may also be useful in the case of class 4B sequences, since these scenes usually have a dominant motion in the background (e.g., in "Carphone" the vibration, if correctly estimated, would be canceled and the only remaining motion in the background would be due to the moving landscape seen through the window). The combination of image stabilization and knowledge-based segmentation has not been attempted, and remained as an issue for future work.

RSST segmentation algorithms

The major drawback of the described algorithms is that they do not deal well with textures. This drawback, however, does not seem to be related as much to the algorithms themselves, as to the region models used. Hence, a subject requiring further study is region models which can appropriately represent textured regions. As to the algorithms, the basic algorithm behind flat and affine RSST needs to be improved to avoid over adjustment for small regions. A possibility might be a more thorough integration of the split and merge phases of the algorithm.

A related subject requiring further study is that of split using models, instead of the simple dynamic range used in this work. Also, a formalization of the impact of split in memory and computational power required should be performed. Finally, a memory efficient version of the algorithms should be implemented so as to allow practical segmentation of larger images.

Supervised segmentation

An issue which remained for future work is the development of supervision algorithms which can substitute, at least partially, human intervention. Issues which also remain for future work are the optimization of the RSST algorithms with seeds (viz. the global error minimizing ones) and the test of the region growing algorithm (for finding the SSSSkT) with amortized linear time execution proposed in Section 4.3.3.

Time-coherent analysis

Three issues remained as the subject for future work. The first is the introduction of better region models (e.g., affine), in order to avoid the false contours, and of illumination models, to avoid the artificial separation of regions which semantically are one only (see the arm in Figure 4.24). The second is the introduction of motion compensation so as to improve the projection of past partitions into the future, which has already been done with success in the watershed algorithms, see [105]. Finally, the study of region models capable of modeling both the 2D textures and their motion from one image to the next, for instance according to an affine model of motion.

7.2.4 Time analysis

Several issues remained for further work:

1. introduction of sub-pixel accurate block matching, so as to improve estimation of small pan movements;
2. quantification of errors in block matching estimates, viz. the estimation of the covariance matrix of the motion vectors;
3. quantification of errors in camera movement estimates;
4. introduction of rotation around the lens axis as a possible camera movement;
5. improvement of interpolation of pixel values in the algorithm for image stabilization; and
6. integration of motion vector field smoothing with the Hough outlier detector.

7.2.5 Coding

A few issues remained for future work:

1. the extension of the partition tree to include a branch for line drawings or "contours that may be open" (which are not the dual of some partition); this is of interest since contour-based coding, or image reconstruction from edges [58, 19, 37, 43], with its long history, still seems to have a large potential in image coding;
2. the implementation of optimized chain coding using algorithms solving the Chinese postman problem; and
3. the extension of the taxonomy tree with a systematization of partition coding techniques, besides partition types and representations.

7.3 List of contributions

This section lists the contributions of the thesis according to the corresponding chapter.

7.3.1 Graph theoretic foundations for image analysis

In this case the results are new in the framework of image analysis.

1. A thorough discussion of seeded SST algorithms (viz. SSF, SSkT, SSSkT, and SSSSkT).
2. An asymptotically linear amortized time algorithm for obtaining multiple SSSSkTs, for different sets of seeds, of the same graph.
3. A discussion of the relation between the SSF and dual graphs, which plays an important role in proving that basic region merging and basic contour closing are one and the same algorithm, solving the same problem.

7.3.2 Spatial analysis

1. A proposal for hierarchizing the segmentation process.
2. A discussion of region- and contour-oriented algorithms using the common framework of SSTs and related concepts from graph theory.
3. A discussion, in the same framework, of the main differences and similarities between region merging, region growing, and contour closing, namely the duality between contour closing and region merging.
4. A description of the watershed algorithm as a SSSSkT problem.
5. An application of the asymptotically linear amortized time SSSSkT algorithm for obtaining multiple region growing segmentations of an image, e.g., in a supervised segmentation environment.
6. First ideas regarding globalization of information in the basic segmentation algorithms, which may lead to a more thorough theoretical foundation for segmentation in the future.
7. A knowledge-based mobile videotelephony segmentation algorithm, able to cope with vibration and camera movement.
8. Extensions of the RSST algorithms, namely the new RSST, the flat RSST with an added split phase, and the use of affine models in the affine RSST.
9. Extensions of the RSST so that seeds are used, and its use for supervised segmentation.
10. Use of the seed extensions of the RSST algorithms for time-recursive segmentation of moving images.

7.3.3 Time analysis

1. Two camera movement estimation algorithms based on block matching, using least squares estimation with removal of outliers (see also [4]), though both using motion vector smoothing as an intermediate step in order to improve estimation.
2. An image stabilization method making use of the estimated camera movement factors.

7.3.4 Coding

1. A camera movement compensation method for classical codecs (which is also applicable to more modern ones, such as those compliant with the forthcoming MPEG-4 standard).

2. A systematization of partition types and representations in the form of a taxonomy tree.
3. A suggestion for improving chain codes of mosaic partitions through the solution of the Chinese postman problem or related problems.
4. A new fast (approximate) closed cubic spline algorithm.

Appendix A

Test sequences

When we look at a television program, we see more than flickering dots: we see people.

Douglas R. Hofstadter

This appendix discusses briefly the formats under which digital video sequences are available, and then proceeds to describe the test sequences used in this thesis.

A.1 Video formats

Digital video sequences usually come in the format specified by the ITU-R Recommendation BT.601-2 [20] or a derivative thereof. That is, in the $Y'C_BC_R$ color space, with an interlaced sampling lattice, where the chroma signals are horizontally subsampled by a factor of 2 relative to the luma signal, and the chroma samples are co-sited with the even luma samples (assuming the first luma sample in each line is sample 0). This format is referred to as 4:2:2. The number of samples is 720 horizontally and 288 vertically (for each field) for the luma signal, for 625 line TV systems (European), and 720 and 240 for 525 line TV systems (American).

In digital video coding, however, two different formats are typically used: CIF (Common Intermediate Format) and QCIF (Quarter-CIF). Both have a 4:2:0 sampling, meaning that the chroma signals are also vertically subsampled by a factor of 2 relative to the luma signal, and both are progressive. The number of samples in CIF is 352 horizontally and 288 vertically for the luma signal. In QCIF it is 176 horizontally and 144 vertically for the luma signal. The image rate in both cases is 30 Hz. Both formats were chosen as intermediates between the European SIF (Standard Interchange Format) format, with 25 Hz image rate and 352 by 288 samples, and the American SIF format, with 30 Hz image rate and 352 by 240 samples (both result in the same total number of samples per second). It is an intermediate format because it requires time resampling to go from European SIF to CIF, and space resampling to go from American SIF to CIF. In practice, though, European SIF sequences are often used as if they were CIF. Hence, in the list of test sequences in Section A.2, the image rate is always indicated, if available. This inconsistency has very little impact on the performance of the algorithms presented in this thesis, though.

Another inconsistency has to do with the relative sample locations between luma and chroma samples. The definitions of CIF and QCIF in H.261 and H.263 [62, 63] specify that chroma samples are located in the middle of the corresponding four luma samples in each 2×2 block. However, MPEG-4 [77] specifies that chroma samples are located in the middle of the two left luma samples in each 2×2 block, which is compatible with the format specified by the ITU-R Recommendation BT.601-2 [20]. Hence, in the list of Section A.2, sequences which are official MPEG-4 test sequences are indicated explicitly. Those sequences have the ITU-R Recommendation BT.601-2 positioning of samples, while the other sequences are believed to have the H.26x positioning of samples. Again, this small inconsistency has very little impact on the performance of the algorithms presented in this thesis.

Some algorithms in the thesis, notably the segmentation ones in Chapter 4, make use of the $R'G'B'255$ color space, where all color components have the same number of pixels (no subsampling), and the samples of the three color components have the same locations. Conversion from the $Y'C_BC_R$ CIF and QCIF format to $R'G'B'$, with the same number of samples as the luma signal, has been performed in two steps. In the first step, the chroma signals have been upsampled by a factor of 2 horizontally and vertically through simple repetition (sample-and-hold). Then, the following color space transformation has been performed for each pixel:

$$\begin{aligned}
R'_{255} &= \big(596 * (Y' - 16) + 817 * (C_R - 128)\big)//9,\\
G'_{255} &= \big(596 * (Y' - 16) - 201 * (C_B - 128) - 416 * (C_R - 128)\big)//9, \text{ and}\\
B'_{255} &= \big(596 * (Y' - 16) + 1033 * (C_B - 128)\big)//9,
\end{aligned}$$

where $//9$ means rounded division by $2^9 = 512$. The calculations have thus been performed in integer arithmetic. The transformation was taken from [164], but an extra bit of precision was added.

A.1.1 Aspect ratios

Sequences in the ITU-R Recommendation BT.601-2 format are obtained through sampling of TV signals with a picture aspect ratio of 4/3. The CIF and QCIF sequences are typically obtained by subsampling sequences in the ITU-R Recommendation BT.601-2 format. Hence, the procedure for obtaining a CIF sequence from a ITU-R Recommendation BT.601-2 European sequence is:

1. Drop the second field in each frame. The result is a 25 Hz sequence with 288 lines of 720 luma pixels and 360 chroma pixels.

2. Subsample both the luma and chroma signals by a factor of two horizontally (the filters for the chroma signals are different according to the desired sample positioning: H.26x

or MPEG-4). Each result image will thus have 288 lines with 360 luma samples and 180 chroma samples, corresponding to an image area with 4/3 aspect ratio.

3. Subsample the chroma signal by a factor of two vertically. The resulting images thus have 288 lines of 360 luma pixels and 144 lines of 180 luma pixels.
4. For each luma line, drop the first and the last four pixels. Also drop the first and the last two pixels of chroma. The result will have the CIF spatial format, though with a 25 Hz image rate.
5. Resample the images in time to go from 25 Hz to 30 Hz.

The conversion to QCIF is usually performed from the CIF sequences and involves only subsampling.

Hence, it can be easily concluded that the aspect ratio of the CIF and QCIF pixels is $\frac{\frac{4}{360}}{\frac{3}{288}}$, which is 16/15. However, some authors mention a slightly different value of 128/117 (cf. 16/15 = 128/120), arguing that not all of the 720 samples, only 702, are viewable in the 4/3 screen [168].

A.2 Test sequences

In this thesis the only formats used are CIF and QCIF, though with the nuances mentioned in the previous section. Sequences may be qualified by the acronym of their format, such as CIF “Carphone” or QCIF “Foreman”. When they appear without any qualifier, the format should be understood to be CIF, unless it is clear from the context that the format is QCIF. The images in the sequences are numbered from zero, i.e., the first image is image 0. The sequences used are described below (the first images of each sequence are shown in Figures A.1 and A.2). The concept of class, defined in Chapter 4, is used in the descriptions:

“Carphone” (CIF and QCIF, 25 Hz, 382 images)
Videotelephony sequence on a mobile, car mounted device. The sequence exhibits camera vibration and background motion (the landscape seen through windows). Most of the time it is a class 4 sequence. [Shot by Siemens, Germany, for the CEC RACE MAVT project.]

“Claire” (CIF and QCIF, 25 Hz, 494 images)
Videotelephony sequence on a studio with a fixed and relatively uniform light background. The background, however, is quite noisy, making it hard for motion estimation. It is a typical class 1 sequence. [Shot by CNET, France.]

“Coastguard” (CIF and QCIF, 25 Hz, 300 images, MPEG-4)
Outdoors scene showing part of a river with two moving boats and water movement, and showing also the bank of the river.[1] It has several panning camera movements.

[1]Images 277 and following are corrupted, so “Coastguard” has really 277 images.

“Flower Garden” (CIF only, 25 Hz, 125 images)
Camera traveling movement over a scene with a sloping garden and a row of houses. In the first plane, though out of focus, the trunk of a tree.

“Foreman” (CIF and QCIF, 25 Hz, 300 images, MPEG-4)
Videotelephony sequence on a mobile, hand-held device. Small panning camera movements occur, together with some rotation of the camera around its lens axis. The final part is a large panning movement in which the speaker disappears from the image. In this last part the camera rotation movements are more prominent. Most of the time it is a class 4 sequence. [Shot by Siemens for the CEC RACE MAVT project.]

“Grandmother” (QCIF only, 25 Hz, 870 images)
Videotelephony sequence with a fixed background, containing parts of a sofa and leaves of a plant against an uniform wall.

“Miss America” (CIF and QCIF, 30 Hz, 150 images)
Videotelephony sequence on a studio with a fixed and uniform dark background.[2] It has poor contrast between the background and the speaker, notably because of the dark hair. It is a class 1 sequence.

“Mother and Daughter” (QCIF only, 25 Hz, 961 images)
Videotelephony sequence with a fixed background containing parts of a sofa and a picture against an uniform wall.

“Salesman” (CIF and QCIF, 30 Hz, 449 images)
Videotelephony sequence in a home or office environment with a fixed, highly non-uniform and structured background. The speaker sometimes nearly stops. It is a typical class 2 sequence.

“Stefan” (CIF and QCIF, 25 Hz, 300 images, MPEG-4)
Sports TV sequence with a tennis player on a rather uniform tennis field and with textured public in the seats. The sequence has strong panning movements and some zoom movements.

“Table Tennis” (CIF and QCIF, 25 Hz, 300 images, MPEG-4)
Television sequence of a table tennis game. It has two different shots with a clear separation. The first shot has a clean zoom movement approximately between images 20 and 107. The background is finely textured. [Shot by CCETT, France.]

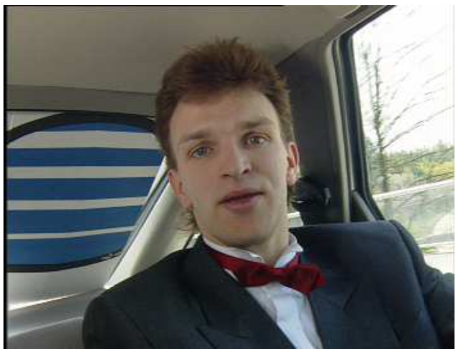
“Trevor” (CIF and QCIF, 25 Hz, 150 images)
Videoconference sequence on studio with fixed but non-uniform background.[3] It is divided in two shots, the first merging through an average image (image 59) to the second. The first shot is a vertically split two-view videoconference scene in a studio, with several persons in each view. It has 59 images (0 to 58). The second shot (Trevor, one may presume) is a typical head and shoulders scene with 90 images (60 to 149), which may be seen as videotelephonic. The second shot is class 2. In the text, only the second shot is used. [Shot by BTRL, UK.]

[2]The CIF version of “Miss America” has really 360 pixels per line.
[3]The CIF version of “Trevor” is available only as part of the “VTPH” sequence, see below.

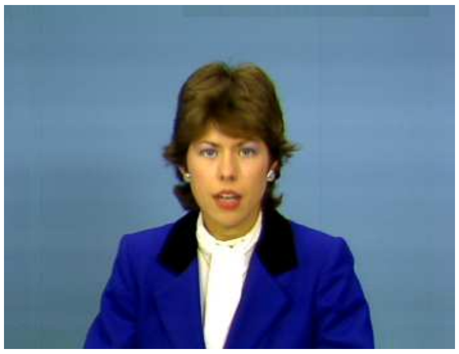
Additionally, there is a sequence called "VTPH" (created by CSELT, Italy) which was edited from "Claire", "Miss America", and "Trevor". It has 160 images (0 to 159), the first 80 coming from the beginning of "Claire" (with a temporal subsampling of 3, so that only every third image are extracted, from 0 to 3×79), the next 51 (80 to 130) coming from the beginning of "Miss America" (also with a temporal subsampling of 3),[4] and the last 29 (131 to 159) coming from "Trevor", starting at image 60 (also with downsampling of 3, from 60 to $60 + 3 \times 28$). The sequence is thus a concoction which simulates a hypothetical videotelephony conference talk shot at 10 Hz.[5]

[4]It should be noticed that the "Miss America" part of the sequence suffers from two problems, which in no way invalidate the results of the simulations. Firstly, the images are missing 12 lines at its top (corresponding to background), and have the same number of lines of noise in the bottom. Secondly, image 100 of the "VTPH" sequence is a repetition of image 99, i.e., images 19 and 20 of the "Miss America" shot are equal (both correspond to image 57 in the original "Miss America" sequence).

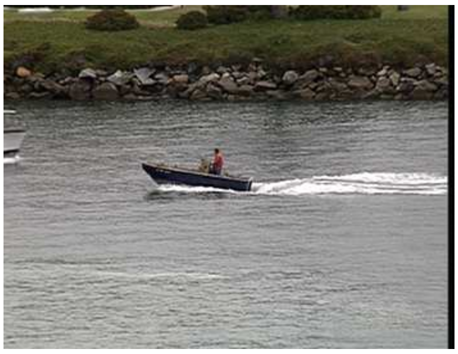
[5]Rigorously speaking, the "Claire" and "Trevor" parts are 25/3 Hz.



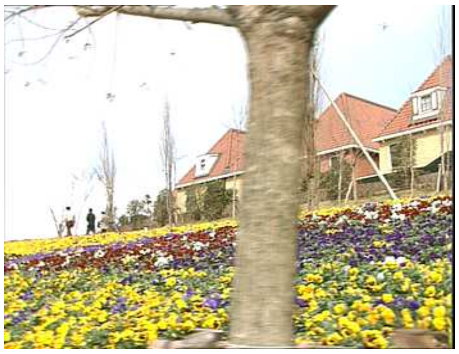
(a) “Carphone” image 0.



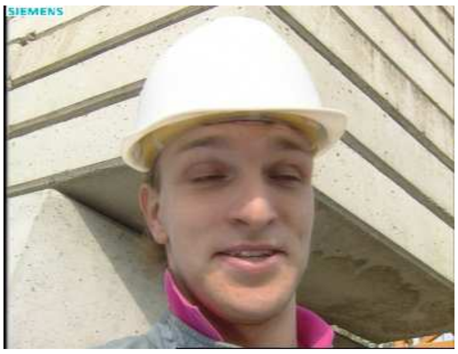
(b) “Claire” image 0 (image 0 of “VTPH”).



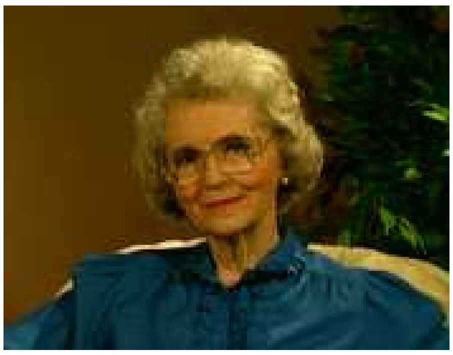
(c) “Coastguard” image 0.



(d) “Flower Garden” image 0.



(e) “Foreman” image 0.



(f) “Grandmother” image 0.

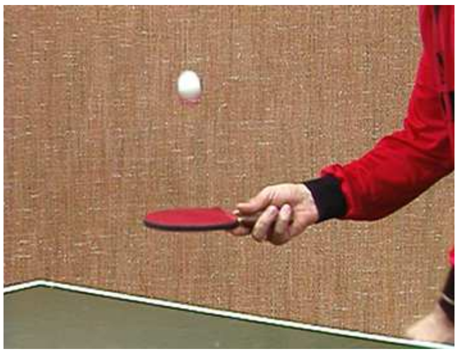
Figure A.1: The test sequences.

(a) “Miss America” image 0 (image 80 of “VTPH”).

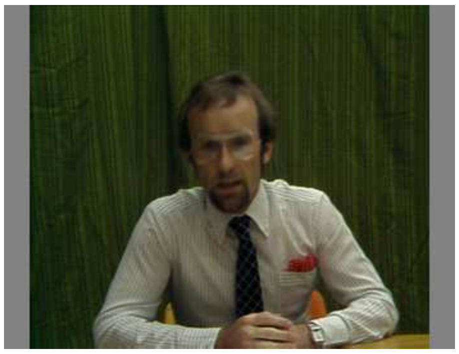
(b) “Mother and Daughter” image 0.

(c) “Salesman” image 0.

(d) “Stefan” image 0.



(e) “Table Tennis” image 0.



(f) “Trevor” image 60 (image 131 of “VTPH”).

Figure A.2: The test sequences (continued).

Appendix B

The Frames video coding library

It has been often said that a person does not really understand something until after teaching it to someone else. Actually a person does not really *understand something until after teaching it to a* computer ...

Donald E. Knuth

The Frames library of ANSI-C functions was developed to simplify the task of writing video coding and image processing algorithms:

1. Frames is free, it is under the GPL (GNU General Public Licence) of the FSF (Free Software Foundation);
2. Frames's current version is 3.2;
3. the Frames documentation can be found in [118], though the document reflects version 2 of the library; and
4. Frames will soon be put into an FTP (File Transfer Protocol) site; for the time being copies of Frames can be asked from the author by sending email to `Manuel.Sequeira@iscte.pt`.

Frames was started in July 1992, and has been evolving ever since. It had small but significant contributions made by Carlos Arede, Diogo Dias Cortez Ferreira, Paulo Correia, and, specially, Paulo Jorge Lourenço Nunes. The bug reports of many users of the library were also a great help.

The next sections briefly describe the Frames library.

B.1 Library modules

This library is composed of several modules, each of which defines data and functions with related purposes. A list of the corresponding header (interface) files and a brief description of each is given below (the prefix of functions, macros, types and global variables is shown between braces). The modules are always described through their header files.

1. Basic header files:

   `frames.h` – includes all headers from the library, thus giving access to all its data structures, macros, and functions.

   `types.h` – defines several basic types and macros; all other header files include this one.

   `errors.h` – {`ERR`} implements consistent error processing across the library.

   `io.h` – {`IO`} basic input/output functions (partially substitutes `stdio.h`).

   `matrix.h` – {`M`} defines (3D) matrix data types and a wealth of functions which operate with them.

   `mem.h` – {`MEM`} memory allocation module.

   `sequence.h` – {`S`} implements data structures and functions for dealing with video sequence files.

2. Main header files:

   `arguments.h`
   – {`ARG`} tools for command line argument processing.

   `bitstring.h`
   – {`BS`} module dealing with strings containing only characters `'0'` and `'1'`, which are interpreted as numbers in binary representation.

   `buffer.h`
   – {`B`} general buffer tools (currently defines a bit buffer file designed for bit-level reading and writing).

   `chc.h`
   – {`CHC`} tools for Cooperative Hierarchical Computation analysis of images [16].

   `colorspace.h`
   – {`CS`} tools for color space specification and conversion.

   `contour.h`
   – {`C`} tools for contours and partition matrices.

   `dct.h`
   – {`DCT`} functions for calculating the DCT.

   `draw.h`
   – {`D`} functions for drawing on image matrices.

   `filters.h`
   – {`F`} discrete 2D or 3D image filters and operators.

`fstring.h`
: – {`FS`} module defining a special type of character string, compatible with all ANSI-C string functions, which can be made to grow automatically as needed when functions from this package are used.

`getoptions.h`
: – {`GO`} tools for processing command line arguments. It performs a similar function to **`arguments.h`**, though it is simpler and easier to use.

`graph.h`
: – {`G`} tools for processing simple graphs.

`heap.h`
: – {`HP`} implementation of heaps, i.e., efficient hierarchical (or priority) queues.

`list.h`
: – {`L`} implementation of simple lists.

`machine.h`
: – machine dependencies file (generated during installation).

`mmorph.h`
: – {`MM`} mathematical morphology operators (but no watersheds...).

`motion.h`
: – {`MD`} tools for motion detection and estimation.

`options.h`
: – {`OPT`} functions to use together with **`arguments.h`** for command line option processing.

`parse.h`
: – {`P`} a simple parser of option files.

`qsort.h`
: – fast macros for sorting vectors (faster than using the ANSI-C library function `qsort()`).

`random.h`
: – {`RAND`} pseudo-random number generators.

`select.h`
: – fast macros for selecting the nth largest element of a vector.

`spiral.h`
: – {`SP`} functions dealing with spirals and distances in lattices.

`spline.h`
: – {`SPL`} functions for calculating the coefficients of splines.

`splitmerge.h`
: – {`SM`} framework for segmentation algorithms with a split phase and three merge phases (see [33]).

`string.h`
: – functions working on strings which complement the ANSI-C libraries.

`vlc.h`
: – {`VLC`} tools for VLCs.

A brief description of the most important modules (header files) is given in the sections below.

B.1.1 `types.h`

Digital images have usually well defined sizes, in bits, for storing each color component of the pixels. Being so, it is very important for an image processing library to define appropriate types with fixed, machine independent sizes. It is the main purpose of this module. It defines integer types with 8, 16, and 32 bits, signed and unsigned. If the machine does not support such a set of integers, the library will not compile. It also defines a boolean type which is missing in C.

For reasons of consistency, this module also defines floating point types, though with machine dependent sizes and precisions. Several general purpose macros are also defined in this module.

B.1.2 `errors.h`

Error conditions are dealt with in a consistent way across this library. Usually functions where a fatal error occurs return an error indication (in the form of a special return value) and store information about the error in variables internal to the `errors.h` module, the so-called error flags. The user may then check for errors and proceed appropriately, e.g. by aborting execution and printing an error message. If macro `DEBUG` is defined in this module at compile time, then error messages are printed immediately as they occur. The same behavior can be obtained using the one of the module functions. In any of these cases, the program will be said to be in debug state.

There are three classes of events: errors, warning and diagnostic. Non fatal events (i.e., warnings and diagnostics), do not set the error flags. Messages are printed only if the program is in debug state for the corresponding class of events, otherwise nothing is done. There are functions for changing the debug state of each of the event classes. It is also possible to clear error conditions and to ask for a string describing the current error.

B.1.3 `io.h`

This module defines types and functions which deal with basic input/output. It contains basically substitutes for the ANSI-C `stdio.h` functions and types. As with some of the functions of `mem.h`, this was done to provide consistent error checking across the library. It also contains provision to attach a debug output stream to any given output stream. This allows printing commands to print to the two streams. This may be used to send to the terminal all the text that is also printed in a given file.

B.1.4 `mem.h`

This module provides the same functionality as the memory management functions of ANSI-C, though with error processing consistent with the rest of the package (see Section B.1.2). It also adds a few functions simplifying the task of dealing with 2D and 3D dynamic arrays of any type.

B.1.5 `matrix.h`

The `matrix` module defines types and functions dealing with matrices (3D arrays of elements of the basic types). Even though all matrices are really 3D, and hence organized into planes, lines, and columns, the user may use them as 2D matrices almost transparently. Matrices can be either temporary or permanent. The validity of permanent matrices is not affected by any functions in this module except for the memory freeing functions. Temporary matrices are used to store intermediate results of matrix operations and are destroyed immediately after being used.

For functions which require a matrix to store the operation result, a null return matrix passed as an argument will force the creation of a temporary matrix. Temporary matrices are freed automatically when passed as operands (and not as result matrices) of matrix functions (except for output functions and except for self operating functions). Temporary matrices can be set permanent and vice versa.

Aside from permanent or temporary, matrices can also be categorized as:

1. sub-matrices vs. first-hand,
2. contiguous vs. non-contiguous,
3. static vs. dynamic,
4. restricted vs. full.

Sub-matrices are just like regular matrices except that the data they refer to belongs to another matrix. Sub-matrices can be used to refer to part of matrix (e.g., only even lines) without worrying about indexing issues. Changing the original matrix will change correspondingly the sub-matrix (since their data is the same), and vice versa.

Contiguous matrices have their planes and lines stored one after another in memory without gaps. This means their data can be accessed as a large vector having the same number of elements as the matrix itself. Contiguousness can considerably increase the speed of some matrix functions. Usually sub-matrices are also non-contiguous.

Regular matrices are dynamic, and are created by functions of this module. However, some functions allow you to "fool" the library by masking an existing matrix so that it looks like a library matrix. These matrices are named static because their data is not freed by functions of the Frames package.

Also, each matrix can be in restricted mode. In this mode, the number of planes, lines, and columns of the matrix are set as if the matrix were smaller then its real size. This allows functions to operate on a small window of the real matrix without having to use special functions or worrying about indexing issues. It also allows the remaining elements of the matrix to be accessed using otherwise invalid indices. Restricting is reentrant and reversible: a matrix may be restricted any number of times and then unrestricted successively.

This module contains a large number of functions operating on matrices. Special versions of the functions are also usually available. Function with names ending in `S` store the result in the first operand matrix, and thus do not have a result argument. Functions with names ending in `Sc` use as second operand a scalar value instead of a matrix. Functions with names ending in `ScS` use as second operand a scalar value instead of a matrix and store the result in the first operand. Functions with names ending in `P` work only on a 3D sub-matrix (for 2D matrices the suffix `R` may be used). Of course, combinations of `P` or `R` with `S` and `Sc` are possible.

B.1.6 `sequence.h`

This module defines functions and types which simplify reading and writing the files corresponding to sequences of images. At present only the IST sequence format is supported (see [118] for a description of the format). The reading and writing functions are capable of automatic color space conversion. Support for pixel aspect ratios, different bits per pixel formats, etc. is provided. Images are read and written using matrices (see previous section). Use of 3D matrices allows several images to be read or written at the same time.

B.1.7 `contour.h`

This module deals with contours and partitions. It contains functions dealing with partition matrices, contour matrices, contours, etc. Functions for calculating the connected components of label images are also provided.

B.1.8 `dct.h`

This module contains functions for calculating the DCT and its inverse. The functions were built specially for 2D DCTs over rectangular regions. The calculations are performed in integer arithmetic with a precision specified by the user. Provision for checking whether a given precision complies with the IEEE (The Institute of Electrical and Electronics Engineers, INC.) standard 1180-1990 is also included. Compliance with that standard is required by most video coding standards.

B.1.9 `filters.h`

This module contains functions that filter matrix images in several ways. Essentially two types of functions exist: those which operate on isolated pixels (scalar filters) and those which operate over windows or neighborhoods (windowed filters). In general the size of the windows or neighborhoods is encoded in the function name.

The fact that the value of a pixel depends on the values of a surrounding window poses some problems near the borders of the images. By default, the functions adapt to this situation by reducing the number of pixels available for calculation. When other solutions are used they are explicitly encoded in the function name by means of a postfix. Currently, the only postfix available is `M`, when the value of inexistent pixels beyond the image borders are obtained by reflections of the existing pixels (mirror effect).

Two other postfixes exist (unrelated to windowing effects near borders, though): `T` and `Tr`. The first is used when the filter result is thresholded so that the output is either 1 or 0 according to whether the result is larger than the threshold or not (the threshold is passed as the last argument of the function). `Tr` is used when the filtered result is calculated using truncation instead of rounding (of course, this only make a difference for integer matrices).

Yet another category of filters exists: those which interpret the images as being binary. For these functions a zero is a zero, anything different from zero is seen as a one. Instead of being prefixed by `F` these functions are prefixed by `Fb`.

With some exceptions, the functions in this module are very similar to the ones of the `matrix` module. When an input matrix in temporary, it is freed. When the output matrix does not exist, a temporary matrix is created.

B.1.10 `graph.h`

This module deals with simple graphs. It will soon be extended to deal also with pseudographs. It was implemented so that:

1. user data is easily associated with both vertices and arcs;
2. vertices and arcs have an associated label and weight;
3. when certain operations are done over a graph or one of its arcs or vertices, appropriate user functions (called hooks) are invoked;[1] and
4. even if it implies additional memory expenditure and redundancy, the data structure representing a graph is easily accessible in several ways (there is a list of all arcs, a list of all vertices, and each vertex contains a list of its own arcs).

Functions are available for inserting and removing vertices and arcs to and from a graph, for performing mergings and splits of vertices, for coloring a graph, etc.

B.1.11 `heap.h`

Often image processing tasks such as segmentation, edge detection, and motion estimation, require the minimization of cost functionals. Some of the optimization techniques which can be

[1]In a sense, this allows users to sub-class the graph data structure.

used to solve such a minimization problem require that one keeps track of the element of a set having the minimum value, where the set can vary along the optimization.

This module defines a data structure and functions which help to track very efficiently the minimum value in a changing set of elements, each with an associated value. The module is very efficient provided that:

1. after each change in the set, the number of eliminated elements is small compared with the cardinal number of the set; and
2. each set change should affect the values of a small number of set elements;

where a set change means the total changes required so that the next iteration of the algorithm can proceed, that is, all the changes until the new minimum value in the set is required.

Tracking of minima is done by associating a heap tree structure to the user data structure containing the elements from which the minimum is to be found.

The heap trees defined in this module are basically binary trees with N leaves, where N is the number of elements of the set from which the minimum is to be tracked. Each node, except the leaves, has two children (branches). The leaves store (as generic pointers) the user data associated to the elements of the set. All other non-leaf nodes, when the heap tree is arranged, store the smallest of their children's value. Hence, in an arranged heap tree, the root node always stores the smallest value of the set. More about heaps (of a slightly different type) can be found in [28, 165].

B.1.12 `motion.h`

This module deals with motion detection and estimation. The functions currently defined deal only with block matching motion estimation. Several functions are available, allowing for a number of variations of the basic block matching algorithm. Pixel masks and lists of valid motions can be provided to the functions. It is also possible to use short-circuited calculation which considerably decreases running time. When two or more motion vectors yield the same minimum error, the full search functions in this module are guaranteed to return the smallest of those motion vectors in terms of the usual Euclidean norm (but without pixel aspect ratio correction), except when a list of valid motion vectors is provided, in which case the first of the motion vectors in the list leading to the minimum error is selected. This allows n-step algorithms, or even pixel aspect ratio corrected Euclidean distances, to be used.

B.1.13 `splitmerge.h`

This module deals with RSST segmentation algorithms, despite its name. The user controls the merging and splitting criteria and schedule, so the module is fully configurable. It makes use of the `graph.h` and `heap.h` modules. The results of all RSST segmentation algorithms in Chapter 4 were obtained by software using this module. Full support for 3D segmentation is provided.

The user configures the segmentation by providing the data to segment, usually some images, and by providing points to hook functions to be called in particular places of the algorithm. Hence, the user can specify what happens when two regions are merged or split, when an adjacency arc is changed, whether to regions should be merged or split, which regions should be merged first, etc. The module also makes an accounting of region areas (or volumes) and border lengths (or areas) to which the user can refer whenever necessary.

B.2 An example of use

Since this appendix only contains a very brief description of the Frames library, an example of use follows which can make the capabilities of the library more clear.

An example of use of the Frames library:

```
/*
 * Program: rsst
 *
 * Description:
 *    Segmentation as specified in the Vlachos and Constantinides
 *    paper (1993), but with an initial split phase (which can be
 *    eliminated). See my PhD thesis for details.
 *
 * Authors:
 *    Manuel Menezes de Sequeira, IST.
 *
 * History:
 *    Author:            Date:               Notes:
 *    MMS                1998/8/28           first release
 *
 * This file is part of the frames library: a library of C functions
 * for video coding and image processing.
 *
 * Copyright (C) 1998 Manuel Menezes de Sequeira (IT, IST, ISCTE)
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library (see the file COPYING); if not,
 * write to the Free Software Foundation, Inc., 675 Mass Ave,
 * Cambridge, MA 02139, USA.
 */

#include <frames/splitmerge.h>
#include <frames/mem.h>
#include <frames/matrix.h>
#include <frames/sequence.h>
#include <frames/getoptions.h>
#include <frames/errors.h>

/* Global information about a segmentation: */
typedef struct {
    Matrixub R, G, B;           /* the image data (matrices of
                                   unsigned bytes). */
    ubyte maxDR;                /* maximum region dynamic range. */
    dword maxRegs;              /* maximum final number of
                                   regions. */
} segmentation;

/* Region data: */
typedef struct {
    dfloat avgR, avgG, avgB;    /* average of color components. */
    ubyte minR, minG, minB;     /* minimum of color components. */
    ubyte maxR, maxG, maxB;     /* maximum of color components. */
} region;

/* Border data (actually, set of borders to the same adjacent
   region): */
typedef struct {
    dfloat errContr;            /* contribution of border elimination
                                   (region merging) to global
                                   error. */
} border;

/* When a new region is created: */
static void *newRegion(SMregion *smreg, SMpartition *part)
{
    segmentation *seg = part->data; /* get segmentation
                                       information. */
    SMpppd p = *smreg->pppds;   /* to get rectangular region size. */
    region *reg;                /* pointer to new region. */

    reg = MEMalloc(sizeof(region)); /* create new region. */

    /* Calculate average of color components in region: */
    reg->avgR = MsumubP(seg->R, p.p, p.l, p.c, p.np, p.nl, p.nc) /
        smreg->size;
    reg->avgG = MsumubP(seg->G, p.p, p.l, p.c, p.np, p.nl, p.nc) /
        smreg->size;
    reg->avgB = MsumubP(seg->B, p.p, p.l, p.c, p.np, p.nl, p.nc) /
        smreg->size;

    /* Calculate dynamic range of color components in region: */
    reg->maxR = MsmaxubP(seg->R, p.p, p.l, p.c, p.np, p.nl, p.nc);
    reg->minR = MsminubP(seg->R, p.p, p.l, p.c, p.np, p.nl, p.nc);
    reg->maxG = MsmaxubP(seg->G, p.p, p.l, p.c, p.np, p.nl, p.nc);
    reg->minG = MsminubP(seg->G, p.p, p.l, p.c, p.np, p.nl, p.nc);
    reg->maxB = MsmaxubP(seg->B, p.p, p.l, p.c, p.np, p.nl, p.nc);
    reg->minB = MsminubP(seg->B, p.p, p.l, p.c, p.np, p.nl, p.nc);

    return reg;
}

/* When a region is freed: */
static boolean freeRegion(SMregion *smreg, SMpartition *part)
{
    MEMfree(smreg->data);
    return ok;
}

/* When a new border is created: */
static void *newBorder(SMadjacency *smadj,
                       SMregion *smreg1, SMregion *smreg2,
                       SMpartition *part)
{
    region *reg1 = smreg1->data; /* get first region data. */
    region *reg2 = smreg2->data; /* get second region data. */
    border *bord;                /* pointer to new adjacency. */

    bord = MEMalloc(sizeof(border)); /* create new border. */

    /* Calculate contribution to global error: */
    bord->errContr = (sqr(reg1->avgR - reg2->avgR) +
                      sqr(reg1->avgG - reg2->avgG) +
                      sqr(reg1->avgB - reg2->avgB)) *
        smreg1->size * smreg2->size / (smreg1->size + smreg2->size);

    return bord;
}

/* When a border is freed: */
static boolean freeBorder(SMadjacency *smadj, SMpartition *part)
{
    MEMfree(smadj->data);
    return ok;
}
```

```
/* When two regions are merged: */
static boolean mergeRegions(SMregion *smreg1, SMregion *smreg2,
                            SMadjacency *smadj,
                            boolean bysize, SMpartition *part)
{
    region *reg1 = smreg1->data; /* get first region data. */
    region *reg2 = smreg2->data; /* get second region data. */

    /* Calculate the color component averages for the merged
       regions and store them in the first region (reg1), which is
       the one remaining after the merge (reg2 is freed
       eventually): */
    reg1->avgR = (smreg1->size * reg1->avgR +
                  smreg2->size * reg2->avgR) /
                 (smreg1->size + smreg2->size);
    reg1->avgG = (smreg1->size * reg1->avgG +
                  smreg2->size * reg2->avgG) /
                 (smreg1->size + smreg2->size);
    reg1->avgB = (smreg1->size * reg1->avgB +
                  smreg2->size * reg2->avgB) /
                 (smreg1->size + smreg2->size);

    return ok;
}

/* When a border needs recalculation (one of the corresponding
   regions changed): */
static boolean recalculateBorder(SMadjacency *smadj,
                                 SMregion *smreg1, SMregion *smreg2,
                                 SMpartition *part)
{
    region *reg1 = smreg1->data; /* get first region data. */
    region *reg2 = smreg2->data; /* get second region data. */
    border *bord = smadj->data; /* get border data. */

    /* Calculate contribution to global error: */
    bord->errContr = (sqr(reg1->avgR - reg2->avgR) +
                      sqr(reg1->avgG - reg2->avgG) +
                      sqr(reg1->avgB - reg2->avgB)) *
        smreg1->size * smreg2->size / (smreg1->size + smreg2->size);

    return ok;
}

/* Decide whether to split a rectangular region: */
static boolean shouldSplit(SMregion *smreg, SMpartition *part)
{
    segmentation *seg = part->data; /* get segmentation
                                       information. */
    region *reg = smreg->data;  /* get region data. */

    /* Should split only if the dynamic range of some component
       exceeds the maximum allowable: */
    return ((reg->maxR - reg->minR > seg->maxDR) ||
            (reg->maxG - reg->minG > seg->maxDR) ||
            (reg->maxB - reg->minB > seg->maxDR));
}

/* Verify whether a border should be eliminated before another: */
static boolean eliminateBefore(void *smadj1v, void *smadj2v,
                               void *dummy)
{
    border *bord1 =
((SMadjacency*)smadj1v)->data; /* get first border data. */
    border *bord2 =
((SMadjacency*)smadj2v)->data; /* get second border data. */

    /* The first border should be eliminated before the second only
       if its contribution to the global error is smaller: */
    return bord1->errContr < bord2->errContr;
}

/* Decide whether two regions should be merged together: */
static boolean shouldMerge(SMadjacency *smadj, SMsize nregs,
                           SMmergePhase phase, SMpartition *part)
{
    /* Get segmentation information: */
    segmentation *seg = part->data;

    switch(phase) {
      case SMbyThresh:
      case SMbySize:
        /* The RSST segmentation algorithm of Vlachos and
           Constantinides has only one merge phase: */
        return false;
      case SMbyNumber:
        /* Merge if the number of regions is larger than the
           allowable maximum: */
        return nregs > seg->maxRegs;
    }
}

/* For segmenting an image: */
static
boolean segmentImage(Matrixdw labels, /* output partition. */
                     /* input image matrices: */
                     Matrixub R, Matrixub G, Matrixub B,
                     ubyte maxDR, /* maximum dynamic range. */
                     dword maxRegs, /* maximum number regions. */
                     dword blockSize) /* initial split block size. */
{
    /* The partition used by the split and merge module functions: */
    SMpartition *p;

    /* Pointer to new segmentation information. */
    segmentation *seg;

    /* Create new segmentation: */
    seg = MEMalloc(sizeof(segmentation));

    /* Store image data: */
    seg->R = R;
    seg->G = G;
    seg->B = B;

    /* Store information for split and merge criteria: */
    seg->maxDR = maxDR;
    seg->maxRegs = maxRegs;

    /* Initialize segmentation by performing split.  The hook
       functions are passed to customize the algorithm: */
    p = SMsplit(R->npla, R->nlin, R->ncol, blockSize,
                no,             /* borders are not 3D. */
                /* hook functions (user functions): */
                eliminateBefore, /* border ordering in merge by
                                    threshold and merge by number (in
                                    this case only the latter phase
                                    is used). */
                NULL,           /* border ordering in the merge by
                                   size phase (not used). */
                newRegion, newBorder, /* new regions and borders. */
                freeRegion, freeBorder, /* real frees. */
                freeRegion, freeBorder, /* simple removals. */
                recalculateBorder, /* recalculating a border. */
                mergeRegions,   /* region merging. */
                NULL,           /* border merging (not used). */
                NULL,           /* broken border (not used). */
                shouldSplit, shouldMerge, /* split and merge
                                             criteria. */
                NULL, NULL,     /* printing regions (not used). */
                NULL,           /* labeling regions (not used). */
                seg,            /* our segmentation information. */
                NULL);          /* report (not used). */

    /* Perform the merge steps: */
    while(SMmergeStep(p) != SMnoMerge)
        continue;

    /* Fill partition matrix with the attained partition: */
    SMlabel(labels,
            NULL,               /* no number of regions necessary*/
            p,
            1);                 /* same resolution as input image. */

    SMfree(p);                  /* free partition. */
    MEMfree(seg);               /* free segmentation information. */

    return ok;
}
```

```
/* Configuration variables for the segmentation: */
static ubyte maxDR;             /* maximum dynamic range. */
static dword maxRegs;           /* maximum number of regions. */
static dword blockSize;         /* initial split block size. */

/* Configuration variables for reading the sequence: */
static char *path;              /* where to read sequences from. */
static char *opath;             /* where to write files to. */
static dword first, period, total;/* images: first, period, total. */

/* For segmenting a sequence using the configuration above: */
void segmentSequence(char *partitionName, char *sequenceName)
{
    Sfile in;                   /* input image sequence. */
    IOfile out;                 /* output partition file. */
    dword n, last;              /* image counter, last plus one. */
    Matrixub R, G, B;           /* input image matrices. */
    Matrixdw labels;            /* output partition matrix. */

    /* Open input image sequence: */
    if((in = SopenFile(path, sequenceName)) == NULL)
        exit(EXIT_FAILURE);

    /* Create input image matrices: */
    if(!ScreateRGBBuffers(in, &R, &G, &B))
        exit(EXIT_FAILURE);

    /* Create output partition file: */
    if((out = IOcreatePath(opath, partitionName)) == NULL)
        exit(EXIT_FAILURE);

    /* Create output partition matrix: */
    if((labels = Mcreate2Ddw(R->nlin, R->ncol)) == NULL)
        exit(EXIT_FAILURE);

    last = first + (total == 0 ? in->pages : total) * period;

    /* Cycle images: */
    for(n = first;
        n < last && Sseek(in, n) && SreadRGB(in, R, G, B) == 1;
        n += period) {
        /* Segment current image (n): */
        segmentImage(labels, R, G, B, maxDR, maxRegs, blockSize);
        /* Write partition matrix to partition file: */
        Mwritedw(out, labels);
    }
    SfreeBuffers(R, G, B);      /* free input image matrices. */
    Mfreedw(labels);            /* free output partition matrix. */
    Sclose(in);                 /* close input image sequence. */
    IOclose(out);               /* close output partition file. */
}

/* Main program: */
int main(int argc, char **argv)
{
    GOoptions *options;         /* for reading command line. */

    /* Enumeration of command line options: */
    enum {DYNRANGE, REGIONS, BLOCKSIZE,
          HELP, ENV, DEF, PATH, OPATH, FIRST, PERIOD, TOTAL};

    /* Table of command line options definitions: */
    char *table[][GO_MAX_SYNONYM] = {
        /* Segmentation options: */
        {"-mdr", "--maxdynrange", "&12", /* default is 12. */
         "%argument is maximum dynamic range."},
        {"-mr", "--maxregions", "&10", /* default is 10. */
         "%argument is maximum number of regions."},
        {"-bs", "--blocksize", "&16", /* default 16x16. */
         "%argument is initial split block size."},

        /* Help options: */
        {"-h", "--help", "%shows help and finishes."},
        {"--env", "%show environment variables in help."},
        {"--def", "%show default values in help."},

        /* Sequence reading options: */
        {"-p", "--path", "FRAMESPATH",
         "%argument specifies where to search for sequences."},
        {"-op", "--outputpath", "FRAMESOPATH",
         "%argument specifies where to write created sequences."},
        {"-f", "--first", "&0",
          "%argument is the first image to read."},
        {"-i", "--increment", "&1",
         "%argument is the time period of read images."},
        {"-t", "--total", "&0",
         "%argument is the total of images to read (0 means all)."},
    };
    /* Number of command line options: */
    dword n = sizeof(table) / sizeof(table[0]);

    /* Help information: */
    GOhelp help = {
        "rsst",
        "segments a sequence according to the Vlachos and
            "Constantinides paper (1993), but with an initial split"
            "phase.",
        "1.0",
        "August 1998",
        "Manuel Menezes de Sequeira",
        "1998 IT/IST/ISCTE",
        "[options] partition sequence"
    };
    boolean showEnv = no, showDef = no;
    char *partition = NULL;
    char *sequence = NULL;
    dword totalArgs = 0;

    /* Initialize option interpreter: */
    if((options = GOnew(table, n, argv, help)) == NULL)
        return EXIT_SUCCESS;

    /* Interpret options and arguments: */
    forever {
        switch(GOnext(options)) {
            /* Segmentation options: */
          case DYNRANGE:
            maxDR = atoi(GOgetParameter(options));
            break;
          case REGIONS:
            maxRegs = atol(GOgetParameter(options));
            break;
          case BLOCKSIZE:
            blockSize = atol(GOgetParameter(options));
            break;

            /* Interpretation events: */
          case GOargument:
            totalArgs++;
            if(partition == NULL)
                partition = GOgetArgument(options);
            else
                sequence = GOgetArgument(options);
            break;
          case GOend:
            if(totalArgs == 2) {
                segmentSequence(partition, sequence);
                return EXIT_SUCCESS;
            }
            ERRprint("Must suply two arguments! (-h for help)");
            return EXIT_FAILURE;
          case GOinvalid:
            ERRprint("Invalid option! (-h for help)");
            return EXIT_FAILURE;

            /* Sequence options: */
          case HELP:
            GOshowHelp(options, showDef, showEnv);
            return EXIT_SUCCESS;
          case ENV:
            showEnv = yes;
            break;
          case DEF:
            showDef = yes;
            break;
          case PATH:
            path = GOgetParameter(options);
            break;
          case OPATH:
            opath = GOgetParameter(options);
            break;
          case FIRST:
            first = atol(GOgetParameter(options));
            break;
          case PERIOD:
            period = atol(GOgetParameter(options));
            break;
          case TOTAL:
            total = atol(GOgetParameter(options));
            break;
        }
    }
    return EXIT_SUCCESS;
}
```

Bibliography

[1] *Britannica Online*, chapter Sensory Reception: Human Vision: Structure and Function of the Eye: The Visual Process: The Work of the Retina. Encyclopædia Britannica, Inc., http://www.eb.com:180/cgi-bin/g?DocF=macro/5005/71/139.html, [Accessed 06 March 1998].

[2] Ad hoc Group on MPEG-4 Video VM Editing. MPEG-4 video verification model version 3.0. Document ISO/IEC JTC1/SC29/WG11 N1277, ISO, July 1996.

[3] E. H. Adelson, J. Y. A. Wang, and S. A. Niyogi. Layered representation for vision and video. In WIASIC'94 [198], page IP1.

[4] G. Adiv. Determining three-dimensional motion and structure from optical flow generated by several moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7(4):384–401, July 1985.

[5] J. K. Aggarwal and N. Nandhakumar. On the computation of motion from sequences of images – a review. *Proceedings of the IEEE*, 76(8):917–935, Aug. 1988.

[6] S. M. Ali and R. E. Burge. A new algorithm for extracting the interior of bounded regions based on chain coding. *Computer Vision, Graphics, and Image Processing*, 43:256–264, 1988.

[7] Y. Aloimonos and A. Rosenfeld. A response to "Ignorance, myopia, and naiveté in computer vision systems" by R. C. Jain and T. O. Binford. *CVGIP: Image Understanding*, 53(1):120–124, Jan. 1991.

[8] J. A. C. Bernsen. An objective and subjective evaluation of edge detection methods in images. *Philips Journal of Research*, 46(2-3):57–94, 1991.

[9] M. Bertero, T. A. Poggio, and V. Torre. Ill-posed problems in early vision. *Proceedings of the IEEE*, 76(8):869–889, Aug. 1988.

[10] M. J. Biggar and A. G. Constantinides. Thin line coding techniques. In *Proceedings of the International Conference on Digital Signal Processing (ICDSP'87)*, Florence, Sept. 1987.

[11] F. Bossen and T. Ebrahimi. A simple and efficient binary shape coding technique based on bitmap representation. Technical Description ISO/IEC JTC1/SC29/WG11 MPEG96/0964, EPFL, July 1996.

[12] N. Brady. Adaptive arithmetic encoding for shape coding. Technical Description ISO/IEC JTC1/SC29/WG11 MPEG96/0975, Teltec Ireland (Dublin City University), ACTS/MoMuSys, July 1996.

[13] P. Brigger, A. Gasull, C. Gu, F. Marqués, F. Meyer, and C. Oddou. Contour coding. CEC Deliverable R2053/UPC/GPS/DS/R/006/b1, EPFL, UPC, CMM, LEP, Dec. 1993.

[14] P. Brigger and M. Kunt. Morphological shape representation for very low bit-rate video coding. *Signal Processing: Image Communication*, 7(4–6):297–311, Nov. 1995.

[15] W. C. Brown. *Matrices and Vector Spaces.* Marcel Dekker, Inc., New York, 1991.

[16] P. J. Burt, T.-H. Hong, and A. Rosenfeld. Segmentation and estimation of image region properties through cooperative hierarchical computation. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(12):802–809, Dec. 1981.

[17] C. Cafforio and F. Rocca. Methods for measuring small displacements of television images. *IEEE Transactions on Information Theory*, IT-22(5):573–579, Sept. 1976.

[18] J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov. 1986.

[19] S. Carlsson. Sketch based coding of grey level images. *Signal Processing*, 15(1):57–83, July 1988.

[20] Encoding parameters of digital television for studios. Recommendation 601-2, CCIR, 1990.

[21] CCITT SG XV WP XV/4. Description of reference model 8 (RM8). Technical Report SIM89, COST211bis, Paris, May 1989.

[22] R. Chellappa and R. Bagdazian. Fourier coding of image boundaries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):102–105, Jan. 1984.

[23] W.-K. Chen. *Graph Theory and Its Engineering Applications*, volume 5 of *Avanced Series in Electrical and Computer Engineering.* World Scientific Publishing Co. Pte. Ltd., Singapore, 1997.

[24] Y.-S. Cho, S.-H. Lee, J.-S. Shin, and Y.-S. Seo. Results of core experiments on comparison of shape coding tools (S4). Technical Description ISO/IEC JTC1/SC29/WG11 MPEG96/0717, Samsung AIT, Mar. 1996.

[25] Y.-S. Cho, S.-H. Lee, J.-S. Shin, and Y.-S. Seo. Shape coding tool: Using polygonal approximation and reliable error residue sampling method. Technical Description ISO/IEC JTC1/SC29/WG11 MPEG96/0565, Samsung AIT, Jan. 1996.

[26] Spline implementation. Proposal COST211ter, Simulation Subgroup, SIM(93)56, CNET – France Telecom, 1993.

[27] S. Coren, L. M. Ward, and J. T. Enns. *Sensation and Perception.* Harcourt Brace College Publishers, 1994.

[28] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts, 1994 [1990].

[29] P. Correia and F. Pereira. The role of analysis in content-based video coding and indexing. *Signal Processing*, 66(2):125–142, 1998.

[30] I. Corset, L. Bouchard, S. Jeannin, P. Salembier, F. Marqués, M. Pardàs, R. Morros, F. Meyer, and B. Marcotegui. Technical description of SESAME (SEgmentation-based coding System Allowing Manipulation of objEcts). Technical Description ISO/IEC JTC1/SC29/WG11 MPEG95/0408, LEP, UPC, CMM, Nov. 1995.

[31] D. Cortez. Classificação e codificação de contornos. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, May 1995.

[32] D. Cortez, P. Nunes, M. M. de Sequeira, and F. Pereira. Image analysis towards very low bitrate video coding. In J. M. Sá da Costa and J. R. Caldas Pinto, editors, *Proceedings of the 6th Portuguese Conference on Pattern Recognition (RecPad'94)*, Lisboa, Mar. 1994.

[33] D. Cortez, P. Nunes, M. Menezes de Sequeira, and F. Pereira. Image segmentation towards new image representation methods. *Signal Processing: Image Communication*, 6(6):485–498, Feb. 1995.

[34] J. Crespo, J. Serra, and R. W. Schafer. Image segmentation using connected filters. In J. Serra and P. Salembier, editors, *Proceedings of the International Workshop on Mathematical Morphology and its Applications to Signal Processing*, pages 52–57, Barcelona, May 1993. UPC, EMP and EURASIP, UPC Publications Office.

[35] A. Cumani. A second-order differential operator for multispectral edge detection. In *Proceedings of the 5th International Conference on Image Analysis and Processing (ICIAP'89)*, pages 54–58, Positano, 1989.

[36] A. Cumani. Edge detection in multispectral images. *CVGIP: Graphical Models and Image Processing*, 53(1):40–51, Jan. 1991.

[37] A. Cumani, P. Grattoni, and A. Guiducci. An edge-based description of color images. *CVGIP: Graphical Models and Image Processing*, 53(4):313–323, July 1991.

[38] L. S. Davis. Two-dimensional shape representation. In T. Y. Young and K.-S. Fu, editors, *Handbook of Pattern Recognition and Image Processing*, chapter 10, pages 233–245. Academic Press, Inc., San Diego, California, 1986.

[39] SIMOC1. Contribute COST211ter, Simulation Subgroup, SIM(94)6, BOSCH, UNI-HAN, BT Labs, RWTH Aachen, CNET, NTR, KUL, PTT, LEP Philips, EPFL, Siemens, CSELT, DBP-Telekom, Bilkent University, UPC, HHI, Feb. 1994.

[40] SIMOC1. Contribute COST211ter, Simulation Subgroup, SIM(95)8, DCU, BOSCH, UNI-HAN, BT Labs, RWTH Aachen, CNET, NTR, KUL, PTT, LEP Philips, EPFL, Siemens, CSELT, DBP-Telekom, Bilkent University, UPC, HHI, and IST, Darmstadt, Feb. 1995.

[41] E. de Boer, N. Tang, and I. Lagendijk. Motion estimation: Translactional versus affine motion model. In WIASIC'94 [198], page C2.

[42] M. Eden and M. Kocher. On the performance of a contour coding algorithm in the context of image coding part I: Contour segment coding. *Signal Processing*, 8(4):381–386, July 1985.

[43] F. Eryurtlu, A. M. Kondoz, and B. G. Evans. Very low-bit-rate segmentation-based video coding using contour and texture prediction. *IEE Proceedings – Vision, Image, and Signal Processing*, 142(5):253–261, Oct. 1995.

[44] S. Even. *Graph Algorithms.* Pitman Publishing Limited, London, 1979.

[45] Standardization of Group 3 facsimile apparatus for document transmission. Recommendation T.4, CCITT, 1980.

[46] Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus. Recommendation T.6, CCITT, 1984.

[47] M. M. Fleck. Some defects in finite-difference edge finders. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(3):337–345, Mar. 1992.

[48] G. D. Forney, Jr. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, Mar. 1973.

[49] H. Freeman. On the encoding of arbitrary geometric configurations. *IRE Transactions on Electronic Computers*, 10:260–268, June 1961.

[50] H. Freeman. Computer processing of line-drawing images. *Computing Surveys*, 6(1):57–97, Mar. 1974.

[51] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP- Completeness.* W. H. Freeman and Company, New York, 1979.

[52] A. Gasull, F. Marqués, and J. A. García. Lossy image contour coding with multiple grid chain code. In WIASIC'94 [198], page B4.

[53] P. Gerken, M. Wollborn, and S. Schultz. Polygon/spline approximation of arbitrary image region shapes as proposal for MPEG-4 tool evaluation – technical description. Technical Description ISO/IEC JTC1/SC29/WG11 MPEG95/0360, RACE/MAVT, University of Hannover, Robert Bosch GmbH, and Deutsche Telekom AG, Nov. 1995.

[54] M. Gilge, T. Engelhardt, and R. Mehlan. Coding of arbitrarily shaped image segments based on a generalized orthogonal transform. *Signal Processing: Image Communication*, 1(2):153–180, Oct. 1989.

[55] R. C. Gonzalez and P. Wintz. *Digital Image Processing.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1987.

[56] R. C. Gonzalez and R. E. Woods. *Digital Image Processing.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1993.

[57] N. R. E. Gracias. Application of robust estimation to computer vision: Video mosaics and 3-d reconstruction. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, Dec. 1997.

[58] D. N. Graham. Image transmission by two-dimensional contour coding. *Proceedings of the IEEE*, 55(3):336–346, Mar. 1967.

[59] R. M. Gray. Vector quantization. *IEEE ASSP Magazine*, 1:4–29, Apr. 1984.

[60] W. E. L. Grimson and E. C. Hildreth. Comments on "Digital step edges from zero crossings of second directional derivatives". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7(1):121–127, Jan. 1985.

[61] C. Gu and M. Kunt. Contour simplification and motion compensated coding. *Signal Processing: Image Communication*, 7(4–6):279–296, Nov. 1995.

[62] Draft revision of recommendation H.261: Video codec for audiovisual services at p $\times$ 64 kbits/s, CCITT study group XV, TD 35, 1989. *Signal Processing: Image Communication*, 2(2):221–239, Aug. 1990.

[63] Video coding for low bitrate communication. Draft Recommendation H.263, ITU-T, Dec. 1995.

[64] A. Habibi. Hybrid coding of pictorial data. *IEEE Transactions on Communications*, COM-22(5):614–624, May 1974.

[65] J. F. Haddon and J. F. Boyce. Image segmentation by unifying region and boundary information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):929–948, Oct. 1990.

[66] R. M. Haralick. Digital step edges from zero crossing of second directional derivatives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(1):58–68, Jan. 1984.

[67] R. M. Haralick. Author's reply. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7(1):127–129, Jan. 1985.

[68] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision*, volume I. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1992.

[69] C. Heckler and L. Thiele. Parallel complexity of lattice basis reduction and a floating-point parallel algorithm. In A. Bode, M. Reeve, and G. Wolf, editors, *Proceedings of the Parallel Architectures and Languages Europe (PARLE'93)*, volume LNCS 694, pages 744–747, Munich, 1993. Springer-Verlag.

[70] E. C. Hildreth. The computation of the velocity field. *Proceedings of the Royal Society of London – B*, 221:189–220, 1984.

[71] A. S. Hornby, A. P. Cowie, and A. C. Gimson. *Oxford Advanced Learner's Dictionary of Current English.* Oxford University Press, Oxford, 21st edition, 1987 [1948].

[72] S. L. Horowitz and T. Pavlidis. Picture segmentation by a tree traversal algorithm. *Journal of the Association for Computing Machinery*, 23(2):368–388, Apr. 1976.

[73] M. Hötter. Object-oriented analysis-synthesis coding based on moving two-dimensional objects. *Signal Processing: Image Communication*, 2(4):409–428, Dec. 1990.

[74] T. S. Huang. Coding of two-tone images. *IEEE Transactions on Communications*, COM-25(11):1406–1424, Nov. 1977.

[75] A. Huertas and G. Medioni. Detection of intensity changes with subpixel accuracy using laplacian-gaussian masks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(5):651–664, Sept. 1986.

[76] R. Hunter and A. H. Robinson. International digital facsimile coding standards. *Proceedings of the IEEE*, 68(7):854–867, July 1980.

[77] ISO/IEC JTC1/SC29/WG11. Coding of audio-visual objects: Visual. Committee Draft ISO/IEC 14496-2, JTC1/SC29/WG11 N2202, ISO, Tokyo, Mar. 1998.

[78] Improved spline implementation. Proposal COST211ter, Simulation Subgroup, SIM(94)41, IST (Portugal), June 1994.

[79] Removal of redundant vertices. Proposal COST211ter, Simulation Subgroup, SIM(94)40, IST (Portugal), June 1994.

[80] Parameter values for the HDTV standards for production and international programme exchange. Recommendation BT.709-2, ITU-R, 1995.

[81] R. C. Jain and T. O. Binford. Ignorance, myopia, and naiveté in computer vision systems. *CVGIP: Image Understanding*, 53(1):112–117, Jan. 1991.

[82] Progressive bi-level image compression. Recommendation T.82, ITU-T, Mar. 1993.

[83] R. Jeannot, D. Wang, and V. Haese-Coat. Binary image representation and coding by a double-recursive morphological algorithm. *Signal Processing: Image Communication*, 8(3):241–266, Apr. 1996.

[84] E. Jensen, K. Rijkse, I. Lagendijk, and P. van Beek. Coding of arbitrarily-shaped image sequences. In WIASIC'94 [198], page E2.

[85] H. Jeong and C. I. Kim. Adaptive determination of filter scales for edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(5):579–585, May 1992.

[86] T. Kaneko and M. Okudaira. Encoding of arbitrary curves based on the chain code representation. *IEEE Transactions on Communications*, 33(7):697–707, July 1985.

[87] P. Kauff and K. Schüür. Shape-adaptive DCT with block-based DC separation and ΔDC correction. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(3):237–242, June 1998.

[88] A. Kaup and T. Aach. Coding of segmented images using shape-independent basis functions. *IEEE Transactions on Image Processing*, 7(7):937–947, July 1998.

[89] J.-L. Kim, J.-I. Kim, J.-T. Lim, J.-H. Kim, H.-S. Kim, K.-H. Chang, and S.-D. Kim. Daewoo proposal for object scalability. Technical Description ISO/IEC JTC1/SC29/WG11 MPEG96/0554, Daewoo Electronics CO.LTD. and KAIST, Jan. 1996.

[90] W. Kou. *Digital Image Compression: Algorithms and Standards.* Kluwer Academic Publishers, Boston, Massachusetts, 1995.

[91] V. A. Kovalevsky. Finite topology as applied to image analysis. *Computer Vision, Graphics, and Image Processing*, 46(2):141–161, May 1989.

[92] V. A. Kovalevsky. Topological foundations of shape analysis. volume 126 of *ASI Series F: Computer and Systems Sciences*, pages 21–36. NATO, 1993.

[93] M. K. Kundu, B. B. Chaudhuri, and D. D. Majumder. A generalised digital contour coding scheme. *Computer Vision, Graphics, and Image Processing*, 30:269–278, 1985.

[94] M. Kunt. Comments on "Dialogue," a series of articles generated by the paper entitled "Ignorance, myopia, and naiveté in computer vision". *CVGIP: Image Understanding*, 54(3):428–429, Nov. 1991.

[95] M. Kunt. What MPEG-4 means to me. *Signal Processing: Image Communication*, 9(4):473, May 1997.

[96] M. Kunt, A. Ikonomopoulos, and M. Kocher. Second-generation image-coding techniques. *Proceedings of the IEEE*, 73(4):549–574, Apr. 1985.

[97] M. S. Landy and Y. Cohen. Vectorgraph coding: Efficient coding of line drawings. *Computer Vision, Graphics, and Image Processing*, 30:331–344, 1985.

[98] H.-C. Lee and D. R. Cok. Detecting boundaries in a vector field. *IEEE Transactions on Signal Processing*, 39(5):1181–1194, May 1991.

[99] C. Lettera and L. Masera. Foreground/background segmentation in videotelephony. *Signal Processing: Image Communication*, 1(2):181–189, Oct. 1989.

[100] J. S. Lim. *Two-Dimensional Signal and Image Processing.* Prentice-Hall International, Inc., 1990.

[101] Y.-T. Liow. A contour tracing algorithm that preserves common boundaries between regions. *CVGIP: Image Understanding*, 53(3):313–321, May 1991.

[102] A. C. Luther. *Video Camera Technology.* Artech House, Boston, Massachusetts, 1998.

[103] P. A. Maragos and R. W. Schafer. Morphological skeleton representation and coding of binary images. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34(5):1228–1244, Oct. 1986.

[104] F. Marqués. *Multiresolution Image Segmentation Based on Compound Random Fields: Application to Image Coding.* PhD thesis, Universitat Politècnica de Catalunya, Departament de Teoria del Senyal i Comunicacions, Barcelona, Dec. 1992.

[105] F. Marqués, M. Pardàs, and P. Salembier. Coding-oriented segmentation of video sequences. In L. Torres and M. Kunt, editors, *Video Coding: The Second Generation Approach*, chapter 3, pages 79–123. Kluwer Academic Publishers, Boston, Massachusetts, 1996.

[106] F. Marqués, J. Sauleda, and A. Gasull. Shape and location coding for contour images. In *Proceedings of the Picture Coding Symposium (PCS'93)*, page 18.6, Lausanne, Mar. 1993. Swiss Federal Institute of Technology.

[107] D. Marr. *Vision*. W. H. Freeman and Company, New York, 1982.

[108] D. Marr and E. C. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London – B*, 207:187–217, 1980.

[109] R. A. Mattheus, A. J. Duerinckx, and P. J. van Otterloo, editors. *Video Communications and PACS for Medical Applications*, volume Proc. SPIE 1977, Berlin, Apr. 1993. EOS and SPIE, SPIE – The International Society for Optical Engineering.

[110] P. C. McLean. Structured video coding. Master's thesis, Massachusetts Institute of Technology, June 1991.

[111] P. Meer, D. Mintz, and A. Rosenfeld. Robust regression methods for computer vision: A review. *International Journal of Computer Vision*, 6(1):59–70, 1991.

[112] F. Melindo. *Tecnologie di Elaborazione e Intelligenza Artificiale Nelle Telecomunicazioni*. Centro Studi e Laboratori Telecomunicazioni (CSELT), Torino, 1991.

[113] M. Menezes de Sequeira. Computational weight of global motion cancellation. RACE-MAVT Contribute 72/IST/WP2.1/C/R/10.3.93/016, IST, Mar. 1993.

[114] M. Menezes de Sequeira. COST/MAVT RM1 implementation and future improvements. RACE-MAVT Contribute 72/IST/WP6.2/NC/R/4.5.94/027, IST, UPC, Barcelona, May 1994.

[115] M. Menezes de Sequeira. Proposal for future work within the "ad hoc group on motion analysis and segmentation for compression". RACE-MAVT Contribute 72/IST/WP6.2/NC/R/21.11.94/31 VID94#36, IST, CMM, Fontainebleau, Nov. 1994.

[116] M. Menezes de Sequeira. Quick cubic spline implementation. RACE-MAVT Contribute 72/IST/WP6.2/NC/R/9.2.94/022, IST, Ulm, Mar. 1994.

[117] M. Menezes de Sequeira. Motion analysis for segmentation-based encoding of video sequences. In Santos et al. [178].

[118] M. Menezes de Sequeira. *Video Coding Libraries*. Instituto de Telecomunicações, IST, UTL, Lisboa, 1996.

[119] M. Menezes de Sequeira. Time recursive split & merge segmentation of video sequences. In *Actas da Conferência Nacional de Telecomunicações*, pages 333–336, Aveiro, Apr. 1997. Instituto de Telecomunicações.

[120] M. Menezes de Sequeira and D. Cortez. Partitions: A taxonomy of types and representations and an overview of coding techniques. Technical Report Image Group 001, Instituto de Telecomunicações, IST, Torre Norte, 1096 LISBOA CODEX, Portugal, May 1996.

[121] M. Menezes de Sequeira and D. Cortez. Partitions: A taxonomy of types and representations and an overview of coding techniques. *Signal Processing: Image Communication*, 10(1–3):5–19, July 1997.

[122] M. Menezes de Sequeira and F. Pereira. Global motion compensation and motion vector smoothing in an extended H.261 recommendation. In Mattheus et al. [109], pages 226–237.

[123] M. Menezes de Sequeira and F. Pereira. Knowledge based videotelephone sequence segmentation. RACE-MAVT Contribute 72/IST/WP2.1/C/R/22.2.93/015, IST, QMWC, London, Feb. 1993.

[124] M. Menezes de Sequeira and F. Pereira. Knowledge-based videotelephone sequence segmentation. In B. G. Haskell and H.-M. Hang, editors, *Proceedings of the SPIE's Symposium on Visual Communications and Image Processing (VCIP'93)*, volume Proc. SPIE 2094 (3 volumes), pages 858–869, Cambridge, Massachusetts, Nov. 1993. SPIE – The International Society for Optical Engineering.

[125] M. Menezes de Sequeira and F. Pereira. Knowledge based videotelephone sequence segmentation II. RACE-MAVT Contribute 72/IST/WP2.1/C/R/27.7.93/019, IST, Lisboa, July 1993.

[126] M. Menezes de Sequeira and F. Pereira. Knowledge-based videotelephone sequence segmentation. In G. Nitsche, editor, *Video Coding Algorithm for Broadband Transmission*, number R2072/BOS/2.1/DS/R/015/b1, CEC deliverable 3, pages 18–32. CEC, Jan. 1994.

[127] Global motion compensation and cancellation. RACE-MAVT Contribute 72/IST/WP2.1/NC/R/13.11.92/009, IST, Munich, Nov. 1992.

[128] Global motion compensation and cancellation – contribute to deliverable 9. RACE-MAVT Contribute 72/IST/WP2.1/NC/R/4.12.92/013, IST, Hildesheim, Nov. 1992.

[129] Global motion compensation and motion vector smoothing in an extended H.261 recommendation. RACE-MAVT Contribute 72/IST/WP2.1/NC/R/13.11.92/005, IST, Lisboa, Sept. 1992.

[130] Global motion detection for camera vibration cancellation. RACE-MAVT Contribute 72/IST/WP2.1/NC/R/13.11.92/010, IST, Munich, Nov. 1992.

[131] F. Meyer. Colour image segmentation, 1992.

[132] F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication and Image Representation*, 1(1):21–46, Sept. 1990.

[133] T. H. Morrin, II. Chain-link compression of arbitrary black-white images. *Computer Graphics and Image Processing*, 5:172–189, 1976.

[134] O. J. Morris, M. de J. Lee, and A. G. Constantinides. Graph theory for image analysis: an approach based on the shortest spanning tree. *IEE Proceedings – Part F*, 133(2):146–152, Apr. 1986.

[135] "Motseg" ad hoc group. Activity report of the "ad hoc group on motion analysis and segmentation for compression". RACE-MAVT Contribute 72/IST/WP6.2/NC/R/21.11.94/30 VID94#34, CCETT, CNET, CSELT, Daimler-Benz, IST, PTT, and Thomson, CMM, Fontainebleau, Nov. 1994.

[136] Coding of moving pictures and associated audio for digital storage media up to about 1,5 Mbit/s. International Standard 11172, ISO/IEC, 1993.

[137] Generic coding of moving pictures and associated audio information. Draft Recommendation H.262, Draft International Standard 13818, ITU-T, ISO/IEC, Jan. 1995.

[138] MPEG-4 Requirements, Audio, DMIF, SNHC, Systems and Video. MPEG-4 overview. Document ISO/IEC JTC1/SC29/WG11 N1909, ISO, Fribourg, Oct. 1997.

[139] MPEG AOE Subgroup. Proposal package description (PPD) – revision 1.0. Document ISO/IEC JTC1/SC29/WG11 N821, ISO, Nov. 1994.

[140] MPEG Video Subgroup. Core experiments on MPEG-4 video shape coding. Document ISO/IEC JTC1/SC29/WG11 N1326, ISO, July 1996.

[141] H. G. Musmann, P. Pirsch, and H.-J. Grallert. Advances in picture coding. *Proceedings of the IEEE*, 73(4):523–548, Apr. 1985.

[142] N. Negroponte. *Being Digital.* Vintage Books, 1996.

[143] A. N. Netravali and B. G. Haskell. *Digital Pictures: Representation, Compression, and Standards.* Applications of Communications Theory. Plenum Press, New York, 2nd edition, 1995.

[144] A. N. Netravali and J. O. Limb. Picture coding: A review. *Proceedings of the IEEE*, 68(3):366–406, Mar. 1980.

[145] A. N. Netravali and J. D. Robbins. Motion-compensated television coding: Part I. *The Bell System Technical Journal*, 58(3):631–670, Mar. 1979.

[146] P. Nunes. Personal communication, June 1998.

[147] P. J. L. Nunes. Detecção de fronteiras em imagens texturadas. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, Aug. 1995.

[148] K. O'Connell and D. Tull. Motorola MPEG-4 contour-coding tool technical description. Technical Description ISO/IEC JTC1/SC29/WG11 MPEG95/0447, Motorola, Nov. 1995.

[149] S. Okubo. What does MPEG-4 mean to me. *Signal Processing: Image Communication*, 9(4):477, May 1997.

[150] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing.* Prentice-Hall International, Inc., Englewood Cliffs, New Jersey, 1989.

[151] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-9(1):62–66, Jan. 1979.

[152] D. W. Paglieroni and A. K. Jain. A control point theory for boundary representation and matching. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'85)*, pages 1851–1854, Tampa, Florida, 1985. IEEE, Signal Processing Society.

[153] M. Pardàs and P. Salembier. Joint region and motion estimation with morphological tools. In *Proceedings of the International Workshop on Mathematical Morphology and its Applications to Signal Processing*, Fontainebleau, Sept. 1994. CMM.

[154] T. Pavlidis. *Structural Pattern Recognition.* Springer-Verlag, Berlin, 1977.

[155] T. Pavlidis. Contour filling in raster graphics. *Computer Graphics*, 15(3):29–36, July 1981.

[156] T. Pavlidis. *Algorithms for Graphics and Image Processing.* Computer Science Press, Inc., Rockville, Maryland, 1982.

[157] T. Pavlidis and S. L. Horowitz. Segmentation of plane curves. *IEEE Transactions on Computers*, 23(8):860–870, Aug. 1974.

[158] T. Pavlidis and Y.-T. Liow. Integrating region growing and edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3):225–233, Mar. 1990.

[159] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr., and R. B. Arps. An overview of the basic principles of the q-coder adaptive binary arithmetic coder. *IBM Journal of Research and Development*, 32(6):717–726, Nov. 1988.

[160] F. Pereira. *Variable Bit Rate Video Coding.* PhD thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, 1991.

[161] F. Pereira, D. Cortez, and P. Nunes. Mobile videotelephone communications: the CCITT H.261 chances. In Mattheus et al. [109], pages 168–179.

[162] R. W. Picard. Content access for image/video coding: "the fourth criterion". Technical Report 295, MIT Media Lab: Perceptual Computing Section, 1994.

[163] R. Plompen. *Motion Video Coding for Visual Telephony.* PTT Research Neher Laboratories, Leidshendam, 1989.

[164] C. A. Poynton. Frequently asked questions about color. FAQ, http://www.inforamp.net/ poynton/ColorFAQ.html, Mar. 1997.

[165] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, Massachusetts, 1994.

[166] M. J. V. Preto and M. J. de Macedo e Pimentel. Segmentação de imagens videotelefónicas. Trabalho final de curso, Lisboa, Jan. 1993.

[167] Objects with different priorities in SIMOC. Discussion COST211ter, Simulation Subgroup, SIM(94)34, PTT Research, Tampere, June 1994.

[168] Quantel, http://www.usa.quantel.com/dfb/. *The Digital Fact Book*, 9th edition, 1998.

[169] M. P. Queluz. *Multiscale Motion Estimation and Video Compression.* PhD thesis, Université Catholique de Louvain, Apr. 1996.

[170] N. Robertson, D. P. Sanders, P. Seymour, and R. Thomas. A new proof of the four-colour theorem. *Electronic Research Announcements of the American Mathematical Society*, 2(1), 1996.

[171] C. Ronse and B. Macq. Morphological shape and region description. *Signal Processing*, 25(1):91–105, Oct. 1991.

[172] K. H. Rosen. *Discrete Mathematics and its Applications.* McGraw-Hill, Inc., New York, 1991.

[173] A. Rosenfeld. Digital straight line segments. *IEEE Transactions on Computers*, 23(12):1264–1269, Dec. 1974.

[174] A. Rosenfeld. Image analysis and computer vision: 1990. *CVGIP: Image Understanding*, 53(3):322–365, May 1991.

[175] P. Saint-Marc, H. Rom, and G. Medioni. B-spline contour representation and symmetry detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1191–1197, Nov. 1993.

[176] P. Salembier, F. Meyer, and M. Pardàs. Multiscale representation of images. CEC Deliverable R2053/UPC/GPS/DS/R/003/b1, UPC, CMM, 1993.

[177] P. Salembier and M. Pardàs. Hierarchical morphological segmentation for image sequence coding. *IEEE Transactions on Image Processing*, 3(5):639–651, Sept. 1994.

[178] B. S. Santos, J. A. Rafael, A. M. Tomé, and A. S. Pereira, editors. *Proceedings of the 7th Portuguese Conference on Pattern Recognition (RecPad'95)*, Aveiro, Mar. 1995.

[179] S. Sarkar and K. L. Boyer. On optimal infinite impulse response edge detection filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(11):1154–1171, Nov. 1991.

[180] J. Serra, editor. *Image Analysis and Mathematical Morphology*, volume II. Academic Press, Inc., San Diego, California, 1992.

[181] J. Serra. *Image Analysis and Mathematical Morphology*, volume I. Academic Press, Inc., San Diego, California, 1993.

[182] U. Shani. Filling regions in binary raster images: a graph-theoretical approach. *Computer Graphics*, 14(3):321–327, July 1980.

[183] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, XXVII:379–423, 623–656, 1948.

[184] T. Sikora and B. Makai. Low complex shape-adaptive DCT for generic and functional coding of segmented video. In WIASIC'94 [198], page E1.

[185] C. Stiller. Motion-estimation for coding of moving video at 8 kbit/s with Gibbs modeled vectorfield smoothing. In M. Kunt, editor, *Proceedings of the SPIE's Symposium on Visual Communications and Image Processing (VCIP'90)*, volume Proc. SPIE 1360 (3 volumes), pages 468–476, Lausanne, Oct. 1990. SPIE and Swiss Federal Institute of Techonology, SPIE – The International Society for Optical Engineering.

[186] K. Thulasiraman and M. N. S. Swamy. *Graphs: Theory and Algorithms.* John Wiley & Sons, Inc., New York, 1992.

[187] V. Torre and T. A. Poggio. On edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(2):147–163, Mar. 1986.

[188] Technical description for MPEG-4 first round of test. Technical Description ISO/IEC JTC1/SC29/WG11 MPEG95/0354, Toshiba, Nov. 1995.

[189] H. J. Trussell. DSP solutions run the gamut for color systems. *IEEE Signal Processing Magazine*, 10(2):8–23, Apr. 1993.

[190] G. Tziritas and C. Labit. *Motion Analysis for Image Sequence Coding.* Advances in Image Communication. Elsevier, Amsterdam, 1994.

[191] Shape coding with an optimized morphological region description. Contribute COST211ter, Simulation Subgroup, SIM(92)23, U.C.L., Feb. 1992.

[192] K. Uomori, A. Morimura, and H. Ishii. Electronic image stabilization system for video cameras and VCRs. *SMPTE Journal*, pages 66–75, Feb. 1992.

[193] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, June 1991.

[194] T. Vlachos and A. G. Constantinides. Graph-theoretical approach to colour picture segmentation and contour classification. *IEE Proceedings – Part I*, 140(1):36–45, Feb. 1993.

[195] J. Y. A. Wang and E. H. Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing*, 3(5):625–638, Sept. 1994.

[196] S. Watanabe, H. Saiga, H. Katata, and H. Kusao. Binary shape coding based on hierarchical chain codes. Technical Description ISO/IEC JTC1/SC29/WG11 MPEG96/1045, Sharp Corporation, July 1996.

[197] T. A. Welch. A technique for high-performance data compression. *IEEE Transactions on Computers*, pages 8–19, June 1984.

[198] *Proceedings of the Workshop on Image Analysis and Synthesis in Image Coding (WIASIC'94)*, Berlin, Oct. 1994.

[199] R. Wilson and G. H. Granlund. The uncertainty principle in image processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):758–767, Nov. 1984.

[200] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, Nov. 1993.

[201] C. A. Wüthrich and P. Stucki. An algorithmic comparison between square- and hexagonal-based grids. *CVGIP: Graphical Models and Image Processing*, 53(4):324–339, July 1991.

[202] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–343, May 1977.