A NOVEL ENERGY FUNCTION AND ITS MINIMIZATION FOR VIDEO OBJECT SEGMENTATION AND COMPRESSION

by

Michael King
B.Sc., Simon Fraser University, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School of Computing Science

© Michael King 2005
SIMON FRASER UNIVERSITY
Summer 2005

All rights reserved. This work may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.
APPROVAL

Name: Michael King

Degree: Master of Science

Title of thesis: A Novel Energy Function and Its Minimization for Video Object Segmentation and Compression

Examining Committee: Dr. Richard Zhang
Chair

Dr. Ze-Nian Li, Professor, Computing Science, Simon Fraser University, Senior Supervisor

Dr. Mark Drew, Associate Professor, Computing Science, Simon Fraser University, Supervisor

Dr. M Stella Atkins, Professor, Computing Science, Simon Fraser University, Examiner

Date Approved: August 3, 2005
PARTIAL COPYRIGHT LICENCE

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author’s written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

W. A. C. Bennett Library
Simon Fraser University
Burnaby, BC, Canada
Abstract

Traditional digital video compression techniques focus on image-block encoding. Conversely, the MPEG-4 standard specifies that a video should be composed of Video Object Planes, improving compression through spatial cohesion. MPEG-4 video compression can incorporate two key observations: First, a block containing different textures needs to be split to enhance compression. Second, Motion Vectors are detected more accurately by grouping blocks into regions, allowing for more efficient predictive coding.

We propose a novel discrete cosine transform energy function $E_{DCT}$ measuring block compression for image and video segmentation. By minimizing $E_{DCT}$ the best possible split of the block can be found. Consistent motion vectors can be obtained by using an innovative energy function in the spatio-temporal domain which measures block motion over multiple frames.

Tests on images and video show promising results, where still image compression achieves approximately 15% improvement over JPEG and our video compression achieves similar improvement over MPEG-2.
Acknowledgments

I would like to thank Dr. Ze-Nian Li who gave me the motivation to join the masters program as well as countless hours of help. I would also like to thank Dr. Mark Drew and Dr. Wo-Shun Luk for their support and guidance. I’d like to thank Dr. M. Stella Atkins for her help over the years and for serving as my examiner. I would also like to thank Zinovi Tauber for his invaluable suggestions and help in my research. Lastly, I would like to thank my family who have supported me throughout my academic years. It has been an enjoyable experience thanks to everyone in the Vision and Media Lab who have helped me during my studies.
Contents

Approval ii
Abstract iii
Acknowledgments iv
Contents v
List of Tables vii
List of Figures viii

1 Introduction 1
  1.1 Object-based Video Compression .............................. 1
  1.2 Motivation ............................................... 3
  1.3 Thesis Organization ........................................ 6

2 Survey of Relevant Work 7
  2.1 Image Segmentation ........................................ 7
  2.2 Video segmentation ........................................ 10
  2.3 Block-Based Motion Estimation .............................. 12
  2.4 MPEG-4 ..................................................... 13

3 A New Energy Function for Image Segmentation 16
  3.1 The New Energy Function ................................... 17
  3.2 Sub-Block Creation ......................................... 20
  3.3 Region Growing ............................................ 27
# Extension to Video Object Segmentation

4.1 3D DCT Motion Energy ........................................................................... 32
4.2 Motion Detection .................................................................................... 33
   4.2.1 Diamond Motion Energy Search ....................................................... 33
   4.2.2 Graph Cuts .................................................................................... 35

# Image and Video Encoding

5.1 Object-Based Image Encoding ................................................................. 39
   5.1.1 Texture and Color Encoding .......................................................... 40
   5.1.2 Shape Coding ............................................................................. 43
5.2 Object-Based Video Encoding ................................................................. 45
   5.2.1 Object-Based I-Frame Encoding ...................................................... 45
   5.2.2 Object-Based P-Frame Encoding ..................................................... 45

# Experimental Results

6.1 Image Segmentation .............................................................................. 49
6.2 Image Compression .............................................................................. 60
6.3 Video Object Segmentation and Compression ........................................ 60

# Conclusions and Future Work

7.1 Conclusions ......................................................................................... 64
7.2 Future Work ....................................................................................... 65

Bibliography ............................................................................................. 67
List of Tables

6.1 Comparative image compression results in bits ........................................ 60
6.2 Comparative video compression results in bits ...................................... 61
List of Figures

2.1 Three locales in an image .............................................. 10
2.2 Steps of SA-DCT ......................................................... 15

3.1 The absolute sum of groups of similar frequency components are similar, as illustrated by the similar absolute sums in the two blocks. ................. 18
3.2 Simple and complex texture blocks produce low and high energy, respectively. 19
3.3 Consistent regions produce low energy, while blocks between regions produce high energy ........................................................................ 20
3.4 A 256 × 256 pixel image. The gray level indicates the relative magnitude of energy in (b). Energy is low in homogeneous texture regions, while high between them. ......................................................... 20
3.5 Overview of the image segmentation process ......................................... 21
3.6 Sobel filters for edge detection .................................................. 22
3.7 The threshold determination of an 8 × 8 block is demonstrated. A block with a real edge as well as a edge that should be threshold out is shown ......... 25
3.8 Procedure for energy minimization for block sub-division .................... 27
3.9 Procedure for Pyramid linking .................................................. 30

4.1 We assume a monotonically decreasing energy toward the minimum energy. In step one (b), the minimum energy is found toward the right search area. Again the best energy found is toward the right in the second step (c). In (d) the third step, the area is searched, but no better energy is found. The search pattern is adjusted and the minimum energy is found in (e) ....... 34
4.2 Motion energy space of a block where the true motion is to the right side. The pixel luminance reflects the value of the the energy ................. 34
4.3 A graph constructed with 3 sub-blocks \( p, q \) and \( r \) is shown in (a). The sub-block \( p \) is currently labeled the sink, \( p \) and \( r \) are labeled as part of the source. (b) shows a cut where there are no changes, which implies the minimum cost cut is obtained from \( t_p^b + t_q^b + t_r^b + (2 \ast n_\delta) \). 

5.1 Overview of the image encoding process ........................................ 40
5.2 Standard JPEG quantization tables ............................................ 41
5.3 Zig-zag pattern .............................................................. 42
5.4 An example of the RLE encoding, it works well when heavily quantized, leaving many zeros. It actually causes expansion if there are not enough zeros. 42
5.5 Example of IQTSC applied on an \( 8 \times 8 \) BAB. The scan order used is SW, SE, NW, NE. ......................................................... 44
5.6 Two BABs, with the same output tree in unmodified IQTSC. The scan order used is SW, SE, NW, NE. ......................................................... 45
5.7 P-frame encoding ................................................................ 48

6.1 (a) Lena Image at a \( 512 \times 512 \) pixel resolution, (b) Energy feature map ... 49
6.2 (a) Edge detection on the \( 8 \times 8 \) window feature map, (b) edge shifting (b) on the Lena image ......................................................... 50
6.3 (a) Binary threshold map, (b) Energy feature extraction using a \( 4 \times 4 \) window (b) on the Lena image ......................................................... 51
6.4 (a) Edge detection on the \( 4 \times 4 \) window feature map, (b) edge shifting on the Lena image ......................................................... 52
6.5 (a) Threshold 6.4b on the binary map in 6.3a, (b) edge thinning on the Lena image ......................................................... 53
6.6 (a) Use adaptive threshold, (b) edge extension on the Lena image .......... 54
6.7 (a) Divide sub-blocks, (b) Region growing on the Lena image ................. 54
6.8 Original parrots image at a \( 384 \times 256 \) pixel resolution ......................... 55
6.9 Parrots energy feature map ......................................................... 56
6.10 Processed image after edge extension ............................................ 56
6.11 Region growing on the Parrots image ............................................ 57
6.12 (a) Baboon Image at a \( 256 \) by \( 256 \) pixel resolution, (b) Energy feature map . 58
6.13 (a) Divide sub-blocks, (b) Region growing on the Baboon image .......... 59
6.14 (a) An artificial image, (b) the regions found ................................. 59
6.15 (a) An artificial image. (b) the regions found

6.16 A fast moving toy car

6.17 3D motion cost for three blocks. The higher the energy, the brighter the pixel on the 3D motion plane.

6.18 This graph shows how the motion search finds the minimum energy in Figure 6.17. Energy Ratio is the energy divided by the maximum energy of the search window.

6.19 The direction for each undivided block is shown with a red arrow. The best direction is found using a diamond search which causes unreliable true motion. When the energy is minimized for a region, a good motion vector can be found.

6.20 A slow moving artificial video of a diamond shape slowly moving towards the right side.
Chapter 1

Introduction

1.1 Object-based Video Compression

As the popularity and demand for high-quality digital video grows, so does the need for greater video compression and accessibility. Digital video compression is needed because of constraints in digital storage and network throughput. These constraints limit the transmission of real-time video as well as the length and quality of video which can be stored on disks. For example, a one minute uncompressed video without audio at a 640 × 480 resolution will be approximately 200MB in size. If viewed over a network, it would need 28Mbps of throughput, which is generally unattainable in modern communication networks. The most prevalent video-disk format, the DVD, would only be able to store approximately 21 minutes of such a video, which isn’t sufficient for its primary purpose: storing feature length films.

Since the early 90’s, significant progress has been made in video compression as well as computer networks that has allowed users to view and store videos. Many types of videos are available online, such as trailers, movies, and real-time broadcasts of events. Videoconferencing between people using the internet has become commonplace with the proliferation of internet service and web cameras. The inability for video compression to satisfy the needs of viewers is evident with the advent of next generation video-disk formats which have over ten times the storage capacity of DVDs. Two major concerns exist however: how to encode digital videos so that they can be stored and transmitted with limited resources; and how to index and search digital videos efficiently.

Traditionally, digital video is encoded as a whole, without taking into account the content
CHAPTER 1. INTRODUCTION

of the video [3]. In traditional video compression, block-based encoding is used, where pixels are divided into squares regardless of content. Blocks which contain parts of two or more objects compress very poorly since the block contains an irregular texture pattern.

Object-based video encoding methods take objects, ideally real world objects, and encode them independently [25]. Object-based video encoding methods have a distinct advantage over traditional video encoding methods by having a greater compression rate, being more scalable, and supporting interaction with the video. The MPEG-4 digital video encoding standard is a modern object-based system that has been widely accepted. Object-based compression is superior to traditional video compression because of the observation that regions within an object have similar texture and color properties. Digital video is encoded using the Discrete Cosine Transform (DCT) transform which excels in compressing regular patterns in texture [3]. Since pixels are grouped into regions with similar texture, these textures can be compressed efficiently.

Generally, regions within an object also move in a similar direction in a video. This property can be used to achieve a higher compression due to increased accuracy in temporal prediction. Since each object is encoded independently, this allows scalable quality for each object, instead of for the entire scene. Detail can be sacrificed from unimportant objects in the scene, such as the background, while maintaining high visual quality for important objects, such as the main actors in a movie. Interactivity can be used so that the watcher of the digital video can influence the appearance and actions of the video.

Many other useful applications of video object segmentation can be achieved as well. For example, additional information could be embedded into the digital video so that video object segmentation not only provides superior compression, it also allows for a unified platform to perform activities such as video indexing and searching. Automatic descriptors of the region can be stored with the video object, with little additional space or computation. Descriptors such as color, texture, shape, and motion are already needed to segment and track these objects during encoding, so very little extra processing is needed. Objects segmented in the video could be readily indexed into a database where users could query them, whether it is to look for a specific scene in a video or to find a particular video based on specific objects.

Object-based encoding depends on manual, semi-automatic, or automatic methods to segment objects from a digital video. Manual or even semi-automatic methods to segment objects in a large digital video library are not practical, so research has been devoted to
automatic segmentation methods. Finding and segmenting objects automatically in a digital video is a very difficult task because the algorithm must understand the scene in the video to be able to segment the scene into real-world objects. However, without an understanding of the scene, a meaningful segmentation can often be achieved based on texture and color.

1.2 Motivation

Recent advances in digital video compression attempt to improve compression by altering constraints such as block shape or precision of estimated temporal motion in digital video. Efforts to use different spatial-frequency transformations which are the basis for most digital video and image compression methods have met with varied success. All these methods attempt to encode the digital video without regard for the content of the digital video they encode.

Although these image-independent methods allow for greater compression, significant gains in compression can be achieved by an object-based method, where spatial and temporal predictions will be more accurate. Since objects often have similar color and texture properties, color and texture predictions are more accurate when done within one object instead of a whole image. Similarly, motion from one frame to the next can be more easily predicted by grouping pixels into objects. Since an object generally moves in a single motion, it is possible to predict with greater accuracy the movement of each portion of the object. A large savings in space is also possible by keeping distinct textures separate, enabling a spatial-to-frequency transformation to have relatively low frequencies.

In digital video segmentation, an intuitive way to segment objects is to find regions which move in a temporally similar way. Many methods have been developed for the purpose of finding temporally homogeneous regions [3]. Some methods group pixels which have similar temporal motion together into one object. An alternative approach is to first segment the image based on color and texture features, then track these objects in time.

Segmenting objects in an image or a video is a difficult task for a broad image or video domain. Pixel level segmentation methods suffer from problems in overcoming noise in the image or video, making it difficult to label pixels on the border between two or more objects. A high-level multi-resolution approach suffers from the opposite problem; it is difficult to determine which pixels belong to which object because of the loss of high-resolution details. An accurate division is extremely difficult without both general regional information and
accurate pixel statistics.

Li, Zaiane and Tauber [20] introduced the concept of feature localization as an alternative to object segmentation. Instead of object regions, locales are proposed. Unlike regions, locales can overlap each other, and pixels in the locale need not be connected. Locales are made out of tiles, which are square blocks of pixels. Pixel level information is present in tiles so that the details are not lost. Localization allows for the detail of pixel level information, while having the robustness of a reduced-resolution method.

This thesis describes a novel energy function used for image segmentation and compression. An extension of this energy function is used to find the pixel motion in a video is also described. Using the novel energy functions, a framework for object-based video compression is proposed.

Object-based video compression requires pixel precise boundaries in order to accurately decode the video. The coarse segmentation used for localization is designed for image retrieval which is an application that doesn’t require a precise boundary: tiles keep pixel level statistics but not the spatial location of the pixels with similar features. The spatial information is needed, so instead of the tile being the smallest unit, the sub-block is used, which has the same information as a tile but has exact spatial information.

In feature localization the tile is an image independent unit, where the image is divided into equal sized square blocks, and each block is a tile. Each tile contains a list of colors to aid in the formation of locales. Similarly, sub-blocks are created by grouping pixels of common texture and color together: while tiles find pixels with similar features, sub-blocks also divide them based on features.

The generation of these sub-blocks can be achieved by detecting irregularities in the texture patterns. Since texture is a neighborhood property it is difficult to analyze in the spatial domain. Instead of the spatial domain, texture should be analyzed in the frequency domain which can naturally describe textures.

Consider a one dimensional group of pixels. Its pixel value is a function of its location. These pixel values can be thought of as samples of a waveform. This waveform can not only be described from samples in space, it can also be described as a combination of frequencies, or waves. The waveform can be represented as a combination of sine waves with different wavelength. The values of the pixels can be a function of frequency, given a particular wavelength of a sine wave, the amplitude is known. By combining the waves from the different wavelengths the original waveform can be reconstructed. In turn, this can
be transformed into pixel values. This transform is useful because it is known that there are few high frequency components in natural images [2]. If a block is properly divided into a number of arbitrarily shaped blocks called *sub-blocks*, the number of high frequency components can be reduced.

A sub-block is an arbitrarily shaped group of pixels. Unlike in traditional image encoding which consists only of regularly square blocks of $8 \times 8$ or $16 \times 16$, object-based image encoding allows arbitrarily shaped sub-blocks. A block consists of one or more sub-blocks.

A large number of high frequency components in a block is generally caused by one of two situations, either there is a complex texture like carpeting, or more likely, there is a combination of more than one texture pattern. In the latter case, by separating the pixels into groups which have the same texture pattern, many of the high frequency components can be eliminated. Finding this boundary will help a segmentation algorithm to find the boundaries between regions.

Once the sub-blocks are created, a pyramidal linking scheme similar to the locale creation is used to grow the regions. Instead of linking by dominant color, color and texture features are used. Once the first image in the digital video scene is segmented, video segmentation begins by estimating the motion of each region in the image. Motion estimation is done using three dimensional spatio-temporal transformation. Often objects move just a few pixels from one frame to the next, which can be approximated by a 2D translation motion. This in turn produces relatively few high frequency components, since the changes in the time domain would be low, resulting in low frequencies in the time domain. The region's movements can be estimated by attempting to find the motion which has the fewest high frequency components while keeping coherence between blocks of a region.

Compression is then achieved by encoding each region separately. Without segmentation, there exist blocks which contain two or more regions; these blocks typically contain many high frequency components, which result in poor compression. When separated, each texture pattern can be encoded on its own resulting in very few high frequency components, enhancing compression.

The prediction for each block whether it is for average color or movement is more accurate within an object compared to if the video was not segmented. Since often an object moves in a single direction, the *motion vector* (MV) calculated for each block in the object will be very similar to each other, hence easily predictable. The prediction also predicts the block contents after motion, allowing for increased compression by only encoding the error of the
prediction. Similarly, areas within a region generally have similar color properties, so an accurate color prediction can be utilized. The only additional information that needs to be added is a small overhead to keep track of the shape of each object.

1.3 Thesis Organization

The rest of this thesis is laid out as follows. Chapter 2 presents a survey of image and video segmentation techniques followed by an introduction to the MPEG-4 digital video standard. Chapter 3 presents the creation process of sub-blocks in an image followed by the region growing process. Chapter 4 presents the method for determining motion of regions in a digital video. Chapter 5 describes the encoding method for image and videos. Chapter 6 shows segmentation and compression results. Chapter 7 presents future work and conclusions.
Chapter 2

Survey of Relevant Work

2.1 Image Segmentation

In general, the goal of image segmentation is to divide all pixels in an image into groups of regions which correspond to real world objects.

Image segmentation methods can be divided into three categories, strong segmentation, weak segmentation and partitioning [33]. Strong segmentation is the ideal segmentation, dividing an image into regions which correspond to real world objects. Strong segmentation is important in applications such as image retrieval, where real world objects are commonly used to index the image database and for user queries. Unfortunately, strong segmentation is extremely difficult for all but the narrowest image domains since the segmentation algorithm must understand the content of the image to divide it successfully. Weak segmentation divides pixels in an image into regions based on similar texture and color features. Fortunately, in many applications of image segmentation, weak segmentation is sufficient because weakly segmented objects often are part of a single real world object. The last type of image segmentation, called partitioning, is where the image is divided regardless of the content of the image, such as the division of an image into square blocks in the JPEG or MPEG-2 compression format.

Image dependent segmentation begins with the summarization of color and texture information in the image, called features. Features consist of color information from the image which have undergone a transformation to simplify and condense the data. Unlike color which is a point property, texture is a local neighborhood property. Many segmentation methods have been proposed in the past, including random field models [39], operator
All these methods attempt to extract a set of features for each group of one or more pixels in an image.

Once these features are extracted, the segmentation process begins. Segmentation methods can be classified as region-based, boundary-based or a hybrid of the two. Region-based methods attempt to cluster pixels with homogeneous features while boundary detection methods attempt to divide an image based on inconsistent features. Many popular region based methods have been developed, such as region growing [20], and split-and-merge [3]. Boundary-based methods depend on edge detection in the spatial feature space [33].

One of the first texture features proposed [33] calculated the feature value as the difference of the maximum and minimum gray level occurring in a square window around the pixel. This feature could discriminate well between complex and simple textured regions. While this feature can be calculated very quickly, it does not differentiate well between similar textures. New modern methods concentrate on a more precise texture measure using a variety of features.

Region-based segmentation methods can be categorized into three general methods: merging, where the image is divided into extremely small regions (often each pixel is its own region) and then combined to form larger regions; splitting, where the whole image is divided into smaller regions; or split and merge, where the image is divided into regions, and the regions are split and merged until a condition is met [3]. The Split and Merge procedure (SMP) is a popular region based segmentation, that attempts to overcome problems of only merging or splitting. Splitting an entire image often leads to the summarization of features leaving out important pixel level details making it difficult to find a precise boundary. Merging very small regions is problematic because at a local level, it is often difficult to determine which texture a pixel belongs to.

The split and merge algorithm starts by an arbitrary segmentation where every pixel must belong to exactly one region. Then each region is split or merged until two conditions are satisfied. First, the maximum feature variance within a region is below a certain threshold. Second, there are no neighboring regions where the combined maximum variance in features is below the threshold. For example, if the feature used is luminance, there will be no regions where the difference of luminance between any two pixels in a region exceed a given threshold. There also does not exist two neighboring regions that contain two pixels that exceed the given luminance difference threshold when merged.
Boundary-based methods suffer from a similar problem as region merging-based methods — it is very difficult to find a large boundary due to local noises in image features. Edge detection methods often have problems differentiating between texture edges within a region and edges between two regions. Many boundary-based methods are used in combination with region growing [42] [41], or instead of edge detection the image is reduced to a graph division problem [37]. A graph is constructed where every pixel in the image is a node in the graph and edges are connected to the nodes of neighboring pixels. Edge weights are some measure of similarity between the two pixels, which could be as simple as the difference in color or intensity. The graph is then divided into an arbitrary number of groups of connected nodes, where each group of connected nodes represents a region in the image. Each division of the graph must create a group of nodes that disconnects it from another group of connected nodes. The graph is divided to minimize the average cost of the edge weights present in the graph. The graph can be divided into an arbitrary number of regions by recursively bifurcating nodes in the graph.

Although most methods fall into region or boundary-based segmentation, there are many methods which cannot be easily categorized. Almost all segmentation methods attempt an exact segmentation where every pixel is exclusively in a certain region. Since this is extremely difficult in a broad domain it is argued that this amount of precision cannot be achieved nor is it needed for certain applications of image segmentation [34].

Tauben, Li and Drew pointed out that for the application of image retrieval, precise image segmentation is not an attainable goal and that coarse segmentation is more useful. In traditional image segmentation, the following three properties are followed:

1. All pixels in a region are connected.
2. Regions are disjoint from each other.
3. The segmentation is complete, where every pixel is included in a region.

In this coarse segmentation, instead of regions, pixels are grouped into locales. A Locale consists of a group of one or more tiles, where a tile is a square block in the image. In addition to the group of tiles it represents, called an envelope, it also has descriptors such as the centroid, mass, eccentricity and other texture and color descriptors. This coarse segmentation method, called localization which is shown in Figure 2.1, has the following three properties:
1. A locale doesn’t have to be connected.

2. Locales can be non-disjoint; a tile may belong to more than locale.

3. The segmentation can be non-complete, not all pixels are in a locale.

![Figure 2.1: Three locales in an image](image)

Localization has proven to be a good segmentation method for image and video retrieval [13]. Its application is somewhat limited for such applications as video segmentation as an exact segmentation is required. Although locales can be used for image compression, where pixels not in any locale are merged into one region, it is difficult to achieve increased compression savings without precise segmentation.

### 2.2 Video segmentation

Automatic video object segmentation is the process of segmenting moving objects from a series of images, called frames. Changes from adjacent frames in a scene can be attributed to a combination of one or more of:

1. Motion of an object.
2. Motion of the camera.
3. Change in lighting.
CHAPTER 2. SURVEY OF RELEVANT WORK

Video object segmentation is the process of segmenting pixels that have similar motion. Motion is often difficult to detect because of occlusion and the aperture problem. Occlusion occurs when moving objects cover and reveal other objects in the scene. Occlusion causes pixels to seem to disappear when covered with another object, or new pixels to appear when they are revealed from behind another object. The aperture problem occurs when it is impossible to tell from the image which direction the object is moving. Since the image is simply a 2D projection of a 3D scene, there could be many motions which would result in the same 2D projection. An example of the aperture problem is a diagonally striped pole which is rotating along the longer axis; this is visually the same as the pole moving in the direction of the shorter axis. Many segmentation methods attempt to overcome these problems, as well as the problem of noise hindering the ability to accurately estimate the motion.

Early video segmentation methods concentrated on first finding the pixels where motion occurs, instead of finding each separate object that moves in the video [16]. The first step to find the motion is to remove all background pixels, so that the remaining pixels can be analyzed. Background or foreground classification can be achieved by finding the temporal variance defined as:

\[ V_t(x, y) = \frac{\sum_{z=0}^{t-1}(P(x, y, z) - \text{Mean}_t(x, y))^2}{t}, \]

where \( P(x, y, z) \) is defined as the pixel value at spatial coordinates \( x \) and \( y \), at frame \( z \). \( \text{Mean}_t(x, y) \) is defined as the average value at coordinates \( x \) and \( y \) over time \( t \).

A high value would indicate movement, since the pixel changes greatly in a short amount of time, while a low value would indicate it is in the background since it does not change much over time. A threshold can be used to indicate whether pixels are part of the background or foreground. A problem with this method is that it does not take into account camera movement, which would cause even background pixels to have a high temporal variance.

Most object-based modern motion segmentation methods fall into one of two categories [3]. The multiple motion segmentation methods attempt to find all objects in the video simultaneously, which makes it difficult to determine how many and where the objects are in the video. The other category of segmentation, called dominant motion segmentation attempts to sequentially find one object at a time, this makes it difficult for smaller objects which have smaller influence over the video.
Many different variations of dominant motion estimation have been used to perform video segmentation. The basic method is to find a single motion that best describes the motion of the entire frame. Once this is done, the image is divided into two regions which represent a region that is well described by this motion, and a region which is not. This process is repeated on the region that was not well described by that motion. The process stops when all pixels have been assigned a motion.

Multiple motion segmentation methods generally consists of 3 steps:

1. Estimate number of independent motions.
2. Estimate model parameters for each motion.
3. Determine support of each model (segmentation labels on the image).

If the number of motions and model parameters are known, then the pixels in the image can be assigned to the model that most closely resembles its motion. If the number of motions is known and the image is already labeled, the model parameters can be estimated based on its members, for example the mean or median motion of its members. Without knowledge of either the model parameters or the segmentation labels it is difficult to find them both, which is often the case. Many approaches have been developed to solve this problem, such as maximum likelihood segmentation or maximum a posteriori probability segmentation.

2.3 Block-Based Motion Estimation

MPEG based compression relies on block-based motion estimation algorithms (BMA) that encode the motion of blocks between different frames of a video [3] [25]. In BMA the block is compared to a block in a search area in the previous frame, and a cost is associated with each movement in the previous frame. The cost is a measure of difference between the two blocks, typically the mean squared error (MSE) is used, but also other measures can be used, such as the mean average error (MAE) or the sum of absolute difference (SAD). The mean squared error of a block of size $N$ with coordinates of the lower left corner $(x, y)$, given motion vector $(i, j)$ is defined as:

$$MSE(i, j) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} (C(x + k, y + l) - R(x + i + k, y + j + l))^2,$$  \hspace{1cm} (2.2)
where \( C(x, y) \) is the pixel value of the current frame at location \((x, y)\) and \( R(x, y) \) is the pixel value of the reference frame at location \((x, y)\).

The motion estimation (ME) can be thought of as a minimization problem. To find the best motion, the motion with the minimum MSE is found. This can be accomplished by using a full search algorithm (FSA). A FSA finds the MSE of every possible motion in the search window. This gives the global minimum MSE, but has the severe drawback of being computationally very expensive. Many fast search algorithms which decrease the search space such as the diamond search, three step search and logarithmic search find a motion much faster [18].

All these fast search algorithms rely on the assumption that the matching error decreases monotonically as the searched point moves closer to the global minimum. This assumption, while able to reduce the search space significantly, does not hold for many circumstances [9]. The assumption is based on a simple gradient texture which in natural video scenes doesn’t often occur. Significant research has been done on finding a better measure of error as well as strategies that reduce the search space while still providing a good motion estimation.

### 2.4 MPEG-4

MPEG-4 is a modern visual standard designed for object based coding [25]. MPEG-4 is designed to not only support traditional frame based video encoding, but also object based coding. It features the ability to support scalable temporal and spatial playback and resilient error coding techniques. The ability to encode synthetic 3D objects with texture is also included in this standard.

In traditional video encoding, each individual frame is encoded together. MPEG-4 supports the notion of *video objects* (VOs) which correspond to real world objects in the scene. Each VO is encoded separately and does not depend on other VO. In each frame, each VO is described as a *video object plane* (VOP). The idea of VOPs is an important factor in MPEG-4. The VOP contains not only the color and motion information as in traditional video coding; it must also contain shape information. Two types of shape coding are possible in MPEG-4 depending on the type of object. Transparent objects are coded using an 8 bit alpha plane which describes the level of transparency in each pixel. Opaque objects are coded using a binary alpha plane. The alpha plane is tightly bounded by a rectangle where each side has a length that is a multiple of 16. The plane is then partitioned into \( 16 \times 16 \)
pixel binary alpha blocks (BABs).

The color information for each VOP is compressed using the Discrete Cosine Transform (DCT) like most other video encoding schemes. DCT is a spatial-frequency transformation performed on a sequence of pixels. A 1D DCT is defined as:

\[
F(u) = \frac{C(u)}{2} \sum_{i=0}^{n-1} \cos \left( \frac{(2i + 1)u\pi}{2n} \right) f(i),
\]

where \(F\) is the DCT value, \(f\) is the pixel value, \(n\) is this size and

\[
C(u) = \begin{cases} \frac{\sqrt{2}}{2}, & \text{if } u = 0 \\ 1, & \text{otherwise}. \end{cases}
\]

In MPEG-4 the VOP is divided into 8 x 8 square blocks. A 2D DCT is performed on the pixels, which is defined as:

\[
F(u, v) = \frac{C(u)C(v)}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} \cos \left( \frac{(2i + 1)u\pi}{16} \right) \cos \left( \frac{(2j + 1)v\pi}{16} \right) f(i, j),
\]

The encoding method provides no compression for random data. The advantage of DCT is that in natural images, there are very few high frequency values [2]. The compression is achieved from having most of the high frequency values near 0, so when entropy encoding is applied fewer bits are needed.

The top left value in the resulting matrix \(F(u, v)\) is referred to as the DC value, while the rest of the values are referred to as the AC values. The DC value represents the average value of the block. The AC values represent the amplitude of sine waves, where values closer to the DC value are the lower frequency values and the values farther away are the high frequency values.

Unlike traditional video compression techniques, non-rectangular groups of pixels must be encoded as well. This leads to two possibilities: either the shape can be extended into a rectangular block of pixels or the DCT must be able to accommodate arbitrarily shaped groups of pixels [25]. Simple padding schemes like the repetition of the border pixel often lead to a loss of compression. If the padding which has no texture and a highly textured group of pixels are in a block together, the resulting DCT transformation will contain many high frequency components making it difficult to compress. It is also possible to assign values to the missing pixels in such a way as to minimize DCT values but it would take a large number of computations to even prevent data expansion [30].
Shape Adaptive DCT (SA-DCT) can be used to encode arbitrarily shaped groups of pixels as shown in Figure 2.2. SA-DCT first shifts all pixels in the horizontal direction so that all the spaces lie on the right side, and then performs 1D DCT on each column. Then from the result, all values are shifted in the vertical direction so that all the spaces lie on the bottom, and 1D DCT is performed on each row. This allows the compression of DCT without having to pad the block.

Figure 2.2: Steps of SA-DCT

In conventional video encoding, the amount of compression, and in turn video quality loss, depends on the quantization matrix. The quantization matrix is a set of values which the DCT values are divided by, thereby reducing the range the DCT values can have.

A major problem with SA-DCT is that the compression rate depends on the shape of the region encoded; this is referred to as the mean weighting defect [15]. This is undesirable, since the resulting VOP quality is uncontrollable. The mean weighting defect can be corrected by using DC separation. DC separation is achieved by setting the average value to 0.
Chapter 3

A New Energy Function for Image Segmentation

An improved image localization method for the purpose of image segmentation is described in this section. Originally, image localization is designed as a coarse segmentation. In localization, statistical measurements are important but not exact pixel membership [19]. Therefore, in image localization a pixel could belong to no regions, exactly one region, or more than one region. The localization of an image is shown in Figure 2.1. Image localization gathers initial statistics for region growing using tiles. A tile is a square image block, in which we group the underlying pixels based on features. Each tile contains color and some geometric statistics such as the centroid and eccentricity. No pixel location is preserved, although the tile location it belongs to is known. Since the DCT block size is identical to tile size, we can use DCT as a texture descriptor for localization.

The purpose of the segmentation is for image and video compression, so the exact pixel location must be kept and pixels must be grouped into contiguous regions. The first step of the image segmentation scheme is to create sub-blocks, this has the advantage of initializing the sub-blocks to reliable statistics. This step is done because the color or texture of any specific pixel is not consistent for region growing, and leads to seeding problems. Hence, the new method is a cross between the pixel-level information obtained from image segmentation, and localization where region growing is based on statistics of pre-established blocks.
CHAPTER 3. A NEW ENERGY FUNCTION FOR IMAGE SEGMENTATION

3.1 The New Energy Function

The first step in sub-block generation is to partition the image into 8 x 8 blocks of pixels in an image. The central assumption for DCT-based image compression is that natural objects contain mostly low frequencies, and so the high frequencies can be efficiently encoded using 0-runs [18] [2]. Building on this assumption and the observation that objects in natural images typically have low DCT values, an effective way to divide blocks can be devised. Block segmentation into regions with similar texture patterns can be achieved by minimizing the combined affinity for high frequencies of the resulting sub-blocks. These sub-blocks should also correspond to separate objects. A novel DCT-based energy function is created and is defined as:

\[ E_{DCT} = \sum_{i=1}^{w} \sum_{j=1}^{h} e^{[(\frac{1}{wh})^2 - 1]} DCT(i-1, j-1), i + j \neq 2, \]  

(3.1)

where \( E_{DCT} \) is the energy of the block, \( DCT(i,j) \) is the absolute value of the \((i,j)\)th DCT component, unless it is the DC component which returns 0, \( w \) and \( h \) is the width and height of the block, respectively. The energy equation maps the texture from a multi-dimensional frequency space into a one dimensional energy space. This DCT energy functional exponentially weighs higher frequencies in DCT since we expect the highest frequencies to be caused by a mixture of objects. The DC value is treated separately since it is color dependant and does not influence the texture. Since the energy function is used to determine how likely all the pixels in a block are all within one region, the average color doesn’t influence and hence is not considered here.

Visually similar textures do not always lead to similar DCT values. The DCT components cannot be compared individually, due to differences between the texture and AC phases. However, their frequencies concentrations are similar as shown in Figure 3.1, and so \( E_{DCT} \) provides a good measure to compare.
CHAPTER 3. A NEW ENERGY FUNCTION FOR IMAGE SEGMENTATION

Figure 3.1: The absolute sum of groups of similar frequency components are similar, as illustrated by the similar absolute sums in the two blocks.
It is known that the DCT representation of natural images has relatively few high frequency components [2]. Typically, images are composed of many natural regions, each with different textures. These observations can lead to some reasonable assumptions, such as most energy values of blocks within a region will be low.

A highly textured block of pixels produces a high energy value from equation 3.1, since a highly textured block of pixels would produce many high frequency values; conversely, a low textured block of pixels like a block of homogeneously colored pixels produces very few high frequency DCT values and results in a low energy value. Two groups of pixels can be compared by calculating their energy value and if they are dissimilar then they belong to different texture regions. Although one cannot say that two similar energy values are always similar in texture, since we are mapping from a multi-dimensional space to one, if we make use of the color of each block as well, we can make an assumption that two groups of pixels spatially close in an image as well as similar in color and energy should be in the same region. A problem common to all weak segmentation techniques is where if two regions have similar features and are adjacent, it is difficult to divide the two regions properly. A problem of this nature is extremely rare using the energy feature; even if two regions of similar color have different textures, but similar in the complexity of the texture pattern, the border can be found between them, since two textures in a single block will cause high energy.

Figure 3.2: Simple and complex texture blocks produce low and high energy, respectively.

An important feature of DCT to note is that if the group of pixels being compared is comprised of two distinct textures it will produce many high frequency values; for example, if the group of pixels being processed is an edge between a black group of pixels and a white group of pixels, the one edge will need many sine waves to represent it, resulting in
numerous high frequency DCT components. Figure 3.3 illustrates this. The energy value expected from an edge between any two distinct textures is expected to be significantly higher than either texture processed separately as shown in Figure 3.4. An algorithm to find the division between textures in an image can take advantage of this property by comparing energy values.

Figure 3.3: consistent regions produce low energy, while blocks between regions produce high energy

Figure 3.4: A 256 x 256 pixel image. The gray level indicates the relative magnitude of energy in (b). Energy is low in homogeneous texture regions, while high between them.

Finding the best block subdivision is a minimization problem over $E_{DCT}$. Stated informally, we are looking for any number of sub-blocks for which the combined $E_{DCT}$ measure is the lowest over all configurations of the block. To find the minimum combined energy of a block $b$, defined as:
\[ E_{DCT_{\text{total}}}(b) = \min \left( \sum_{r \in R} E_{DCT}(r) \right), \text{ for all possible sets of regions } R \in R_T, \quad (3.2) \]

where \( R \) is a set of regions in a block, and \( R_T \) is the set of all possible sets of regions in a block.

A naive implementation of this minimization would require a combinatorial number of calculations which is cost prohibitive. Instead, we attempt to find a close first estimate of split locations.

Once the split is found and refined, the sub-blocks generated are used as the basis for the regions in the image segmentation - the sub-blocks are the basic elements used in a region growing algorithm similar to the generation of locales [19]. An overview of the region creation algorithm is shown in Figure 3.5.

On a typical 512 \times 512 pixel resolution image, the segmentation algorithm takes approximately 2 minutes on a computer with an AMD 64 3200+ processor. The vast majority of computation time comes from the two feature extraction phases, which take approximately 80% of the processing time. The energy block minimization takes approximately 10% of the computation time. The region growing takes approximately 5% of the computation time, with the last 5% taken by the remainder of the steps. The computation time grows linearly relative to the area of the image.

\[ \text{Image} \]
\[ \uparrow \]
\[ \text{Feature extraction 8x8 block size} \rightarrow \text{Edge Detection} \rightarrow \text{Shift Edges} \rightarrow \text{Create binary edge mask} \]
\[ \text{Feature extraction 4x4 block size} \rightarrow \text{Edge Detection} \rightarrow \text{Shift Edges} \rightarrow \text{Threshold edge mask} \]
\[ \uparrow \]
\[ \text{Gaussian blur} \rightarrow \text{Edge thinning} \rightarrow \text{Adaptive threshold} \rightarrow \text{Edge extension} \rightarrow \text{Energy block minimization} \rightarrow \text{Region Growing} \]

Figure 3.5: Overview of the image segmentation process
3.2 Sub-Block Creation

Image segmentation using the energy feature can be broken into two major steps: the first step is to create sub-blocks by using the region borders detected, the second step is to join these sub-blocks into regions. To find the boundaries of the regions, a single texture feature is used. This feature is calculated using energy function 3.1 which characterizes the complexity of the texture. The energy function with a block size of $8 \times 8$ is calculated for every pixel in the image. The value of the energy function is then assigned to the center pixel and a feature map is created.

When the energy calculations are done over an entire image, one would expect regions with similar texture to produce similar energy values. Between regions there will be bands of high energy, where two or more textures are present in the block as shown in Figure 3.4. The band of energy has a radius of up to $2(n - 1)$, in size, where $n$ is the size of the block. This band is caused when at least a part of one row or column of pixels is present in the window from a region, and the rest of the window is from another region. The energy strength is high when the block enters a new region, and slowly increases as the center of the block approaches the border of the two regions. This increase is relatively small, so the energy bands in effect have very similar energy values throughout the band.

The initial boundaries are found using Sobel edge detection on the feature map. The Sobel edge detection is done using two masks on the image as in Figure 3.6.

![Sobel filters for edge detection](image)

Using the values of the two edge filters, the magnitude of the edge present as well as the directionality of the edge can be computed. The edge magnitude is defined as:

$$E = \sqrt{E_x^2 + E_y^2},$$  

(3.3)

where $E_x$ and $E_y$ are the edge filter values in the horizontal and vertical axis, respectively. The orientation $\theta$ can be calculated as:
CHAPTER 3. A NEW ENERGY FUNCTION FOR IMAGE SEGMENTATION

\[ \theta = \arctan \frac{E_y}{E_x} \]  

The directionality and magnitude of the edge will be used to find the center of the energy bands and in turn provide an initial block division. The angle calculated is likely to be inaccurate due to many problems. These inaccuracies are due to the noise in the image caused by camera smoothing or color bleeding, as well as from the limited resolution of the image (digitization).

To compensate for these problems, an angle smoothing algorithm is used. Weak edges are first eliminated using a low threshold. For each remaining pixel with some edge magnitude, connected pixels up to 2 pixels away with similar edge magnitude are found. Two line segments of length 2 on opposite sides of the current is attempted to be found. Only available edge values are found, so if shorter line segments or no line at all is found for a side, these edges are not included in the calculation. A total of up to 5 pixels are found, and the median angle is used to smooth the angle calculated for each pixel. This method of angle smoothing is used to more accurately find the line orientation instead of a simple average over a small area, since connected edges with similar magnitude are likely to be part of the same line.

Once the edges have been smoothed, the borders of the regions need to be found. The shared border of multiple regions lie in the middle of the energy band, which has variable radius depending on the angle of the boundary. The radius ranges between \( \sqrt{2(n-1)^2} \) when the angle is 45 degrees, to \( n-1 \) when the angle is 0. The edges are shifted for angles when \( E_x > E_y \):

\[
S_x = n - 1 \quad (3.5)
\]

\[
S_y = (n - 1) \tan \theta \quad (3.6)
\]

otherwise

\[
S_x = \frac{n - 1}{\tan \theta} \quad (3.7)
\]

\[
S_y = n - 1, \quad (3.8)
\]

where \( S_x \) is the shift in the horizontal direction and \( S_y \) is the shift in the vertical direction. Any pixels shifted out of the image boundary due to rounding error are placed in the closest pixel inside the image.
All the pixels in the edge map are shifted based on this calculation. This allows for an accurate line for block division, since the shifting is like an edge accumulation array. In effect, each pixel votes for where the line segment should be. The strength of the vote is the magnitude of the edge value. Even a large inaccurate edge will not have a large magnitude after shifting, as most edges will be accurate, and when they shift to the correct position, will have a edge magnitude larger than a single edge value.

Even after angle smoothing the edge direction is still not accurate, resulting in some pixels shifting one pixel off. Therefore, the resulting combined line therefore will have small holes as well as some pixels that are double the intensity of neighboring edge pixels due to noise and digitization. A continuous edge is desirable in locating the border, so a Gaussian blur is applied to the edges to smooth out the holes and high intensity edges. The final result is an edge map showing the boundaries between the regions, as well as edges from within a region as the texture fluctuates within the region.

An effective way to reduce error from shifting while reducing the location dependency of the energy function is to use a multi-resolution approach. The entire procedure is repeated with a smaller block of size $4 \times 4$. The edge map of the larger block is then used to remove false edges from the smaller block size edge map, while lowering the distance the edges must shift. This process is achieved by converting the edge map with the $8 \times 8$ block size into a binary mask. This binary mask is then used to eliminate edges found in the edge map using block size $4 \times 4$. Edges that do not exist when the $8 \times 8$ block size is used are thrown away when the $4 \times 4$ block size calculation is done.

Edge thinning is used to provide a precise segmentation. Non-maximum edge suppression is used to thin the edges. The non-maximum suppression of edges removes the edges which have a stronger edge neighbor in a direction toward or opposite to their own direction. This thins thick lines, and preserves the line where it has the highest edge value.

The next step is to throw out edges which do not represent region boundaries. Region boundaries have many relatively high and similar edge values. Edge values from within a texture most likely are more dispersed in value and are more random and varied spatially. We can take advantage of this observation by using a threshold which attempts to keep clusters of similar edge values while disregarding the rest. A sliding window is used to try to adaptively find the local thresholds.

A sliding window of size $16 \times 16$ is moved $8$ pixels at a time around the whole image, so that all pixels not near the border of the image are included in four windows. In each
window, a local edge histogram is constructed. A threshold is chosen by the contents of the edge histogram. Starting with a minimum threshold, the total of every group of three contiguous bins is calculated. The calculation proceeds by increasing the bin values used until a minimum total is reached, or the maximum bin number has been reached. If no three contiguous bins reach the minimum threshold, then a high threshold is used. If a set of three bins are found, the threshold is set to below the lowest bin in the set. This threshold is designed to take out noise which typically is more varied in value and spatial location. This procedure is shown in Figure 3.7.

![Threshold](threshold.png)

(a) Edge block  
(b) Edge Histogram

Figure 3.7: The threshold determination of an 8 x 8 block is demonstrated. A block with a real edge as well as an edge that should be threshold out is shown.

The minimum threshold of each of the windows that the pixel is involved in is used. All groups of pixels which fall below this minimum are set to an edge value of 0. After this is done, all groups of edges which do not have a minimum connected size are also thrown out using a morphological open image operation.

The final step is to extend the boundaries to the borders of the image as well as join edge segments. Since the energy value is assigned to the center of the window, all pixels within n/2 of the edge of the image do not have borders, as they had no energy assigned to them. Edges which are close to the border, within n/2 + 2 are assumed to extend to the border, if the edge angle is closer to parallel to the nearest border than perpendicular to it. The edge is extended in a straight line perpendicular to the angle toward the image boundary to complete the edge.

Edges within the edge map are analyzed and the endpoints of each group of connected pixels are found. Endpoints are found by first finding connected edges, which is done by labeling pixels which are connected. Then for each group, an arbitrary pixel is chosen from
a line segment. From that pixel, a path is walked for every branch. The end of each path is labeled an endpoint. A small area perpendicular to the angle of the endpoint away from the line is searched. If another edge that is not currently connected to the current edge intersects this area, then a line perpendicular to the edge angle is drawn connecting the endpoint with the other edge.

The image is then partitioned into $8 \times 8$ blocks and divided into sub-blocks using the edges found in the previous steps. The sub-blocks are now initialized, but need to be refined due to inaccuracies from the process. When two unique textures are part of a sub-block, many high frequencies will be present. If each sub-block has only one texture in it, the combined high frequency components are less than the high frequency components of the whole block.

To refine the cut in the block, ideally the boundary can be moved pixel by pixel and the combined energy recalculated to find the minimum combined energy using equation 3.1. This would be computationally expensive so instead an approximation is used. Since the errors are small, the real edge should be close by and assumed within one pixel of the current boundary. The boundary is iteratively shifted using line segments half the size of the last iteration. The procedure is shown in Figure 3.8.
Procedure Sub-Block Minimization

For each block:
  For each border between two regions:
  
  Find border pixels $P$
  Sort($P$) starting and ending from an arbitrary endpoint
  Regions = region information, the pixel labeling
  Let $n$ be the smallest integer, where the size of $P$ rounded up = $2^n$
  for $i = 0$ to $n$
  
  Divide $P$ into $2^i$ equal sized groups of connected pixels
  For each group of pixels $P_{seg}$:
  
  energyBase = current total block energy
  energy1 = Move border $P_{seg}$ into region 1
  energy2 = Move border $P_{seg}$ into region 2
  if (energy1 < energy2)
  
  if (energy1 < energyBase)
    set $P_{seg}$ toward region 1
  
  else
  
  if (energy2 < energyBase)
    set $P_{seg}$ toward region 2
  
  
Figure 3.8: Procedure for energy minimization for block sub-division
3.3 Region Growing

The next step in image segmentation after the sub-blocks have been created is to create regions. Regions are created using a pyramid linking scheme, similar to the localization procedure described in [19]. A $4 \times 4$ overlapped pyramid structure is used, where each parent competes for child nodes. The first step of region growing is initially to label each sub-block as its own region. Two iterations are used in the region growing, the first iteration merges together regions which consist of whole blocks exclusively, while the second iteration merges sub-blocks into the regions. The larger regions are merged first because the statistics associated with larger blocks are more reliable, since they encompass a larger area.

Competition happens between every two consecutive levels of the pyramid. Each region in a level of the pyramid is called a node, where between any two consecutive levels of the pyramid, the bottom level nodes are labeled child nodes and the top level nodes are labeled parent nodes. At the lowest level of the pyramid, the regions are initialized to the sub-blocks generated. The next level of the pyramid attempts to create larger regions by linking one or more child nodes into a parent node. Once the parent nodes have finished competing with other parent nodes for child nodes, the parent level of the pyramid is finalized. For the next level of the pyramid, the parent nodes are now labeled as child nodes, and the competition to merge child nodes begins anew. This process is repeated until the parent nodes compete for child nodes across the whole image.

Each region contains not only a list of sub-blocks that it encompasses, but other statistics to help in the competition process. The average energy, color, centroid, mass and eccentricity of each region are calculated. The equations for the mass $M$, centroid $C$ and eccentricity $Ecc$ are:

\[ M(R) = \text{Number of pixels in the region } R \]  \hspace{1cm} (3.9)

\[ C(R) = \frac{\sum_{i=1}^{M(R)} P_i}{M(R)} \text{, where } P_i \text{ is the coordinate of pixel } i \]  \hspace{1cm} (3.10)

\[ Ecc(R) = \frac{\sum_{i=1}^{M(R)} \| P_i - C(R) \|}{M(R)} \]  \hspace{1cm} (3.11)

Eccentricity is a measure of spatial variance from the centroid of the region. Eccentricity is used to ensure that a region is clumped together in a compact region and not dispersed.
throughout the whole image. It also favors the merging of smaller regions, making it unlikely that very small regions will survive the competition process.

Competition for every level of the pyramid begins by using a 2 $\times$ 2 non-overlapped window: in the first level up to four blocks are involved. The amount of blocks is variable since only whole blocks are included at this point. A child node is merged into a parent node if the texture energy and color are similar, as well as if the combined eccentricity is under a threshold based on mass of the region. If no such parent node exists, a new parent node is created. Each level of the pyramid essentially merges regions from an area four times larger than the previous level. Once all children are initially merged, the competition phase begins.

While in the first phase a child node could only merge to a parent originating from one of the child nodes in a 2 $\times$ 2 node area, in the second phase it may merge in a larger area. Each child node searches for the best parent node in this area, where 4 parents compete for the children. If a parent within the texture, color and eccentricity thresholds is found that has a lower Euclidian distance between their centroids, it is linked to that parent instead. Child nodes are shifted until no changes are possible.

The statistics are updated with each child movement as follows:

\[
M(R_T) = M(R_1) + M(R_2)
\]

\[
C(R_T) = \frac{C(R_1)M(R_1) + C(R_2)M(R_2)}{M(R_T)}
\]

\[
Ecc(R_T) = \frac{(Ecc(R_1) + C(R_1)^2)M(R_1) + (Ecc(R_2) + C(R_2)^2)M(R_2)}{M(R_T)} - C(R_T)^2
\]

Since each child can only move to parent nodes that are closer, the process has a finite number of iterations. After the region creation has been completed, sub-blocks are added, with sub-blocks merging to the region with the lowest Euclidian distance and within the color, texture and eccentricity thresholds. If no such region exists, a new region is created. Very small regions are merged into the closest region with the most similar energy and color. The linking process is shown in Figure 3.9.
Procedure Pyramid linking

Initialize children nodes to every block that was not divided
Initialize parent nodes to empty

pyramid width = image width / blocksize
pyramid height = image height / blocksize

While (pyramid width > 1) and (pyramid height > 1) {
    For i = 1 to pyramid width; i = i + 2
        For j = 1 to pyramid height; j = j + 2
            For a = 0 to 1
                For b = 0 to 1
                {
                    Try to match each region in child node\((\text{i*2}+\text{a})[(\text{j*2}+\text{b})\) with every parent node region \([\text{i}][\text{j}]\)
                    If match found
                        add to the parent node region it matches to
                    else
                        create new parent node region at \((\text{i},\text{j})\)
                }
            For i = 1 to pyramid width
                For j = 1 to pyramid height
                    {
                        For every region in this child node, look at all the regions in the parent nodes from \([\text{i/2-1} to \text{i/2+1}]\)[\text{j/2-1} to \text{j/2+1}]
                        If a parent matches and is closer in centroid distance merge to that one instead
                    }
    parent nodes = child nodes
    pyramid width = pyramid width / 2
    pyramid height = pyramid height / 2
}

For each sub-block that was divided, merge into the region that matches its properties, and has closest centroid

Figure 3.9: Procedure for Pyramid linking
Chapter 4

Extension to Video Object Segmentation

The multi-motion approach is taken to video segmentation. Since the number of regions as well as their support is known from the image segmentation, the motion parameters for each object can be found. Without knowing the number of objects as well as the support, it would be impossible to find the motion parameters using a multi-model approach. Traditionally, the motion of each block can be found using a fast motion approximation, like the diamond search that minimizes the MSE \([44]\). By comparing the current block and the block in various spatial patterns in the previous frame, a local minimum can be found by finding the location in the previous frame that minimizes MSE. If the global minimum MSE is found, it does not guarantee the best encoding. Even with a FSA all traditional cost functions, including MSE, SAD, and MAE does not imply that the maximum compression is achieved. The errors are encoded using DCT, which does not correspond to having a minimum cost function like MSE, SAD, or MAE. Only cost functions which use the frequency domain can minimize the space the error encoding needs. A DCT based cost function is well suited for this task.

If a search technique finds the global minimum it may not be the true movement of the block. The true movement of a block is more desirable than a movement that simply minimizes MSE because blocks in the same region will have a more predictable motion.
4.1 3D DCT Motion Energy

Motions can be more accurately predicted using multiple frames, since object motions are typically smooth over a short time period. It is unlikely that an object would rapidly switch directions over a few frames, which represent just a fraction of a second. According to Newton’s first law of motion, an object will stay at the same speed and direction unless acted upon by an unbalanced force. Since the objects do not get moved by an unbalanced force, except gravity and wind resistance, the motion will not change often. Moreover, gravity and wind resistance provide small consistent changes that do not change motion significantly.

Many 2D motions can be approximated by finding a linear motion which best fits temporally nearby frames. Similar to images, the frequency domain in time is a more desirable domain to work with. Slow and small changes in the object are more likely than a large change. Instead of using MSE as the measure for the accuracy, a novel motion detection based on 3D DCT is used, using an extension of equation 3.1,

\[
E_{3DCT}(MV) = \sum_{i=1}^{w} \sum_{j=1}^{h} \sum_{k=-t/2}^{t/2} e^{\left[\frac{2\pi^2 z^2}{w^2} + \frac{2\pi^2 z^2}{h^2} - \frac{\pi^2}{t^2}\right]} DCT(i + MV_x(k) - 1, j + MV_y(k) - 1, k),
\]

where \(w\) is the width, \(h\) is the height, \(t\) is the number of frames, \(MV_x(k)\) and \(MV_y(k)\) is the motion vector of frame \(k\) in the horizontal and vertical direction, respectively, and \(DCT(i + MV_x(k), j + MV_y(k), k)\) is the 3D DCT given the motion vector \(MV\), defined as:

\[
DCT(i + MV_x(k), j + MV_y(k), k) = \frac{C(i)C(j)C(k)}{8} \sum_{l=0}^{w-1} \sum_{m=0}^{h-1} \sum_{n=0}^{t-1} \cos \frac{i(2l + 1)\pi}{2w} \cos \frac{j(2m + 1)\pi}{2h} \cos \frac{k(2n + 1)\pi}{2t} f(l + MV_x(n), m + MV_y(n), n),
\]

where \(f(l + MV_x(n), m + MV_y(n), n)\) is the pixel value in frame \(n\), at location \((l+MV_x(n), m+MV_y(n), n)\).

A low \(E_{3DCT}\) energy would mean that the pixels with the motion predicted had very little change or that the change was gradual, such as due to lighting changes. Slow changes, like changing illumination, would have very little effect on the 3D energy if it was approximately equal across the block over time. A slow constant change from lighting would not cause the...
4.2 Motion Detection

4.2.1 Diamond Motion Energy Search

Motion estimation begins by first finding the best predicted MV for non-border blocks inside a region. The inner blocks of a region are more stable and easier to find the movement for than the boundary blocks. To find the optimal motion for each block the diamond search is used. The search pattern is a diamond shape, where there are 8 possible movements, with a 45 degree square of length 2 pattern, as show in Figure 4.1. Each of the motions is over 4 frames, so in reality each frame moves just 0.25 pixels. To calculate the energy at each point, equation 4.1 is used with 4 frames. The energy for each of the 8 possible motions is calculated as well as no motion. If the minimum energy is one of the 8 possible motions, then the calculation is repeated with the diamond centered at the minimum motion. This is repeated until the minimum energy can not decrease with any of the 8 possible motions.

The second phase of the diamond search is to find the best motion by finding the minimum motion energy in a 4-connected neighborhood. In many cases the assumption holds true that the energy decreases as it reaches the global minimum as in Figure 4.2.
CHAPTER 4. EXTENSION TO VIDEO OBJECT SEGMENTATION

Figure 4.1: We assume a monotonically decreasing energy toward the minimum energy. In step one (b), the minimum energy is found toward the right search area. Again the best energy found is toward the right in the second step (c). In (d) the third step, the area is searched, but no better energy is found. The search pattern is adjusted and the minimum energy is found in (e).

Figure 4.2: Motion energy space of a block where the true motion is to the right side. The pixel luminance reflects the value of the energy.
4.2.2 Graph Cuts

Often when objects move in natural video, all parts of the object move in a similar direction. To find the best motion vector for all the sub-blocks in a region, they all must be considered. This can be formulated as an energy minimization problem, where the data and the smoothness can contribute to the energy value.

\[ \text{Energy} = \text{Energy}_{\text{data}} + \text{Energy}_{\text{prior}} \]  

(4.3)

This represents the Gibbs form of the MAP-MRF hypothesis, where \(\text{Energy}_{\text{data}}\) is the data term representing the cost function and \(\text{Energy}_{\text{prior}}\) is the smoothness term [10].

The problem can be solved using graph cuts [5] to estimate the best motion set for all the sub-blocks in the region. Since energy minimization in this case is an NP hard problem an exhaustive search is not practical. Instead many people try to find an approximation of the global minimum, which may be the local minimum. The local minimum is defined as the lowest value that can be attained within one "move", where a move is a change of state. Finding the movement vectors of the sub-blocks in a region can be formulated into a similar energy problem. There are two competing factors that contribute to what the motion vector should be, the energy found by the 3D DCT energy function (the noisy data), and what motion vectors its neighbors are (maintaining smoothness). So the global energy of a region that we seek to minimize is

\[ E_{\text{motion}}(M) = E_{\text{smooth}}(M) + E_{\text{data}}(M). \]  

(4.4)

where \(E_{\text{data}}\) is the energy from the discrepancy between the set of motion vectors \(M\) and the data, and \(E_{\text{smooth}}\) is the energy from the discrepancy between neighboring motion vectors.

To use graph cuts, a set of labels \(L\) must be defined. The set of possible labels \(L\) corresponds to all possible MVs in the search window. For example, a label can be \((MV_x, MV_y)\), where the 2D motion is represented as a label. So with a minimum movement of 1 pixel and search window of size \(9 \times 9\), there would be 81 possible labels. Thus when the best label is found for a sub-block, the best MV is found. For the motion vector labeling problem, the energy from the motion vector is

\[ E_{\text{data}}(M) = \sum_{b \in B} E_{3\text{DCT}}(l_i), \]  

(4.5)
where $B$ is the set of all sub-blocks in the region, $l_i$ is the $i$th label in the set of possible labels $L$, and $D_b(l_i)$ is the energy of sub-block $b$ given label $l_i$. $D_b(l_i)$ is equal to equation 4.1, where the MV input is the the MV representation of label $l_i$.

The energy given by $E_{smooth}$ is:

$$E_{smooth}(M) = \sum_{\{a,b \in B\}} V(M_a, M_b),$$  \hspace{1cm} (4.6)

where $V(M_a, M_b)$ represents the energy from the difference of the labels of sub-block $M_a$ and $M_b$.

$$V(M_a, M_b) = c\sqrt{(l_x(a) - l_x(b))^2 + (l_y(a) - l_y(b))^2},$$  \hspace{1cm} (4.7)

where $c$ is a constant, $l_x$ is the horizontal component of the motion vector and $l_y$ is the vertical component of the motion vector.

![Graph cut example](image)

(a) Example of Graph cut  \hspace{1cm} (b) The example with the cut removed

Figure 4.3: A graph constructed with 3 sub-blocks $p$, $q$ and $r$ is shown in (a). The sub-block $p$ is in currently labeled the sink, $p$ and $r$ are labeled as part of the source. (b) shows a cut where there are no changes, which implies the minimum cost cut is obtain from $t_p + t_q + t_r + (2 \ast n_{\gamma\delta})$.

To minimize the energy, a series of graphs need to be constructed, one for each possible label as shown in Figure 4.3. Let $G = (V, E)$ represents a weighted graph with two special vertices called the source $\gamma$ and the sink $\delta$. Let the cut $C \subset E$ be a set of edges such
that the sink and source vertices are separated in the induced graph $G(C) = (V, E - C)$.

Additionally, no proper subset of $C$ also separates the sink and source in $G(C)$. Let the cost
of the cut $|C|$ be defined as the sum of edge weights in the set. The minimum cut is defined
as the cut that separates the sink and the source in a graph with the lowest cost.

Let the set of possible labels $L$ correspond to a finite number of motion vectors that
a sub-block can be assigned. To find the lowest energy, all the sub-blocks in the region
are initially assigned a label corresponding to its best motion vector found. An iterative
procedure begins, where the best swap between every label in $L$ is found, and the swap
with the largest energy loss is done. This process is repeated until no further energy loss
is obtained with a single label swap. With this procedure a local energy minimum can be
found.

It has been shown that a swap of labels is equivalent to finding the minimum of a carefully
constructed graph [5]. Let $\delta$ and $\gamma$ be the two current labels that are being swapped where
$\delta \gamma \in L$. Also let $M(\delta)$ and $M(\beta)$ be the number of sub-blocks in the region that are
currently labeled $\delta$ and $\gamma$, respectively. The graph has $M(\delta) + M(\gamma) + 2$ vertices, each
sub-block with either label is a vertex, and the sink and source are vertices as well in the
graph. If the case where the number of vertices is equal to 2, because no sub-blocks are
currently labeled as $\delta$ or $\gamma$, as there is nothing to be swapped and there is no minimum cut,
the swap can be skipped.

An undirected edge exists between every vertex that represents a sub-block to every other
vertex in the graph. Edges between two sub-block vertices are called n-link (neighborhood
link) and edges between a sub-block vertex and either the sink or source is called a t-link
(terminal link). Let the source vertex be labeled as one of the labels, $\gamma$ and the sink vertex
be labeled the other label $\delta$. The weight of a t-link is the cost of that sub-block to be
assigned the motion vector; a t-link between vertex $p$ to vertex $\gamma$, $t_{\gamma p}$ is assigned the energy
value of sub-block $p$ with motion $\gamma$. The weight of a n-link is the smoothness cost between
the vertices as in equation 4.1. Each cut must contain exactly one t-link for every sub-block
vertex. If neither t-link is in the cut, then the source and sink are still connected, if both
are in the cut, a proper subset of can be used to separate the sink and source. The t-links in
the cut determine the best label swap, any pixel with a t-link cut from $\gamma$ should be labeled
$\gamma$, and otherwise it should be labeled $\delta$. There are many known algorithms to find the
minimum cut [5]. The one we adopted is the maxflow algorithm described in [4]

The complexity of graph cuts grows polynomially in relation to the number of labels
possible. For example, when a motion vector is allowed to move up to 9 pixels horizontally and vertically, there are 81 possible labels. With 81 possible labels, there are $81 \times 80 = 6480$ possible graphs needed for every iteration of graph cuts. Although this number is very large, in reality a small subset of graphs need to be constructed. A graph only needs to be constructed when there is at least one sub-block that is labeled either the source or the sink. Many regions have only several sub-blocks meaning that it is only possible to have several labels. Additionally, graphs with only one block need only choose the motion vector with the lowest cost.
Chapter 5

Image and Video Encoding

Digital video that has been segmented is able to achieve considerable compression because of greater predictability in both motion and color. Once each object in the video is segmented, key frames known as I-frames are encoded using SA-DCT for each block. P-frames are described in terms of motion from the previous frame. The motion of each block as well as the shape transformation and color changes are needed to be stored for each frame. The shape information is needed because the shape changes every frame. The error encodes the difference from the block referenced by the region and the true block. Images are compressed the same way I-frames are encoded in the digital videos.

5.1 Object-Based Image Encoding

The JPEG image format has been the worldwide standard for lossy compression of digital images since 1992 [3]. JPEG depends on the DCT transform with an entropy coder to provide lossy compression. JPEG2000 has emerged as a relatively new standard that adds additional functionalities that JPEG lacks, as well as having a better compression rate [18]. JPEG2000 uses wavelets instead of DCT in JPEG to provide a more flexible and efficient compression method. While JPEG2000 is a better image compression method, it currently is not widely used, unlike the popular JPEG format.

The techniques used here essentially follow the modern digital image compression standard JPEG. The bit stream encoded is made up of information grouped by each region in the image. Each region has two parts, the shape of the region and the color and texture data of each block in the region. The shape is encoded into Binary Alpha Blocks (BABs),
while the color and texture is encoded using DCT. An overview of the steps involved in image encoding is shown in Figure 5.1.

![Image Encoding Process Diagram](image)

**Figure 5.1: Overview of the image encoding process**

### 5.1.1 Texture and Color Encoding

While the whole blocks are encoded using DCT, the boundary sub-blocks use SA-DCT. To correct the mean weighting defect which dictates that in SA-DCT the value of the DC partially depends on the shape of the block, the average pixel value is removed from the block before SA-DCT is done. Instead of the SA-DCT being performed on the pixel values, they are performed on the DC-corrected pixel values, defined as:

\[
f_{DC}(B, i, j) = f(B, i, j) - \text{Mean}(B),
\]

where \( B \) is the block and \( f(B, i, j) \) is the pixel value at coordinates \((i, j)\) in block \( B \). The average is then put into the DC value instead of the value calculated, which is 0 since the average has been set to 0.

To reduce the number of bits needed to encode the average value, each block is predicted as having an average of the averages of the top and left block, whenever possible. This poses another problem, since the inner blocks are encoded using DCT, their DC values will be different than the average value. To correct this problem, the average value of the inner blocks will be removed for DCT as well, and the average will replace the DC value.
A major advantage of grouping blocks into regions as well as dividing blocks into texture regions is for increased predictability. In traditional block-based DC prediction, boundary blocks as well as blocks that border boundary blocks will have high error from prediction. The prediction of boundary block DC will be poor, since it is influenced from another region. The block being predicted by the boundary block will also be poor, since it is a new texture region. When textures are properly divided, each block within the region typically have similar DC values. These can be predicted with greater accuracy so most error values will be near 0.

Compression is controlled by the quantization matrix. Image quantization is the process of reducing the possible values a DCT component can take on. The JPEG group has a recommended standard quantization tables shown in Figure 5.2. The quantized value at location \((i, j)\) is defined as,

\[
Q_{\text{val}}(i, j) = \frac{\text{DCT}(i, j)}{Q_{\text{matrix}}(i, j)},
\]

where the \(Q_{\text{matrix}}(i, j)\) is the value of the quantization matrix at location \((i, j)\). The quality and size of the image can be adjusted by scaling the values in \(Q_{\text{matrix}}\) higher to provide more compression, or lower to provide higher quality.

![Quantization Tables](image)

**Figure 5.2: Standard JPEG quantization tables**

Low frequency values are quantized much less than high frequency values since changes of low frequency values are more noticeable than high frequency values to the human visual system. One would expect many of the high frequency quantized values to have a value of 0, since there are few high frequency components in natural objects and the quantized value for the high frequency components are high. This observation can be exploited to increase
CHAPTER 5. IMAGE AND VIDEO ENCODING

compression. The 2D matrix of quantized values is put into a 1D matrix where the values are put in using a zig-zag pattern which is shown in Figure 5.3. This would cause many values at the end of the matrix to be zeros, so run length encoding (RLE) can be used. RLE starts by reading from left to right, the non-zero number is output followed by the number of 0s that follow it. This process is shown in Figure 5.4.

![Figure 5.3: Zig-zag pattern](image)

Figure 5.3: Zig-zag pattern

(212, 8, 6, 0, 8, 0, 0, 0, 0, 1, 0, 0, 0, 0)  (212, 0), (8, 0), (6, 1), (8, 5), (1, 5), (0, 0)

(a) Original numbers  (b) RLE transformed

Figure 5.4: An example of the RLE encoding, it works well when heavily quantized, leaving many zeros. It actually causes expansion if there are not enough zeros.

It is desirable to have the quality dependent on the quantization matrix $Q_{matrix}$ and not the shape of the block. In SA-DCT the pixels are shifted toward the top left corner before quantization. If the same quantization matrix is used regardless of shape, smaller blocks will be quantized relatively less. The smaller quantization is due to the observation that the $Q_{matrix}$ values near the DC component are smaller, and increase going down and right. To make it independent of shape, the $Q_{matrix}$ values are scaled depending on the shape of the block.

It is ideal for the quantization matrix to dictate the quality and not the shape of the block. The zig-zag process in a sub-block is done but pixels which do not exist are skipped instead of padded.

Accurate block divisions allow multiple blocks of SA-DCT to have higher compression than a single DCT block. When two textures share a block, many high frequency values exist, which is the basis for Equation 3.1. This causes RLE to provide very poor compression, possibly even causing data expansion in high quality image compression cases where $Q_{matrix}$
values are small. If segmented properly, each sub-block will have its own unique texture. Since it is known that in natural images, objects have very few high frequencies [2], most high frequency values are expected to be 0. This causes very efficient compression using RLE.

Once all blocks have been encoded by applying SA-DCT, quantization, zig-zag transformation, and RLE encoding, the bits need to be written to the bit stream. Except the first, top left block, the DC value is predicted if possible from the average of the top and left blocks [3]. So instead of the DC value, the error between the DC value and the prediction is used. Each block is then written to the bit stream after entropy encoding is done, such as Huffman [18].

5.1.2 Shape Coding

Each region's shape is saved using a binary alpha plane. The value for the alpha plane is set so that pixels outside the object are set to 255 and inside the object is set to 0. Each alpha plane is tightly bounded by a rectangle where each side has a length that is a multiple of 16. The plane is then partitioned into 16x16 binary alpha blocks.

Improved Quadtree-based shape coding (IQTSC) [8] is an extension of Quadtree-based shape coding (QTSC) which provides a fast and efficient method of storing BABs into a tree structure. IQTSC starts with a quadtree decomposition by creating a square block with a length that is a power of 2, in our case of length 16. The block is then recursively decomposed into 4 equal sized square blocks until all pixels in the block are homogeneous. The decomposition can be represented with a tree, where each node has either 0 or 4 children: a node has 0 child nodes when all its pixels are homogeneous, otherwise it has 4 children.

In the traditional QTSC method, each node in the tree has 3 possible values: it can be opaque – all of the pixels in the block belong to the region, transparent – none of the pixels in the block belong to the region, or semi-opaque – some, but not all pixels in the block belong to the region. This can be encoded as a string composed of values from 0-2, where 0 corresponds to an opaque block, 1 is a transparent block and 2 is a semi-opaque block. This technique wastes bits because often it is not needed to know if a block is transparent or opaque, simply if it is homogeneous. With predictive coding the decoder can predict if a block should be opaque or if it is transparent. The decoder predicts the state of homogeneous regions larger than one pixel based on its known neighboring areas. The shape is decoded starting from areas neighboring known labeled areas. The type predicted, opaque
or transparent, is predicted based on the majority of known neighboring labels. Then the area around the newly predicted area is predicted, repeating until all areas are known.

In IQTSC each node needs only one bit to represent its state. If the block is a size greater than one, it has two possible values, either homogeneous or not homogeneous. If the block is of size one then it has two possible values as well, either it is opaque or transparent. An example is shown in Figure 5.5.

<table>
<thead>
<tr>
<th>Level</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>opaque</td>
</tr>
<tr>
<td></td>
<td>transparent</td>
</tr>
<tr>
<td>!1</td>
<td>homogeneous</td>
</tr>
<tr>
<td></td>
<td>non-homogeneous</td>
</tr>
</tbody>
</table>

(b) An 8 x 8 BAB, the gray area is where the region is opaque, the white area is where the region is transparent.

(a) IQTSC tree legend

(c) IQTSC tree

Figure 5.5: Example of IQTSC applied on an 8 x 8 BAB. The scan order used is SW, SE, NW, NE.

Although the predictive coder often correctly predicts whether a block is opaque or transparent correctly, some extra information is needed to correct it when the prediction is incorrect to make it lossless. In [8] Chen et al claim that their implementation of IQTSC is lossless, although it must be lossy since any predictive coder needs a mechanism to correct wrong predictions which it does not possess. Figure 5.6 shows two shapes that will be encoded the same and it is impossible to predict both cases correctly. We added an improvement to compensate in the event the predictive coder fails. Instead of labeling a block homogeneous or not homogeneous, it is labeled as homogeneous and predicted correctly or not. A special code can then be used to indicate the prediction for the parent
node was incorrect.

\[ \text{(a) An } 8 \times 8 \text{ BAB} \quad \text{(b) A different BAB sharing the same IQTSC tree} \quad \text{(c) The IQTSC tree for both a and b.} \]

Figure 5.6: Two BABs, with the same output tree in unmodified IQTSC. The scan order used is SW, SE, NW, NE.

Further savings can be obtained by shifting the shape in a block. The number of symbols needed to encode a shape in a BAB depends on the location of the object in the BAB. If we allow small vertical and horizontal shifts we can decrease the size of the tree while only adding a small overhead to encode the distance that the block is shifted.

5.2 Object-Based Video Encoding

Two types of frames must be encoded, I-frames and P-frames. I-frames ideally represent the start of a scene or when a large change in the video occurs. I-frames are encoded independent of other frames, while P-frames depend on previous frames to improve compression.

5.2.1 Object-Based I-Frame Encoding

The encoding of I-frames is essentially encoding a frame without using knowledge from any other part of the digital video. The techniques used here essentially follow digital image compression. The I-frame is encoded as described in the previous section, as an image.

5.2.2 Object-Based P-Frame Encoding

P-frames depend on the previous frame and only include the information to transform the previous frame into the current frame. The P-frame encoding is shown in Figure 5.7. It
is important to note unlike traditional P-frames, the motion is not estimated from the P-frame to the I-frame, it is estimated from the previous frame to the current P-frame. This is needed because the region is only known in the previous frames and not in the current P-frame. In traditional video encoding, a block is taken and the best motion is found by using a greedy approach, finding the minimum MSE from the last I-frame. In the method proposed, the blocks from the frame before predict the current frame using the motion calculated in section 4.2. Backward prediction is easier to implement since all pixels in the current frame have a prediction.

Backward prediction is possible when tracking regions by segmenting each frame similar to the I-frame. Each sub-block can be matched to the frame before. This would likely cause better prediction as well as more accurate region tracking. The computational cost is however prohibitive to this approach. Unlike MPEG which uses a simple partition scheme requiring almost no computation, image segmentation requires a large amount of computation. To encode P-frames forward prediction is used to significantly reduce computation.

Forward prediction poses new problems, since there will be pixels which have no predicted value, when no block is predicted to move into that area. There will also be pixels which have multiple predicted values when the area of more than one block is predicted to move to that pixel.

The holes have four possibilities, it may be a new region, it may belong to one region, it may belong to more than one region or it may be partially a new region. If a hole is surrounded by only one region, it is assumed to be part of that region, unless it is not similar to the color and texture energy. If it is not similar, a new region is created. Holes which border on multiple borders pose a more difficult problem, as some pixels could be from one region and other pixels from another region. Ideally, one would attempt to cut this hole into multiple regions, similarly to what was done in the initial image segmentation. However, currently a simpler method is used, where color is used to segment the hole. If no such color exists, the hole is classified as a new region. Although a new region as a part of a hole could exist, this case is not handled; instead it is assumed that holes exist as only a new region or part(s) of a current region. Once the region is predicted, the value of the holes is predicted to be a linear interpolation of nearby pixels in the region. If the region is declared as new, it is encoded separately like in an I-frame.

In the case where there are multiple predictions for a pixel's value, the average is taken. With this method, every pixel in the frame has a predicted value. The difference of these
predicted values to the true values is known as the error. The error needs to be encoded as well to accurately represent the frame.

The bit stream to encode the p-frame is separated into regions just like the I-frame. For each existing region three pieces of information are needed, the motion, the shape and the error. For a new region the information needed is the same as in the I-frame.

The motion is generated using a multi-resolution graph cuts method to increase the search area while using as few labels as possible. The image is reduced in resolution by a factor of four, and graph cuts is done on a $9 \times 9$ search area, resulting in a search area of $32 \times 32$ search area. Once the best label is found, graph cuts is done in the $4 \times 4$ pixel area around the best labels found.

The motion like the DC in I-frames is similarly predicted, except for the first motion, all other motions are predicted from the left and up neighbors. Only pixels where the object expands or contracts need shape coding. The shape is encoded using IQTSC. The error of the frame is encoded like in image compression, where the error values take the place of the actual pixel values.
CHAPTER 5. IMAGE AND VIDEO ENCODING

Procedure Encode P-frame

For each block find the best MV using the 3D DCT measure
For each region
{
    Initialize the initial labeling of the blocks by the best MV found
    While (true)
    {
        Create graph for each possible swap of labels
        If there is no swap that makes the energy lower
            break;
        else
            Do the swap that reduces the energy the most
    }
}
In each region, encode each block’s MV using prediction from the top and left block
Interpolate each sub-block based on neighboring blocks to find their MV

Find groups of pixels which were not predicted or have overlap with different regions.
For each group of pixels that overlap
{
    Use the 2D energy function and color matching to find the best region
    Add the pixels to the shape of the region
}

For each group of pixels that were not predicted
{
    Use the 2D energy function and color matching to find if it is part of a neighboring region
    If it is part of a neighboring region
        add the pixels to the shape of the matching region
    else
        Create a new region
}

For each region
{
    If it is a new region
        Encode the new region as in the I-frame
    else
        Encode each sub-block using SA-DCT with mean weight correction from the error of the prediction
        Encode the shape using IQTSC
}

Figure 5.7: P-frame encoding
Chapter 6

Experimental Results

6.1 Image Segmentation

Figure 6.1: (a) Lena Image at a 512 x 512 pixel resolution, (b) Energy feature map

A 512 x 512 pixel image of lena was used as a test image shown in Figure 6.1a. This image is challenging in many respects. The feathers in the hat and the hair have many edges and varying colors. The skin, hat and parts of the background have similar color which makes it difficult to segment apart. There is also a strong directional light causing
widely varying skin color, noticeably on the shoulders.

The figures from Figure 6.1b to Figure 6.6b provide an approximate block subdivision, to be used in Figure 6.7a. Refer back to Figure 3.5 for an overview of the segmentation process.

The feature map is shown in Figure 6.1b in which the energy level $E_{DCT}$ is calculated for a block centered at the pixel. The hair area although has a lot of edges, has very few high frequencies, so the energy value is low. This is because the block size is large enough so that the hair texture energy can be location independent. The feathers have a much larger complex texture pattern, causing a high energy area, with varying energy depending on the area because the block size used is too small. A part of the hat has a similar color and texture as the background, which is shown with a space in the hat, making it difficult to segment. Although there is in fact some low energy there, it is much smaller in magnitude as the other borders of the hat.

![Figure 6.1b: Edge detection and edge shifting](image)

Figure 6.2: (a) Edge detection on the 8 x 8 window feature map, (b) edge shifting on the Lena image

Edge detection on Figure 6.1b is shown in Figure 6.2a. The edges are found as expected, since within each energy band the magnitude of the energy is relatively similar. Edge shifting is done in 6.2b toward the center of the energy band. The shifting is relatively accurate, as many lines show up clearly, but it does not form a single thin line, as many parts of the
lines are more than 1 pixel in diameter. Very few unwanted edges are shown, except for the feathers in the hat region, as well as parts of the hat.

A binary mask with a low threshold is shown in Figure 6.3a. The wide band of low values is from the inner regions of the energy band since the energy slowly increases toward the center. The inner edge pixels shift also causing these thick edge bands. The smaller $4 \times 4$ is shown in Figure 6.3b. It is evident that the lines shown are sharper, as well as more detail is shown, such as the weak segment of energy between the back of the hat with the background. The high energy areas such as the feathers have a much more widely varying energy value.

The edge map on Figure 6.3b is shown in Figure 6.4a. Compared to Figure 6.2a, this edge map is much sharper, but has many unwanted edges in the feather region. The shift is shown in Figure 6.4a, which is much more accurate than 6.2b.

Figure 6.3: (a) Binary threshold map, (b) Energy feature extraction using a $4 \times 4$ window (b) on the Lena image.
CHAPTER 6. EXPERIMENTAL RESULTS

Figure 6.4: (a) Edge detection on the 4 x 4 window feature map, (b) edge shifting on the Lena image

The mask in Figure 6.3a is applied to Figure 6.4b, to generate the filtered image shown in Figure 6.5a. The mask filters out extraneous edges like some feathers, but doesn’t seem very effective in this image. This is because there are not any regions with textures that repeat over a large area. Edge thinning is then performed in Figure 6.5b. Edge thinning produces a 1 pixel thick border between regions, which is accurate in most cases in this Lena image. This shows that the changes within the energy band are much weaker than the change between the start of the energy band and the outside areas.

An adaptive threshold is applied which is shown in Figure 6.6a. All weak line segments are eliminated as well as small groups of edge pixels. Edge extension is shown in 6.6b. The edges near the border or joined to the border, also lines within the image are joined, such as the triangle above the top right part of the hat.

The divisions found in Figure 6.6b used an initial approximation of the edge boundary. An energy minimization was performed on each block to provide the final block division shown in Figure 6.7a. Some edges were thrown out when they did not divide a whole block, resulting in some seemingly missing division. Also lines which were too far away from the true division, such on the shoulder on the right side, did not divide properly. Figure 6.7b shows the result of the region growing. This image is over segmented as a result of
Figure 6.5: (a) Threshold 6.4b on the binary map in 6.3a, (b) edge thinning on the Lena image

poor texture matching. The areas of the face and feathers are significantly over segmented because of poor energy thresholds. Blocks with high energy should be merged with other high energy blocks instead of divided. A high energy non-divided block represents a complex texture, which should join easily with other high energy blocks.
CHAPTER 6. EXPERIMENTAL RESULTS

Figure 6.6: (a) Use adaptive threshold, (b) edge extension on the Lena image

Figure 6.7: (a) Divide sub-blocks, (b) Region growing on the Lena image
A 384 × 256 pixel image of 2 parrots is shown in Figure 6.8. Ideally the background doesn’t need to be divided much, since it is out of focus and fuzzy, making few high frequency AC components.

The feature map is shown in Figure 6.9. As expected, most of the background does not have high energy values. Most of the energy detected is from the region borders. The exception is the pattern near the eyes, which produce high frequency values.

The cuts in Figure 6.10 are well defined, except for some extra noise in the feathers and the facial areas. This is caused by the edges in the feathers. The regions grown are well defined as shown in Figure 6.11. By forbidding merging of a block division, better results can be achieved here.
CHAPTER 6. EXPERIMENTAL RESULTS

Figure 6.9: Parrots energy feature map

Figure 6.10: Processed image after edge extension
Figure 6.11: Region growing on the Parrots image
**CHAPTER 6. EXPERIMENTAL RESULTS**

Figure 6.12: (a) Baboon Image at a 256 by 256 pixel resolution, (b) Energy feature map

The Baboon in Figure 6.12a is a challenging image to segment using boundary based methods because of the large number of edges. The feature map shown in Figure 6.12b picks up the major edges, as well as a lot of extraneous edges across the face.

The boundaries in Figure 6.13a show the division of the eyes and nose well, but a large number of extra edges are shown. A reasonable segmentation is shown in 6.13b, doing a good job of dividing textures.

A simple $128 \times 64$ pixel test image with two regions is shown in Figure 6.14a. With low frequency textures, it easily finds the exact segmentation in Figure 6.14b.

A $128 \times 64$ pixel test image with four regions is shown in Figure 6.15a. Even with different shapes it is able to perform a good segmentation shown in Figure 6.15b.
CHAPTER 6. EXPERIMENTAL RESULTS

Figure 6.13: (a) Divide sub-blocks, (b) Region growing on the Baboon image

Figure 6.14: (a) An artificial image, (b) the regions found

Figure 6.15: (a) An artificial image, (b) the regions found
6.2 Image Compression

To calculate the number of bits needed, the entropy of each image is calculated [18]. To find the entropy, the probabilities of each symbol that needs to be encoded must be found. Each symbol is a number that is put into the bit stream, such as the error of the predicted DC value, or the values of the IQTSC output tree. The entropy is defined as:

\[
\text{Entropy} = - \sum_{i=1}^{N} p_i \log_2(p_i)
\]

(6.1)

where \(N\) is the number of unique symbols, and \(p_i\) is the probability of symbol \(i\). The probability of symbol \(i\) is calculated as the number of times \(i\) appears divided by the total number of symbols. The sum of all probabilities would naturally equal to one. The entropy is equal to the minimum number of bits needed to encode one symbol using the probability distribution \(P\), where \(P\) is the probability distribution of all the symbols used. Therefore, the entropy is equal to the bits per symbol (number in the bit stream). The total number of bits would be equal to the entropy times the number of symbols.

The compression results are shown in Table 6.1.

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Segmented data</th>
<th>Segmented shape</th>
<th>Segmented Total</th>
<th>JPEG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lena</td>
<td>588316</td>
<td>19701</td>
<td>608017</td>
<td>739408</td>
</tr>
<tr>
<td>Parrot</td>
<td>623694</td>
<td>15671</td>
<td>639365</td>
<td>742560</td>
</tr>
<tr>
<td>Baboon</td>
<td>835832</td>
<td>20190</td>
<td>856022</td>
<td>1169846</td>
</tr>
<tr>
<td>Test Image 1</td>
<td>5287</td>
<td>50</td>
<td>5337</td>
<td>8224</td>
</tr>
<tr>
<td>Test Image 2</td>
<td>2966</td>
<td>602</td>
<td>3568</td>
<td>14543</td>
</tr>
</tbody>
</table>

6.3 Video Object Segmentation and Compression

The test video shown in Figure 6.16 was used. The motion cost using the 3D energy function is shown for 3 blocks in a frame in the video in Figure 6.17. The best motion is found for each undivided block, as well as the region optimized MV in Figure 6.19.

Figure 6.17 shows the energy motion plane for 3 blocks. Although the entire plane is not monotonically decreasing toward the global minimum, it is clear that even in this large
motion, it is in general decreasing as it reaches the minimum energy as shown in Figure 6.18. Once the search is within a few pixels of the true motion, it is indeed monotonically decreasing toward the minimum energy.

A second artificial video is shown in Figure 6.20. This simple video shows that, with precise segmentation and sharp borders, a large compression improvement can be obtained as shown in Table 6.1.

To keep quality the same between encoding methods, a similar MSE was used while encoding using object-based video compression and MPEG methods. In the test case, diamond search was used for MPEG-2 as well. The results of this compression using the same bit calculation as in images are shown in Table 6.2.

<table>
<thead>
<tr>
<th>Video Name</th>
<th>segmented MSE</th>
<th>MPEG-2 MSE</th>
<th>Segmented Kbps</th>
<th>MPEG-2 Kbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toy Car</td>
<td>0.718</td>
<td>0.704</td>
<td>324</td>
<td>389</td>
</tr>
<tr>
<td>Test video</td>
<td>0.145</td>
<td>0.157</td>
<td>4</td>
<td>33</td>
</tr>
</tbody>
</table>
Figure 6.18: This graph shows how the motion search finds the minimum energy in Figure 6.17. Energy Ratio is the energy divided by the maximum energy of the search window.
CHAPTER 6. EXPERIMENTAL RESULTS

Figure 6.19: The direction for each undivided block is shown with a red arrow. The best direction is found using a diamond search which causes unreliable true motion. When the energy is minimized for a region, a good motion vector can be found.

Figure 6.20: A slow moving artificial video of a diamond shape slowly moving towards the right side.
Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis presents a novel segmentation method for use both in image compression as well as video compression specifically for the MPEG-4 standard. Although true object segmentation is not achieved, the gap between region segmentation and object segmentation is being bridged. Using consistent MVs, regions approximate video objects in digital videos. This thesis provides the following contributions:

Energy function
A new texture feature based on the DCT energy function $E_{DCT}$ has been introduced which can be used to find texture boundaries. This feature has the desirable property of being invariant to rotation or position in the image.

Region growing technique
A pyramid linking technique based on the localization method is used to create regions based not only on homogeneous features like color and texture, but also the boundary information found.

New motion measure
Instead of a greedy method that finds the lowest MSE to detect the best motion, a new motion quality measure $E_{3DCT}$ has been introduced based on 3D DCT. Similar to the energy feature proposed for image segmentation, it minimizes the encoding error by using DCT as the cost function. It promotes motion consistency and balances the
competing forces of the error values in a MV with the enforcement of smooth motion.

**IQTSC enhancement for MPEG**

An enhancement to the IQTSC shape coding algorithm has been introduced for use MPEG shape coding, while only adding a small amount to the compression size. By allowing lossless compression of shape coding, IQTSC is a low complexity algorithm that offers high compression rates for shape coding.

**MPEG-4 video encoding**

A partial algorithm to encode a digital video using VOPs is shown. The advantages of object based encoding in both video and image compression have been exploited to provide enhanced compression.

### 7.2 Future Work

Many improvements can be implemented to provide greater image and video compression as well as increase the speed of the algorithm. Additions are possible for scalable video and better object tracking as well.

**Dominant texture estimation**

Since regions are grouped by similar texture, an average texture can be generated which can be used as the basis for the region. Different schemes can be used to replicate the texture as a basic representation of the region so that instead of encoding the actual texture, one can only encode the difference between the base texture and the actual texture. Large savings can be obtained for simple or consistent texture. To guarantee compression savings, non-uniform regions can be encoded normally, if no gain can be achieved.

**Non-linear motion prediction**

To better predict the motion of each frame, the motion of the previous frames should be taken in consideration. Temporal prediction would lessen the amount of non-zero motion vectors that need to be sent as well as produce considerably more accurate motion estimation. Instead of having a linear prediction, multiple linear motions taken from temporally overlapping blocks of nearby pixels can be interpolated to
achieve a more precise motion estimation. The idea of 3DCT has only been partially investigated, more research could lead to a higher quality cost measure for MVs.

Temporal video segmentation
Many video segmentation methods use temporal motion for segmentation and scene transition detection, while the algorithm proposed does not. Regions which move together can be grouped into one object for the purpose of motion estimation, while regions which have parts that move independently should be split. The optimization method used here should be expanded to handle region splitting.

Efficient object motion search
The current method uses a hierarchical graph cuts approach to minimize the energy of the object, which requires a large number of computations. The minimum energy can be more efficiently solved using an Euler-Lagrange derived PDE diffusion. This search will direct block level 3D-DCT MV search efficiently and will optimize for motion consistency across the region.

Energy feature generation
The current energy feature generation is slow, calculating the DCT for smaller blocks then using these values for both $8 \times 8$ and $4 \times 4$ energy block generation would increase the speed significantly.

Energy based region growing
The energy should influence the region growing by having high energy blocks merge together. This represents a region which has complex texture, and should not be divided into many small regions.

Complete object model
The encoding method shown is only a partial MPEG-4 encoding model. To obtain much greater compression each object should be tracked across the scene, and the object stored. Each object in a frame can be represented with the position, rotation and any errors, for example from small lighting changes, instead of encoding it at each frame in the video.
Bibliography


