UNIVERSITY OF OSLO
Department of Informatics

Compression Methods for Video Streaming over an Event Notification Service

Håkon Torjus Bommen

Cand. Scient. Thesis

14th August 2005

Preface

In January 2003 the *Distributed Media Journaling* (DMJ) project announced a master thesis proposal with the working title: *Video Coding for Streaming over an Event Notification Service*. Two students were interested, *Frank Jensen* and myself. Both were given the opportunity to write a thesis, each with a different approach on the original task. Everyone involved in the project have been located at the *Simula Research Laboratory* located at Fornebu, just outside of Oslo.

The DMJ project is one of several projects conducted at the Simula Research Laboratory (Simula). Simula was established in January 2001, through a resolution of the Norwegian Parliament. The Laboratory conducts basic research in selected areas within information and communication technology. The DMJ project is funded by the Norwegian Research Council through the DITS program, under grant no. 126103/431.

While Frank Jensen was originally given the task of real-time compression, I chose the task of more CPU intensive non real-time compression. Motion compensation and bidirectional coding were suggested by my supervisor in this project, Viktor S. Wold Eide early on. How to implement the two techniques has been up to the author.

The vast amount of data in a video stream makes searching for bugs like searching for a needle in a haystack. An important lesson learned has been not to assume what a code section does, no matter how simple the task. Although this coding was challenging, most of the time spent o this thesis has been used to produce this document. This is by far the largest, and the most structured document I have ever written.

I would like to express my gratitude to the following people for there support and assistance during my master thesis: My Wife Anne Wicki and son Marvin Elias. Frank Jensen for comments and help. My supervisors, Frank Eliassen, for helping me keep focus on what was important, and Viktor S. Wold Eide, for help with the prototype and day to day guidance.

Håkon Torjus Bommen
Simula Research Laboratory, Norway
14th August 2005

Abstract

Distribution of video can be done by different means. A signal may be multicasted, where several receivers can watch the same video stream, or distributed on a one-to-one basis. A distribution where one transmission may cover several receivers reduces the total bandwidth usage, while the strength of one-to-one distribution is that a receiver may control the flow of signals.

Receiving devices vary in screen sizes, processing capabilities, and available bandwidth. As cell phones have gained access to the Internet there has been many new techniques for displaying a web page well on both cell phones and computers. An equal ability to scale video streams is also a much wanted feature.

The *Distributed Media Journaling* (DMJ) project attempts to implement a prototype that generates one signal that is displayable for all receivers regardless of bandwidth, processing capabilities and screen sizes. The prototype makes use of a *content-based network* to transmit video signals.

For compression the prototype makes use of forward prediction and *discrete cosine transform* (DCT). The video signals are compressed considerable, but it is still not a very efficient compression mechanism.

This thesis describes the addition of two compression techniques that will enhance the compression ratio of the prototype without lowering the quality of the signal. The techniques are called motion compensation and bidirectional coding.

The motion compensation algorithm shifts elements of a former frame in order to provide a frame that better matches a current frame. Most frames in the video are represented as a differential to another frame. If a better match can be found, less correctional data is needed, hence resulting in better compression.

Bidirectional coding contribute to compression by also allowing future frames to be used as a base for differential coding. In many cases better matches can be found in future then in former frames.

Experiments made as part of this thesis shows that by applying motion compensation and bidirectional coding the size of the differential code in the video stream is reduced by up to 70%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document describe the addition of two well known compression techniques called motion compensation and bidirectional coding. The goal is to provide a higher compression ratio for a prototype video server and client made in the *Distributed Media Journaling* project.

1.1 Distributed Media Journaling

Distributed Media Journaling (DMJ) is a research project focusing on capturing and indexing distributed multimedia applications in real-time. The goal of the project is to develop a framework that makes it easier to develop applications for real-time analysis of images, video and sound. This framework can be used to discover and categorize particular events in real-time.

One of the latest efforts has been to study the use of content-based networks for distribution of multimedia streams. While DMJ focuses on capturing events, the software used for transmitting video can be described as either an event notification service or a content based network. This thesis will mostly use the term content based network for describing the video transportation mechanism.

A video is divided into several layers. A base layer contain a minimum video signal, while several enhancement layers provide enhanced quality. An enhancement layer may improve framerate, signal to noise ratio, or add color. Each frame is also divided into regions. The different types of layers and the division of a frame into regions are called *video quality dimensions* throughout this thesis.

The layered video approach combined with a content based network provide a solution that is scalable on the server side, on the client side, and across the network. The processing in this framework may be divided between available computers giving server side scalability. A client may receive a video signal appropriate to the resources available giv-

ing client side scalability. Network scalability is provided by the content based network. Traffic across the content based network does not increase proportionally to the number of subscribers to a single video source.

A prototype named *Video Streaming over Content-Based Networking* (VSCBN) was made as a proof of concept.

1.2 Problem

A high compression ratio for video signals transmitted across a network is important because it conserves precious bandwidth. The DMJ project has implemented the VSCBN prototype for compressing and transmitting video signals. Compression in the prototype is based on forward prediction and *discrete cosine transform* (DCT). These are among the simplest forms of compression for video. The current compression technique lowers the bit-rate of the video signal substantially. It doesn't however exploit motion compensation nor bidirectional coding. Adding these to techniques to the existing compression algorithm should improve compression ratio considerable.

The motion compensating algorithm shifts elements of a former frame in order to provide a frame that better match a current frame. If a new frame can be coded as a differential to a frame that match itself more closely, less correctional data is needed. The reduction of correctional data improve compression rate. The process of finding suitable matches is a very time consuming process that require much processing power for real-time encoding.

Bidirectional coding contribute to compression by allowing future frames to be used as a base for differential coding. Better matches can often be found in future frames, especially if the frame to compress is the first frame in a new scene. Unfortunately the technique prevent the compression of each frame as they arrive. This creates an end to end delay, which makes this form of compression less suitable for applications with strict real-time requirements.

The layered structure of the video enables a viewer to subscribe to individual parts of the video. This layered structure comes with a cost compression wise. Each subscription made to a specific layer must also include all lower layers in the same video quality dimension. On the encoding side, no further assumption can be made on which layers a client subscribes to. This is a serious restriction since compression of video often involve reusing parts of a video stream. Parts of a video that a subscriber may not receive cannot be used as a base for further compression. Due to the layered structure the compression rate of the prototype cannot exceed the compression ratio of codecs (encoder/decoder) that are

not layered.

Increased compression rate in the prototype is a priority for the DMJ project. This can be achieved by implementing more sophisticated and resource demanding compression techniques. Some gain may also be possible from further optimization of the existing implementation. New compression techniques must however not break with the existing layered structure.

1.3 Goal

The goal of the work described in this thesis has been to improve the compression ratio of the VSCBN prototype by adding two well known compression techniques.

The challenge for this thesis is to implement motion compensation and bidirectional coding that compress efficiently without breaking the restrictions of the existing layered approach. Hopefully this should give the prototype a considerable higher compression ratio.

The added compression techniques introduce higher memory usage, a delay between encoding and decoding, and require much more computational power. The improved compression rate should at least justify this extra resource usage in most scenarios where bandwidth is a limited resource.

1.4 Method

The research method in this thesis is experimental. The prototype made in the context of the DMJ project has been modified to be capable of providing motion compensated and bidirectional coded video. The modified prototype will be referred to as the extended prototype.

Testing has been conducted to determine the gained compression rate of the extended prototype. Initially, tests were conducted comparing the updated prototype with all new features disabled to the original version. Next, the extended prototype was tested with bidirectional coding and motion compensation enabled, in combination and independently. The tests have been conducted comparing compression rate for various quality settings in selected video dimensions.

The two compression technique introduce an extra overhead which is currently sent uncompressed. The compression of these fall outside of the scope of this thesis. Processing speed has also been considered outside the scope of this thesis, although some effort has been made to produce code that executes fast.

1.5 Results

Experiments made as part of this thesis show the added compression techniques reduce the differentially coded data with up to 70% compared to the original prototype.

1.6 Structure of This Document

In this document we discuss additional components to a compression scheme to be used for video signals over a content based network. Chapter 2 provide general information about video compression, transmission (with focus on the Internet and content based networking), and the VSCBN prototype. Motion compensation and bidirectional coding is explained in greater detail in chapter 3. Chapter 4 describe the design choices for the added features, while chapter 5 give a detailed description of the updated prototype. The last two chapters contain test results, conclusion and a description of possible further improvements.

Chapter 2

Background

The *Video Streaming over Content-Based Networking* (VSCBN) prototype connects the use of content based networks and video with fine granularity along several different *video quality dimensions*. Before going into details, this chapter provide a basic introduction to video compression, transmission, and a description of the VSCBN prototype. Some of the information here may be required for fully understanding the rest of this thesis.

The chapter is made out of three main sections. The first section covers video compression but also describe elements from general compression and image compression that are used in video compression. Color theory and sub-sampling is also discussed in this section. The second section covers various techniques for transmitting video across a network, beginning with the multicast concept and ending up with a content based network. The third section gives a description of the VSCBN prototype.

2.1 Compression

This section describes compression techniques and color sub-sampling. The compression techniques are described in their most general form. Motion Estimation and bidirectional coding are two compression techniques that are explained in further detail in the following chapter. Readers who are confident with video compression may want to skip this section.

2.1.1 General Compression

The compression techniques described here can compress any computer file regardless of the structure of the content. Image compression may be based on this kind of compression, or use variations of these in addition

to a specific image compression technique to further reduce the compressed image size. More information with examples of the following compression schemes, except fractal image compression, can be found in [7].

Run Length Coding

One of the simplest forms of data compression is known as *run length encoding* (RLE), also known as *run length limiting* (RLL).

Suppose you have a text file in which the same characters are often repeated, one after another. This redundancy provides an opportunity for compressing the file. Compression software can scan through the file, find these redundant strings of characters, and then store them using an escape character, followed by the character and a binary count of the number of times it is to be repeated.

The run length technique may also be used to compress binary data. When compressing a binary source the number of consecutive zeros or ones in the bit stream is stored. In this way, the average number of bits needed to represent a character is reduced.

Huffman Coding

Huffman coding is an efficient, lossless compression technique. Symbols, for example characters in a text file, are converted to a binary code where the most common characters in the file have the shortest binary codes.

Binary codes for each symbol may be generated from the source that is to be compressed, or predefined using statistical values. A binary tree, usually called a Huffman tree, is often used where the symbols lie as leaf nodes. Common signals are placed close to the root of the tree. When decoding a symbol each bit represent moving to the left or right child node, starting from the root. Once a leaf node is encountered, the symbol associated with that node is added to the decompressed data, and the procedure is repeated. At the encoding side, a table containing all binary codes and the symbols they represent is often used. This is usually described as a *Variable Length Code* (VLC) table.

Arithmetic Coding

Instead of using binary strings of variable length, arithmetic encoding stores large numbers that each represent a varying number of symbols. According to statistical probability each character is associated with a large or small subrange of the number's maximum range. The first symbol is acquired by finding the subrange the stored number lie within. The

found subrange is then divided into smaller subranges using the same distribution. The next symbol is found by calculating which sub-subrange the number lie within. This process is repeated until the accuracy of the number is too coarse to match any specific subrange. Arithmetic coding is slightly more efficient then Huffman encoding.

LZ-77 and LZW

There is a simple, clever, and effective approach to compressing text known as *LZ-77*, which uses the redundant nature of text to provide compression.

LZ-77 exploits the fact that words and phrases within a text file are likely to be repeated. When they do repeat, they can be encoded as a pointer to an earlier occurrence, accompanied by the number of characters to be matched.

The encoded stream consist of pointers and uncompressed characters, which are distinguished by a leading flag bit. The uncompressed characters are extended from 8 to 9 bits, working against compression. A pointer is chosen when the pointer can be represented by fewer bits then the uncompressed characters it replaces would have required.

The key to the operation of LZ-77 is a sliding history buffer, also known as a *sliding window*, which stores the text most recently transmitted. When the buffer fills up, its oldest contents are discarded. The size of the buffer is important. If it is too small, finding string matches will be less likely. If it is too large, the pointers will be larger, working against compression.

LZW is a compression technique that is based on LZ-77. We start with a dictionary consisting of 256 single character strings. During encoding strings longer then the ones already in the dictionary are added. Once the dictionary has reached its maximum size, no more entries may be added. Refinements of LZW provide the core of GIF and TIFF image compression.

2.1.2 Image Compression

General compression does not concern itself with what kind of data is being compressed. Some techniques are best suited for compression of text where words and phrases are often repeated. Others depend on some characters being more common then others. The properties of a typical image makes it possible to create more efficient methods for image compression, some of which are described here.

Compression of an image is in video terms described as spatial compression, meaning compression within a frame.

Lossy and Lossless Compression

Image compression techniques are generally divided into two categories: lossy and lossless. If a compressed image can reproduce an exact replica of the original image, the encoding process is considered lossless. Lossy compression techniques achieve a higher compression rate, but the reproduced image is slightly altered compared to the original.

JPEG2000 is a highly efficient compression format that may compress both lossy and lossless [27]. The lossless compression have a compression rate between 1:2 (normal photographs) and 1:10 (medical image material). For lossy compression one may chose a compression rate between 1:10 (high quality) and 1:100 (low quality).

Discrete Cosine Transform

Compression of an image in the JPEG format is achieved by interpreting blocks of the picture as different weights of cosine patterns [10]. The image is first divided into small blocks containing 8 x 8 pixels. For each block a set of coefficients are calculated so that the sum of all cosine functions weighted by the corresponding coefficients closely match the original pixels. The process of finding these coefficients are called the *discrete cosine transform* (DCT).

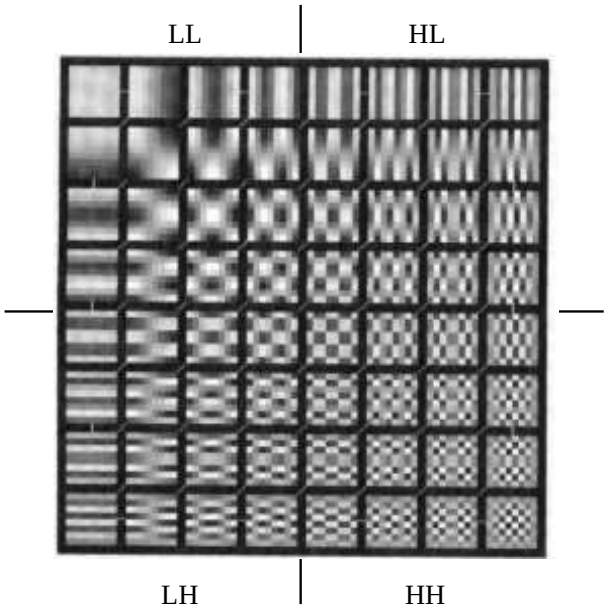

Figure 2.1: Cosine functions in two dimensions. Illustration taken from the MPEG2 manual

Cosine functions are used to describe waves. Every function can be described as wavelength, amplitude, and start position. Based on wavelength, cosine functions can be divided into groups of high and low frequencies. Cosine functions in two dimensions, used for image compression can be divided into four groups according to frequency horizontally, and vertically. The groups are named LL, HL, LH, and HH. (L for low-, and H for high-frequencies, horizontal first and vertically second), as illustrated in figure 2.1. With coefficients from the low frequency functions a low frequency image can be generated. These are the functions that make up the most noticeable features of an image. By adding the higher frequency information a higher quality image can be generated. By leaving out much of the high frequency information an image can be compressed without loosing the most noticeable features of the image. The process of leaving out information is called quantization.

In the process of quantization, a fixed or generated matrix is used. The quantization matrix reflects which cosine functions that are the most important for perceived image quality. When applying quantization, the most important low frequency functions are preserved, while the less important high frequency coefficients are more often set to zero. Only large coefficients in the high frequency domain survive the quantization process.

In the last process, high frequency coefficients, that are likely to be zero, are stacked together. This is done by reading the coefficients in a zigzag pattern. The values are then stored using lossless compression, for example Huffman or run-length. Zero values at the end of the chain are replaced with a *end of block* signal.

The overall compression depend on quantizer table, and the nature of the image. Images containing smooth surfaces can be highly compressed, while images containing many small details cannot be compressed to the same extent.

Wavelets

Wavelets are a group of functions that have certain special characteristics. In mathematical terms, a wavelet function is both oscillatory and localized in time [19]. The functions resemble that of a ripple in a pond since the average amplitude is zero (the ripple does not create more water, just displace it). Figure 2.2 shows examples of wavelet functions.

There are many different wavelets used for different tasks, not only image compression. A mathematical description of wavelets with focus on compression can be found in [6].

Since the wavelet functions fade out at the edges, blockiness that sometimes occur in DCT based compression can be avoided. The wavelet transform takes an image and computes its wavelet coefficients. The

Figure 2.2: Three examples of wavelet functions

coefficients, combined with the wavelet function, can later be used to reconstruct the image.

What makes wavelets better than older compression methods such as DCT based compression is their ability to adapt to the size and location of regions in the image. While DCT based compression often works in terms of eight-by-eight squares, wavelets can describe regions of varying size, shape, and location.

Fractal Image Compression

The idea of fractals is to exploit self-similarity in an image for compression [22]. This can only be solved by advanced mathematical methods that are still being a target of research. According to [21] fractal compression can compete with wavelets in compression ratio over peak signal to noise ratio. Decoding speed is linear O(n), but encoding is typically slow: up to O(n^4) for exhaustive search. Fractal compression is probably the most hyped compression technique ever. When proposed it was claimed that it would be able to compress in a ratio as large as 10.000 to one, but no one at the time were able to create a working encoder.

2.1.3 Video Compression

Spatial compression in a video uses the same techniques as in image compression. However, many of the image compression techniques are not as suitable in compressing video as they are in compressing images. Since each frame in a video has to be displayed at a correct time, the decoding needs to be executed at an appropriate speed. Video streams also consume much more storage space then images, so there is a greater need for a high compression ratio.

A video can be described as a sequence of still images. Simple video compression schemes like MJPEG is basically a series of images compressed with an image compression technique. The compression tech-

niques described in this section exploits redundancy between images for further compression. This type of compression is called temporal compression.

Efficiency and Reliability

To describe the efficiency of a video compression technique it is common to compare the compression rate to how much the signals are distorted. To measure compression ratio is quite straight forward, but deciding the quality of the resulting video is much more difficult. Perceived quality can be measured by simply letting a group of people watch two equal movie sequences, and letting them decide which has the best quality. This is a subjective test that may give inconclusive results. Another method for deciding quality of the resulting video is to calculate the *Signal to Noise Ratio (SNR)*. To calculate the SNR, an algorithm that examines each pixel, and adds up the differences between an original and a compressed version of a video is used. A combination of human observation and a computed result is usually used for deciding the quality of the resulting video after compression.

Compression efficiency may also refer to encoding/decoding speed, or hardware complexity. Real-time applications require fast encoding and decoding. Also, the cost of manufacturing specialized decoding hardware may be a reason for choosing one compression technique over another.

If a video is sent on an unreliable protocol, parts of the signal may be lost during transmission. By minimizing dependencies in the video the effect of such data loss can be minimized. Unfortunately this is in direct conflict with compression algorithms which try to avoid sending redundant data, creating dependencies in the process. Creating a codec often involve creating a balance between reliability, complexity, speed, signal to noise ratio, and compression rate.

Interframe Coding - MPEG-2

Instead of coding each image from scratch, it is possible to code a frame as the differentials from a former frame [10]. Two following frames usually do not change very much, meaning that the difference between them can be represented by less data then coding the frame from scratch.

By using the former picture, and the differential, the decoder can reconstruct the original image. A more advanced technique that uses both a former and a future image, called bidirectional prediction, is described in section 3.1.

The downside of interframe coding is that a user cannot decode any given frame independently. This makes interframe coding less suitable

for video editing.

Motion Compensation - MPEG-2

Motion compensation is among the basics for most video compression algorithms. It can be assumed that most pictures can be modeled as a transformation of a former picture by moving different parts of the image. The displacement need not be the same all over the picture, so motion vectors must be generated on several locations within each image in order to model the next image.

To reproduce the original image the motion vectors still need to be accompanied by an image containing the difference between the modeled image and the original, but the difference between the two images is smaller when applying motion compensation. What is gained in the compression of differential code after motion compensation will in most cases more then compensate the additional bits required to represent the motion vectors. Motion compensation is described further in section 3.2.

Conditional Replenishment - inter H.261

The inter H.261 compression scheme makes one important addition to a compression method named H.261. The enhanced method exploits temporal redundancy only through conditional replenishment. Compression is achieved by dividing the images into small sized blocks, and only sending blocks that have changed noticeable since last time the block was transmitted. For video that contains many static scenes and a small amount of noise, this may reduce the bitrate considerable.

Fine Granularity Scalability - MPEG-4

Bitplanecoding is one of the techniques used in amendment to MPEG-4 to achieve *fine granularity scalability* (FGS) [14]. Conventional DCT coding uses run-level coding to store the quantized DCT coefficients. Bitplanecoding is an alternative to run-level coding that provide granularity in the signal to noise domain.

After DCT coding each 8x8 pixel block, the 64 coefficient values are zigzag ordered into an array. A bitplane is defined as 64 bits, one bit from each coefficient value at the same significant position. The highest bitplane contain the most significant bit of each value, while the lowest bitplane contain the least significant bit. Each bitplane is compressed by a lossless technique. The highest bitplanes are discarded if they contain only zeroes.

By discarding low bitplanes some of the precision of each coefficient value is sacrificed. In other words, starting with the highest non zero bitplane, the image quality increases for each following bitplane.

Video Object Coding - MPEG4

Instead of just compressing each image, and apply regular temporal compression, MPEG-4 can compose the image from defined objects [11]. These objects can be images, video, web pages, sprites, backgrounds, shapes and textures. To construct a MPEG4 stream, the video source should define a video as a combination of MPEG-4 elements.

2.1.4 Colors

This section describes how human beings perceive light, different representations of color, and how subsampling of chrominance information can be made to compress a video or an image loosing only a small portion of the perceived quality.

The color models in this text are based on how humans perceive light and is referred to as additive color models. Other color models exist that describe spectral light that is not limited to how the human eye perceive light, and subtractive color models which describe how a given surface reflect or absorb light of different wavelengths.

How we Perceive Light

The normal human retina contains two kinds of light sensitive cells, the rod cells (active in low light) and the cone cells (active in normal daylight) [30]. Normally, there are three kinds of cones, each containing a different pigment. The cones are activated when the pigments absorb light. Combinations of activated cones at one location in the retina is what we perceive as color. Because the eye contains three kinds of cones, all visible colors can be represented by the activity level of the three cones. The color with wavelengths that activates one kind of cone is called a primary color. Studies show that all visible colors can be modeled by a combination of the three primary colors.

Perception Thresholds

There is a limit for how small color changes can be before they are unnoticeable for the human eye. Studies show that small changes in brightness are more easily detected then equally small changes in color. Extensive research by the Munsell Color Laboratory show that 8 bits per chrominance value produce enough colors to be below the threshold of

color-difference perception. For luminance the threshold lie at about 13 bits.

Color Models

RGB is a color model that describe a color by a combination of the primary colors. It contains three values - one for each of the primary colors: red, green and blue. Black is represented by three zero values, while white has all values set to the maximum value.

A color may be represented as a location within a three dimensional cube where the primary colors are represented by each axis. Instead of indexing by values for red, green and blue, one may index the cube by a vector starting from the "black" corner where all values are zero. This is illustrated in figure 2.3. The length and angle of the vector can be used to convert back to the primary colors. This method of indexing a color in the color cube separates color values from luminance values, which has certain advantages when used in video compression.

While RGB is a color model that represent colors by storing a mixture of the primary colors, HSV is a format where a color is indexed by hue, saturation and brightness (value). Brightness is linked to the distance to the black corner, hue is the orientation around the central axis from black to white, while saturation is based on the distance from the central axis. It is common to draw the HSV colorspace in the form of a hexcone.

YUV is a format based on the same principles as HSV that have finer granularity in the areas describing different shades of green. Another model very similar to YUV is called YCrCb. YUV and YCrCb is commonly used for video.

Color Subsampling

By representing a color in the YUV or similar formats it is possible to reduce the resolution of the luminance and chrominance plane independently. Since changes in colors are less noticeable then changes in brightness it is common to downsample color values. This can be done in two ways. The first is to reduce the number of bits representing the color values. The second and more common alternative is to set a lower resolution of the color plane.

Bit Downsampling An image consist of a number of pixels, each representing a color with a number of bits. A large number of bits can produce a large number of different colors. Downsampling could be done by cutting off the least significant bits of each chrominance value.

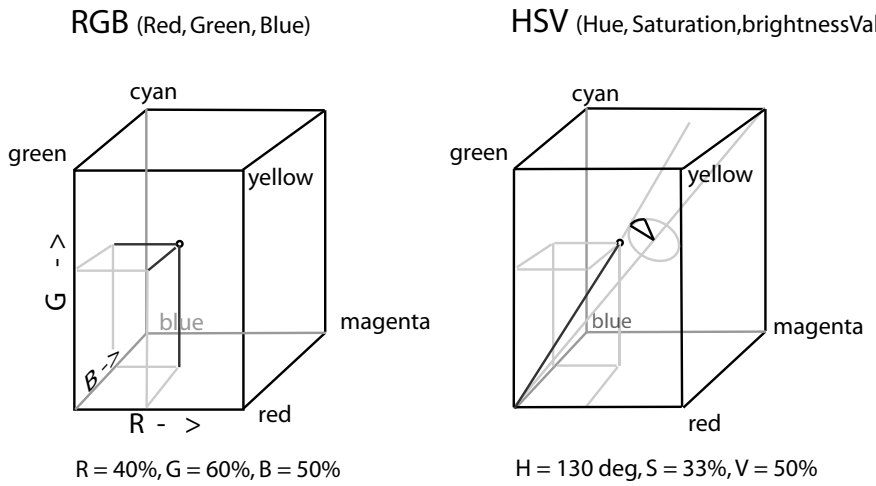

Figure 2.3: A color index in the color cube indexed by RGB to the left, and HSV to the right.

One reason to avoid bit downsampling is to prevent double rounding errors during colorspace conversion. If a value is rounded up during conversion from RGB to YUV, and rounded up again when converting back to RGB, the resulting value will be too high. This rounding error shows up as a contour, or banding in the image. By representing each value with a larger number of bits, the effects of rounding errors is reduced.

Resolution Downsampling The second and more common method to downsample colors is to reduce the number of pixels in the color plane. This kind of subsampling can be done at different ratios. The chrominance channel may be half the resolution (4:2:2), one quarter the resolution (4:2:0) or one eight the resolution (YUV9) [26]. By moderately downsampling the chrominance channel very little visual quality is lost. A too high subsampling rate can result in noticeable color artifacts around the edges of brightly colored objects.

Resolution downsampling techniques are often described using three numbers separated by colons. Each number represents the number of samples in a 2x2 block, so 4:4:4 represent no subsampling while 4:2:0 represent a video where four luminance pixels are matched with only two chrominance pixels. There is also a subsampling named 4:1:1 which uses the same downsampling rate, but does so in a slightly different manner. In fact, even the 4:2:0 format is interpreted differently by MPEG-1 and MPEG-2 [24], so this kind of notation does not give a full description of a subsampling method.

2.2 Transmission

Normal packaging works poorly for video transmission [16]. Reliable transmission (TCP/IP) is slow. Unreliable transmission (UDP/IP) may loose or duplicate packages. If different parts of a video stream are dependent on each other, the loss of some packages may render a large portion of the transmission useless.

This section discuss the multicast concept, event notification services, and content based networks. A content based network routes packages based on package content rather then destination address. By using a content based network, packages that should reach more then one receiver can make more efficient use of the network, compared to normal packaging. Content based networks are closely related to event notification services, and implement the multicast concept.

2.2.1 Multicast

A signal is considered multicasted if one transmission may cover several receivers. The most efficient strategy is to deliver the messages over each link of the network only once and only create copies when the links to the destinations split. This is called *down stream replication*. IP multicast is the most known multicast protocol, but other protocols exist that implement the multicast concept. Real-time multimedia streaming (such as Internet radio) is one type of transmission that benefits from the use of multicast.

The assignment of IP multicast addresses is controlled by the Internet Assigned Numbers Authority (IANA). IANA has assigned the Class D address space to be used for IP multicast. All IP multicast group addresses fall in the range between 224.0.0.0 and 239.255.255.255. No mechanism has yet been demonstrated that would allow the IP Multicast model to scale to millions of senders and millions of multicast groups as would be necessary to make fully-general multicast applications practical in the commercial Internet [29]. This is one of the reasons most communication on the Internet is based on one-to-one unicast connections.

2.2.2 Event Notification Service

In programming languages, for example Java, keystrokes and mouse movements are recorded as events. An event object is passed to a set of registered listeners that reacts accordingly. The idea that an object can react to changes in another object has also been adapted for use between objects at different locations [5]. Distributed event systems have two main characteristics.

Heterogeneous Since notifications are the unit for carrying data across a network, event notification services are well suited in a heterogeneous environment. Notifications can be implemented without platform specific code.

Asynchronous Senders and receivers are decoupled. The publisher of an event does not concern itself with which objects an event reaches. Interested objects are responsible for subscribing to interesting event types, and the event notification service is responsible for distributing the notifications in a sensible way.

There are several implementations of Event-Notification services. There may be a centralized server that all notifications must go through. This is a sensible approach for events that should be restricted to a local area network. Other implementations may consist of a network of nodes that events may pass through. The obvious advantage of a distributed design is scaleability.

Publish-subscribe is one of the paradigms in use. An object that generates events *publish* what type of events it may generate. This is called an *advertisement*. Other objects *subscribe* to the events that are of interest to them. The containers carrying information about an event are called *notifications*.

2.2.3 Content-Based Networks

The Internet today is based on address-based unicast or multicast networks, where packages are given explicit destination addresses. The key element in a content-based network is that routing is based on content rather then destination addresses [4]. A distributed event notification service can therefore be considered to be a content-based network.

The greatest challenges with content based routing lies in deciding what messages to forward to each neighboring router. This must be done so that messages are sent to all interested nodes in the most efficient manner by applying the multicast concept. The movement of messages through a content-based network is driven by predicates applied to the content of the messages. Forwarding in such a network amounts to evaluating the predicates stored in a router's forwarding table in order to decide to which neighbor routers the message should be sent.

Content-based networks include SIENA, Keryx and Gryphon [4]. There are also several centralized content based services like GEM, Elvin and Yeast.

2.3 VSCBN Prototype

This section describes the *Video Streaming over Content-Based Networking* (VSCBN) prototype before the modifications describes in thesis were added. The prototype provides fine granularity video streaming over content-based networking

The content-based network provides scaleability at the server and across the network, with respect to the number of receivers. Fine grained selectivity along different video quality dimensions is provided to support heterogeneous receiver capabilities.

VSCBN is currently a proof of concept prototype to demonstrate how a video signal can be transmitted to multiple receivers using the SIENA network. It is an experimental program, and is not intended to compete with comparable commercial software. It has not been tested on other platforms than Linux, but should port easily since most of the code is in Java. The prototype was originally made by Viktor S. Wold Eide, and is developed in the context of the *Distributed Media Journaling* (DMJ) project [25] at Simula Research Laboratory.

2.3.1 Distributed Media Journaling Project

The DMJ project focuses on indexing distributed multimedia in real-time by making use of automatic content analysis. One of the motivations for creating a fine granularity video stream has been to easily divide the workload of content analysis between available computers.

2.3.2 Features

The DMJ project has addressed the problem of heterogeneous receivers and efficient transmission by making use of a content based network for transmitting fine granularity multimedia signals. The fine granularity scheme ensures that receivers may subscribe to signals they are capable of handling, while the content based network provides a means to transmit signals efficiently only to those receivers who are interested. Real-time capture and streaming has also been a primary feature for the VSCBN prototype.

Scalability

With the combination of fine granularity scalability and content based networks the prototype offer scalability on the sending and receiving side, as well as across the network.

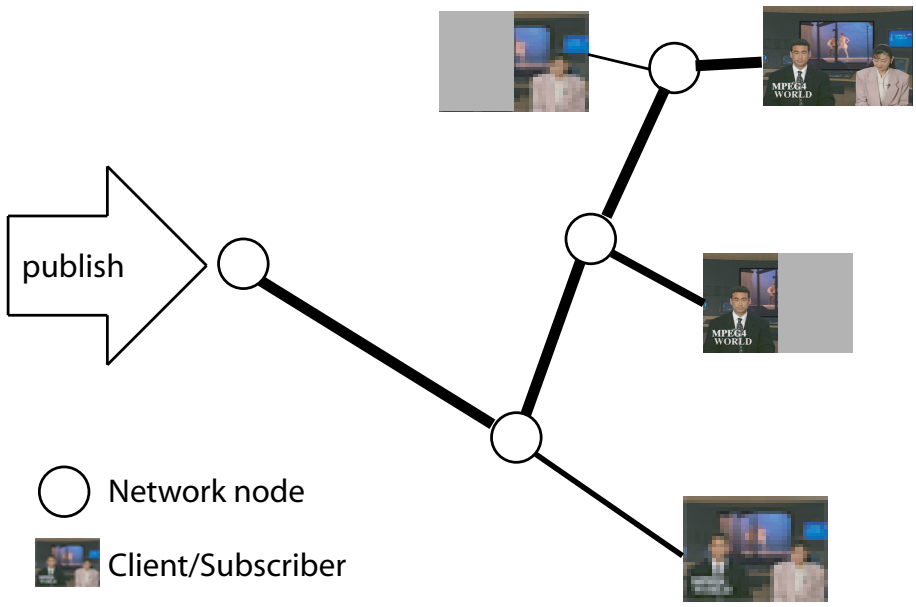

Figure 2.4: Video subscriptions between VSCBN server and clients, passing through a content based network

Sender side scalability Video servers transmit a video signal once, regardless of number of clients and the preferences of each client. The workload may also be divided between several computers.

Network scalability If there are no interested receivers, no signal is transmitted over the network. If there are several interested receivers one signal is sent from the server, which is replicated at the node where the path to each receiver is split. An example of transmissions across such a network is illustrated in figure 2.4.

Receiver side scalability Each receiver may specify the quality of the video signals to receive in several dimensions. Each dimension can be balanced according to user preferences to make the most of limited resources like bandwidth, CPU, display, or power consumption.

Fine Granularity

To achieve receiver side scaleability the video is divided into several dimensions. The receiver chooses a quality level for each dimension. A signal that is well suited for the receiver can be achieved if the levels are set appropriate. The dimensions are:

Luminance and Chrominance A user may choose to subscribe to a black and white signal containing luminance values only. The additional subscription to a chrominance layer will produce color images.

Signal to Noise Ratio Different quality levels are sent on different layers, one built on top of the other. A receiver must subscribe to a base layer, but may also subscribe to higher levels in the signal to noise dimension. This will produce a less distorted video signal.

Temporal Resolution A receiver may choose to subscribe to a subsection of all frames. All frames are divided into time layers so that a subscription to a layer and all lower layers will produce a constant framerate. A higher subscription level produce higher framerate.

Region of Interest Each frame is divided into a number of blocks called superblocks, which are indexed by row and column. A receiver may choose to subscribe to each superblock individually.

A layer representing multiple resolutions could have been added to the above list. This can instead be simulated by downsampling each frame from a low quality subscription.

The video is divided into video quality dimensions as illustrated by figure 2.6. Each frame belong to a temporal layer. The frames are divided into regions indexed by row and column. Each region consist of chrominance and luminance data, which again consist of a base layer and three enhancement layers for increased image quality.

Real-Time Capability

There are some considerations in making a real-time application. One is to have a compression scheme that is fast enough to encode and decode a reasonable number of images per second. Another is that the overall delay between sender and receiver should not be more then a fraction of a second. Compression techniques which have not met these two requirements have not been added to the VSCBN prototype.

Performance

In [9] tests of the compression rate of the VSCBN prototype is described. It also show that the granularity dimensions are truly independent of each other.

2.3.3 Compression

There are several video compression techniques to choose from. The techniques used by the prototype are all commonly used and well known.

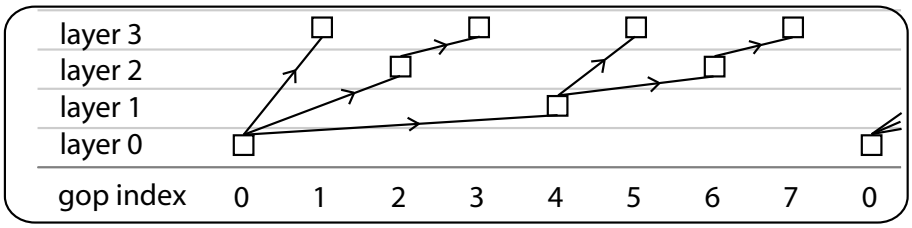

Figure 2.5: How reference frames are chosen using forward prediction only

Color Sub-sampling

The VSCBN prototype does, if needed, a color conversion from RGB to YUV. It then apply moderately color subsampling (4:2:0) where color information is one quarter of the original resolution. It is the same color space used by MPEG-2, halving the resolution both horizontally and vertically.

Spatial Compression

Different video coding techniques were considered in the design. The choice fell on *Discrete Cosine Transformation* (DCT), a compression technique used in MPEG-2. DCT is a block based compression technique. That means that the algorithm divides the frame into blocks, and compress each block individually. DCT applies to blocks of 8x8 pixels.

DCT is a safe choice for spatial compression. It is well known, and its block based structure makes it a simple matter to build additional compression techniques like motion compensation on top.

Temporal Compression

Temporal compression involves encoding the difference between two images so that the first image and the differential is enough to reconstruct the second image. The prototype does this in the most basic way by subtracting pixel values from a given reference frame from the pixels at the same horizontal and vertical position in the frame to compress. The resulting differential values lie in most cases closer to zero, and can therefore be compressed more efficiently. The exact frames used as references within a group of pictures (GOP) is illustrated in figure 2.5. Differentially coded data is only sent if it contain a noticeable change.

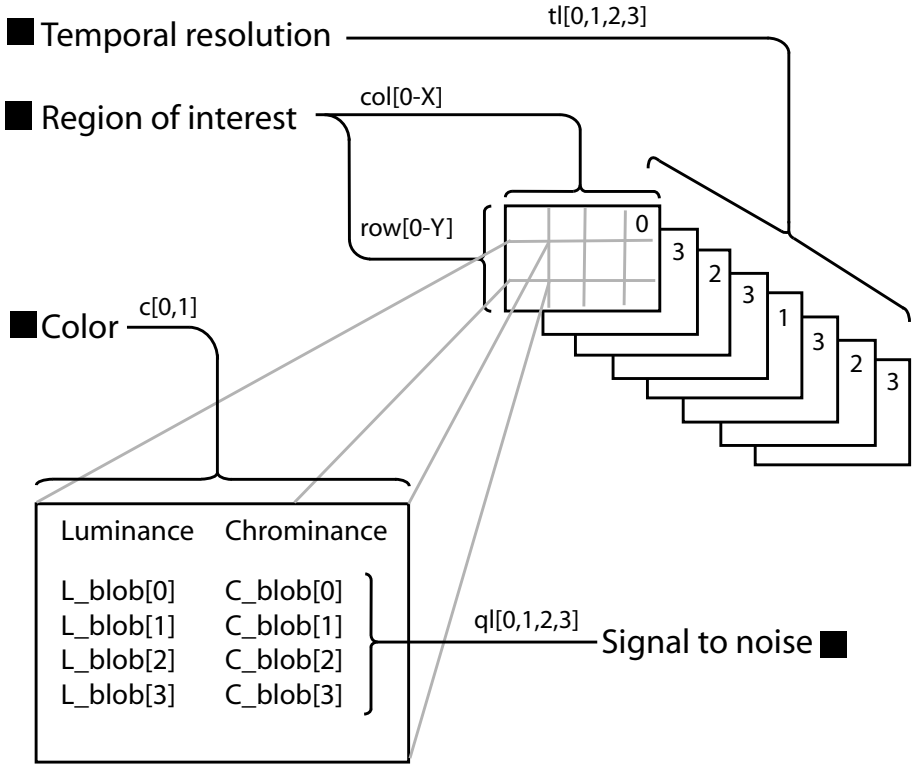

Figure 2.6: How the data is structured to provide granularity in the four video quality dimensions

2.3.4 Entropy Coding and Transmission

After compression, the resulting data is serialized and packaged into units that can be transmitted across a content based network.

The Entropy-Coding Stage

To store DCT data efficiently, coefficients that are likely to be of similar value are grouped together. This is done by reading the values in a zigzag pattern. Next, lossless compression is applied on that structure. The prototype makes use of fine granularity scalability, described in section 2.1.3. Each bitplane is compressed using Huffman generated VLC tables based on statistics generated from various video clips.

Transmission

Notifications are the units that are routed by the SIENA content based network. Each notification contain enough information to route the appropriate notifications to interested receivers, and a video payload.

To transport video signals efficiently it is important to send an appropriate number of notifications. If the notifications are too small, the number of notifications increase which leads to extra overhead. If a notification is too large it will be fragmented into smaller pieces during transmission.

The number of notifications used for sending a single image depends on the number of layers containing enhanced quality in each dimension. The maximum number of notifications per frame can be calculated by multiplying the number of layers in each dimension. The notifications will vary in size. The chrominance values are for example downsampled so that notification carrying these values are about half the size of a notification containing luminance values.

2.3.5 Software

The VSCBN prototype has been used with a content based network named SIENA, and an updated version of SIENA extended with IP multicast support. The experimental programs developed by DMJ are available from the DMJ projects homepage [25], published under the GNU public licence.

SIENA

SIENA is an Internet Wide-Area Event Notification Architecture that strives to maximize expressiveness without sacrificing scaleability [4]. It uses

event filters for matching subscriptions to notifications. An event filter contain attribute constraints by the use of operators like $=, \neq, <, >$, etc. Several event filters can also be combined to produce so-called patterns.

The prototype makes use of one or more event filters to enable subscription to different video dimensions. SIENA was chosen as the distribution network. It supports patterns and general peer to peer networks, and is released under the GNU public license.

SIENA extended with IP multicast support

To enable a more efficient distribution of notifications at the end-nodes of a content based network IP multicast may be used. If there are several receivers connected to a single end-node subscribing to the same notifications, the end-node may transmit each notification once instead of one time for every receiver. The service is targeted at usage within a local area network or an administrative domain. According to its developer the prototype provides high performance and scaleability [8].

2.4 Summary

This chapter gave a general introduction to video coding and content based networking, two important domains in the VSCBN prototype, along with a description of the prototype itself.

The prototype first converts the color-space of the video from RGB to YUV if needed. It then subsamples the chrominance plane by quartering the resolution. Compression is based on DCT, combined with forward prediction. The resulting compressed data is placed into SIENA notifications, which is the unit carrying data in the content based network called SIENA. By subscribing to the right notifications, a receiver can choose the quality in each of the video quality dimensions. The prototype scales well on the sender and receiver side, and the content-based network provide efficient transmission across the network.

Chapter 3

Temporal Compression

This chapter gives a description of bidirectional coding and motion compensation. The description here provides the foundation for the addition of these two techniques to the VSCBN prototype.

3.1 Bidirectional Coding

Forward prediction uses a previously decoded frame as a basis for creating a new frame. Bidirectional encoding involves also using a future frame as a reference for differential coding [3]. On some occasions, information needed to compress a frame is only present in a future frame. Bidirectional coding contribute to compression by allowing this information to be moved back in time.

When using bidirectional coding the encoder has to wait for a future image to become available before starting compression. This creates a delay which makes this technique less suitable for real-time compression.

3.1.1 How Bidirectional Coding Works

As an object moves, it conceals the background at its leading edge and reveals the background at its trailing edge. The area of background that was previously concealed normally requires new data to be transmitted because this information can not be obtained from a previous frame. A similar problem occurs if the camera pans as new areas come into view. In figure 3.1 we illustrate three frames from a video stream of a flying plane. Frame number two can be closely modeled from the first with the exception of the previously uncovered areas in black. These areas can instead be taken from the outlined areas from the third frame.

The use of bidirectional coding minimizes the problem of uncovered areas by allowing information to be taken from frames before and after the current frame. If a new area is being revealed, it will be present in

Figure 3.1: Three frames from a video where the camera follows the airplane, panning to the right. The dark eras in frame two can be reconstructed from the outlined eras in frame three, but are not present in frame one.

a later frame, and the information can be moved backwards in time to create part of an earlier picture.

Bidirectional coding reduces the amount of differential data needed, thus improving compression rate. When coding a bidirectional frame there is a choice of using a former frame or a future frame as reference. A combination of the two frames can also be used. The coder should consider several possibilities and choose the frame, or frames, that give the highest compression as reference. A signal describing which frame information is taken from must be added to the video stream.

3.1.2 Frame-Sequences

Frames can be divided into three categories depending on which frames it may use for reference. Intra frames (I) contain all the information needed to generate the full picture. Predicted frames (P) are coded with reference to a past frame. Bidirectional frames (B) may be coded with reference to both past frames and future frames. An example of a video stream using all three types of frames is illustrated in figure 3.2.

Although B-frames are the most compressed, a regular stream of I-frames is needed to recover from inaccuracies and errors in the transmission. If a part of the signal is lost or distorted, the video signal will not be corrected before an I-frame is sent. The frame-sequence for a video can often be specified in the encoder program. For figure 3.2 the *Group Of Pictures* (GOP) can be written as *IBBBPBBB*.

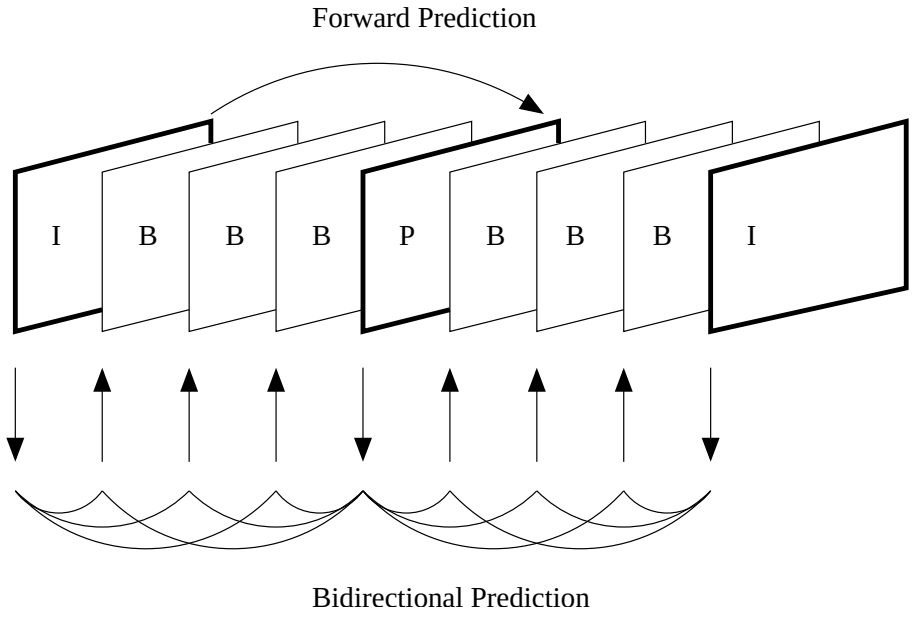

Figure 3.2: Information flow between I, P and B-frames in the MPEG format

3.1.3 Dynamic GOP

The frame-sequence described above is static i.e. it doesn't adapt to the current video. It is possible to set the number of consecutive B-frames, and the number of P-frames in a GOP based on the characteristics of the video. This is sometimes called *Scene Adaptive Dynamic GOP Structure* [23]. A dynamic GOP can lead to more efficient compression. GOP lengths may be set to synchronize a big change between two following frames, for example a scene change. This avoids using differential coding between frames with little or no redundancy. Also, if a frame sequence contain only small changes from frame to frame, more B-frames can be used without reducing the quality significantly.

3.1.4 Requirements

Bidirectional coding requires higher memory usage at both encoder and decoder side. There is also a higher computational cost in the encoding process, and a higher overall delay.

Extra Memory Usage

Bidirectional frames cannot be encoded before the future reference is available. Since frames cannot be encoded as they arrive, the frames must be buffered in order to encode them at a later stage. Extra memory is required to store the buffered frames.

At the decoding side, the frames are received in the order from which they can be uncompressed directly. Bidirectional frames are not sent in the captured order, so these frames must be reordered before they can be displayed. This reordering queue also requires extra memory.

Computational Requirements

Bidirectional coding involves using both future and former frames as reference frames. A forward and a backward reference lead to a higher workload for the motion estimating algorithm. Instead of finding matches in a given former picture the algorithm now needs to search both a former and a future frame. An alternative is to choose the reference direction by a simpler method first, and then do a motion vector search in that direction only.

3.1.5 Optimization

Bidirectional coding increases the amount of meta-data by adding direction signals to the video stream. For a high definition video signal this extra payload may take a very small portion of the video signal. For a low quality video this extra payload can be considerable. There should be a balance between meta-data and correction data based on the likely usage of the compression technique. The size of an area a reference applies to, and what information they may contain affect the size of the added meta-data.

The process of evaluating which reference that will produce the most efficient compression, is the same process as the one used in motion compensation, and will be described in section 3.2.4.

Information Contained in a Reference

A reference may go strictly forward or backwards, or specify an average between the two possible references. To signal a straight forward or backward reference only one bit is required to store the direction signal. If many signals point the same direction they may be further compressed. Allowing the combination of references may produce better matched, but require slightly more storage space and more computational power. If

both future and former frames are unsuited for differential coding, coding from scratch may in some cases be the most efficient option.

Frequency of References

A reference signal may apply to an entire frame, or may apply to smaller blocks of a frame. Using the same direction signal for the entire frame solves the case where a big change occurs between two frames. This approach however does not work well with panning. In the leading edge new information is introduced which can only be obtained from a future frame, while simultaneously at the tailing edge information from former frames is needed to represent the elements disappearing from the frame.

3.2 Motion Compensation

Section 2.1.3 gave the general idea of compensating for motion in a video. This section describes the process in further detail, and describe alternative algorithms and settings for calculating motion vectors.

Motion compensation can be very processing intensive at the encoding side. There are several strategies for calculating motion vectors. More thorough methods give better results, but these are also the most time consuming.

3.2.1 How Motion Prediction Works

Block based motion compensation uses blocks from a previously decoded frame to construct a modeled image of the frame to compress. The previously decoded frame is called a reference frame. For each block in the frame to compress a search algorithm finds matching blocks in the reference frame and if suitable, its motion vector is substituted for the block during transmission. If no sufficiently good match can be found, a block may be transmitted without using data from its reference image.

Spatial compression is used to compensate for the inaccuracy between the modeled image and the frame to compress. The closer the modeled image is to the frame it is to represent the less correctional data needs to be transmitted.

Motion vectors may not correspond to the actual movement within the scene. This may be due to noise, weaknesses in the matching algorithm, or local minima. If, for example, an object changes shape during the movement, the motion vector predicting algorithm may not be able to follow the exact movement due to the distortion. The primary goal of motion prediction is not to find the actual movement in a video, but rather to find matches that in turn will result in efficient compression.

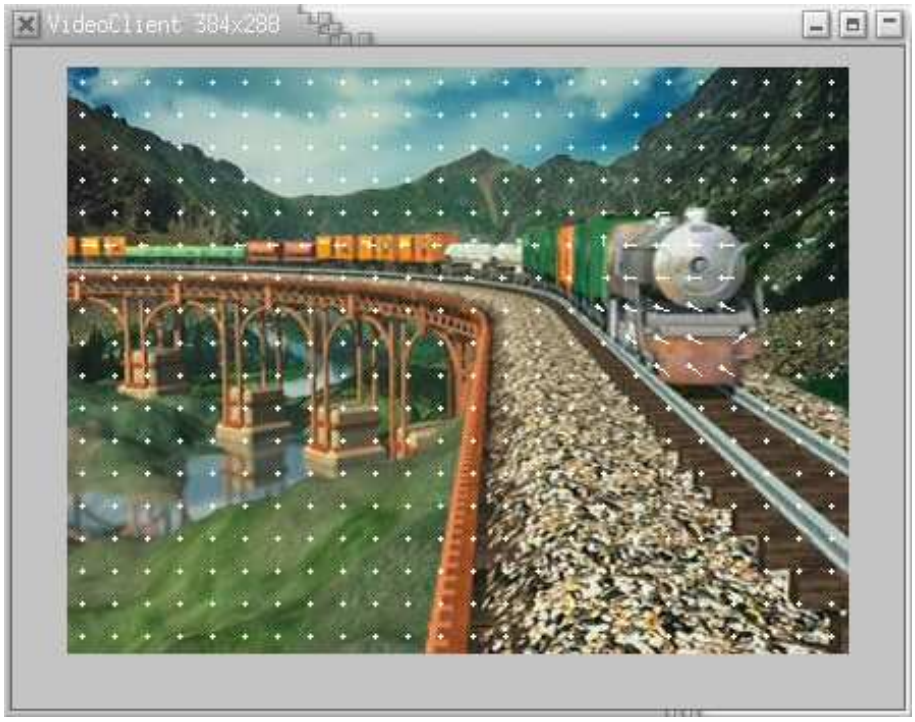

Figure 3.3: A video frame showing calculated motion vectors. The dots are placed at the position in a former frame where the match was found. The sample is taken from SimCity3000 intro.

Figure 3.3 show a frame from a video with calculated motion vectors drawn on top.

3.2.2 Units for Motion Compensation

Block size affects the performance of the compression technique. Fewer larger blocks will result in fewer motion vectors transmitted. Smaller blocks result in a greater number of motion vectors, but each matching block is more likely to closely match its target and less correction data will be required.

The first considered units for motion compensation were squares. Later research indicate that motion is more likely to occur in the horizontal plane then the vertical, and so rectangles should provide a slightly more efficient unit. The newer standardized codecs use variable sized blocks, deformable, or mesh based algorithms [13]. These more advanced methods have not been considered implemented in this thesis, and are only described here briefly.

The ultimate unit for motion compensation is the one that closely matches a moving object, using a minimum number of bits describing this area. In figure 3.4 we show three examples of units for motion compensation. The upper right illustrates squares of a fixed size as a unit. The lower left illustrates variable block sizes. The ultimate units are the ones that match the objects in the video most closely. This is illustrated in the lower right image.

Variable Size Block Matching (VSBM)

Moving areas can be quite accurately matched by allowing variable size square blocks. Such methods are known as Variable Size Block Matching (VSBM) methods. Most commonly the segmentation is recursive: each block is either split into several sub-blocks, or kept whole. This creates a tree structure, where each node is a block [20]. In this way small moving objects can be modeled without too much of the background being moved at the same time.

Deformable Block-Matching Algorithms (DBMA)

Using deformable blocks, such as triangles or quadrilaterals instead of simple squares makes it possible to handle more complex motions like rotation, scaling and shearing. This leads to *Deformable Block-Matching Algorithms* (DBMA).

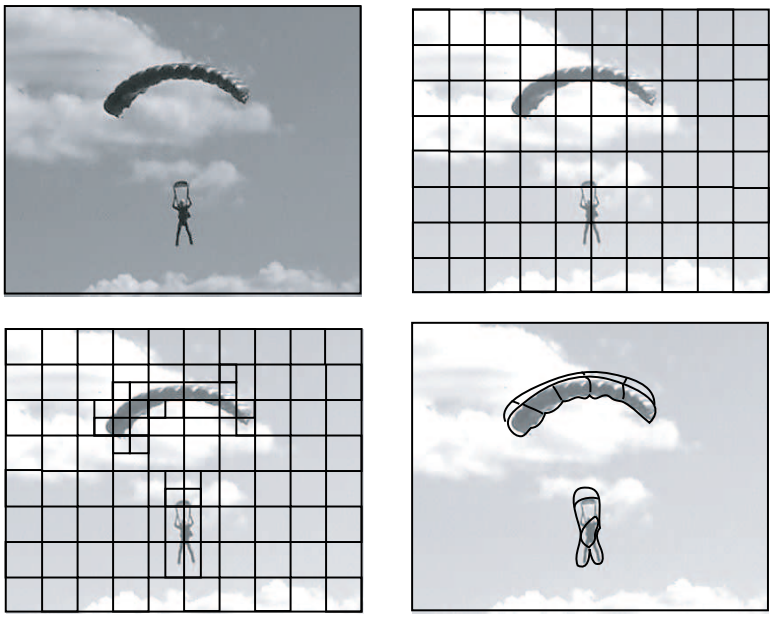

Figure 3.4: Four images showing different units for motion compensation. Top left is the original image. Top right has a fixed block size. Bottom left has variable block size. Bottom right has units closely matching the moving objects.

Mesh-based Models

An extension of DBMA is to implement mesh-based models. The technique forces continuity across block boundaries, and thus yields smoother motion fields.

3.2.3 The Block Matching Problem

The task of finding a good matching algorithm for block based motion compensation can be divided into choosing a search range, search pattern, and matching criteria [15]. Since motion compensation often is the most time consuming process of compressing video, it is important for real-time compression not to choose a slow block matching algorithm. Some algorithms may have a fast average speed, but are slow in a worst case scenario.

Search Range

There are two reasons to limit the search range for the block matching algorithm. By restricting the number of possible matches, there will be a fewer number of distinct motion vectors. This causes the compression of motion vectors to be more effective. The other reason to limit the search length is due to the computational cost of searching possible matches. By doubling the search length the computational time needed for searching for the best match is quadrupled. In addition, the most commonly found matches lie close to the original positions, which further discourage a long search range. When following a fast moving object, a larger search area is needed. This is also the case if the framerate is low.

The maximum displacement of a motion vector need not be the same in the horizontal and vertical direction. Because motion is more likely in the horizontal direction than vertical, rectangular search areas are popular.

Search Pattern

The optimal block matching algorithm has the ability to find the best possible match within a given search radius. The easiest way to find the best match is to search all possible matches within the given search area, and choose the best one. This method is called a full search, exhaustive search, or brute force search. For implementations where speed is not an issue, implementations of the full search algorithms give the best possible result. There are also numerous algorithms for obtaining good results by only checking a fraction of all possible matches.

Which method to use depend on how many comparisons one is willing to try before choosing the best solution. More comparisons are likely to give a better match, but use more computational power. According to [18], two refinements of Three Step Search called New Three Step Search and Four Step Search is among the most suited algorithms for motion prediction in real-time video transmission. Some examples of block matching algorithms [15] follows.

Full Search / Optimized Full Search This approach is the ultimate in finding the best matches, but also the slowest. A full search evaluates every possible match and will always find the best possible match. An optimized full search first does a search on downsampled versions of the images. By comparing the matches found, it is possible to rule out the possibility of the best match being found in some areas. A full search is then done only in those areas where the best match can be found.

Three-Step-Search (TSS) This algorithm first evaluates the center and eight surrounding locations of the original position. The location that produces the best match then becomes the center of the next stage, and the search length is reduced by half. This sequence is repeated three times.

New-Three-Step-Search (NTSS) A refinement of three step search that produce equally good results with a lower average number of evaluated matches. Before starting a *Three Step Search*, the positions bordering the start position are evaluated. If these positions produce a match within a given threshold the rest of the search is canceled.

Greedy Algorithms A greedy algorithm attempts to reduce the number of matching criteria evaluations by shifting the center of a search pattern immediately on finding a vector with a distortion lower than the current center.

3.2.4 Matching Criteria

The best possible match is the one that lead to the highest possible compression. To fully evaluate a match it is necessary to compress all the differences between all candidate blocks, and the blocks to reconstruct. This will consume huge amounts of processing time, and has usually not been considered worth the effort. Instead the usual approach is to calculate the difference by a given *matching criteria*. Here is a short description of some of the most common criterions in use [15].

Mean Absolute Difference (MAD) Corresponding pixels from each block are compared and their differences summed.

Mean Square Difference (MSD) The mean square difference function is similar to the mean absolute difference function, except that the difference between pixels is squared before summation.

Pel Difference Classification (PDC) The Pel Difference Classification (PDC) distortion function compares each pixel of the target block with its counterpart in the candidate block and classifies each pixel pair as either matching or not matching. Pixels are matching if the difference between their values is less than a given threshold. The greater the number of matching pixels the better the match.

3.2.5 Optimization

Starting with basic block-matching, there are several optimizations that speed up the process, or produce matches with higher precision.

Starting Position

Since motion estimation is so computationally expensive any method for finding good matches fast should be considered. One technique is to start a search in a position where a good match is more likely to occur. Start positions can be set to match vectors found in surrounding blocks (spatial dependency), or in previous searches (temporal dependency) [15]. Both spatial and temporal dependency can be exploited simultaneously.

The precision of the calculated start positions are only as good as the vectors it is based on. If the motion vector follows a moving object correctly, it may be used as a starting position for later motion searches. If on the other hand the motion vector does not match the actual moving object in the video, the starting position may give the next search a disadvantage by setting it off in the wrong direction. A motion calculating algorithm may be fooled due to noise, a weaknesses in the matching algorithm, or local minima. Searches should therefore not depend entirely on the calculated start positions.

Edges

Extra care is needed to handle information at the edges of the picture. When tracking a moving object, matches that contain pixels outside the video frame may be considered. The pixels lying outside the video have

to be coded from scratch. Still, if the overall compression is higher by using a block lying partially outside of the frame this is still the best choice. Uncovered areas can be considered to be black (or some other color), or reuse some of the nearby pixels within the superblock. Reusing pixels at the edges is perhaps the most efficient choice since pixels close to each other are likely to be somewhat similar.

Thresholds and Weighting

If the change for a block between two images is smaller than a given threshold, the task of searching for better matches may be canceled. If there is practically no change between the two blocks, searching for better matches is unlikely to give a better match, and the computational expense can be avoided. For a video with little noise, this value can be set very low since an area without movement may change very little. If transmitting a noisy video, there will always be an amount of difference from frame to frame due to the nature of the video. Dirac, a codec created by BBC, performs motion estimation using a set of parameters that can be set to describe the amount of movement in the video being compressed [12].

Sub Pixel Accuracy

The quality of the match can often be improved by interpolating pixels in the search area, effectively increasing the resolution within the search area by allowing hypothetical candidate blocks with fractional displacements. This means that a motion vector may represent a movement with half pixel, or quarter pixel accuracy. Each pixel within the matching block will be constructed using an average of the pixels surrounding the matching position.

Overlapped Block Motion Compensation

Block based compression can produce swift changes at the edges of each block. Overlapped block motion compensation (OBMC) is an attempt to compensate the problem of blocking artifacts [28]. Blocks are usually twice as big, horizontally and vertically, and overlap with all eight neighboring blocks. Due to overlapping, each pixel is the average of four pixels from four different blocks. Overlapping blocks increase prediction accuracy when tracking large moving objects.

Motion Vector Correction

Motion vectors might not correspond to the actual motion in the scene [15]. It is however possible to re-evaluate a calculated vector that may be erroneous. Vectors that do not correspond to its neighboring vectors are candidates for correction. By changing the most erratic vectors and suggesting alternatives, the motion vectors are made smoother. Smoothing techniques can add considerable complexity to a video compression algorithm. It may also cause small objects to be coded badly because their motion vectors might be considered erroneous when they are in fact correct.

3.2.6 Alternative Methods

There are techniques that use a different approach to the block matching problem, two of which are explained here.

Multidimensional Search Spaces

Changes due to illumination variation, object rotation or motion in directions other than that parallel to the camera plane are not compensated by block based motion compensation, described in this chapter. Multidimensional motion compensation schemes attempt to compensate for these additional factors [15]. Deformable block-matching algorithms and mesh-based models are examples that makes use of multidimensional search spaces.

DCT-Based Motion Estimation

US Patent 5,790,686 describes a more efficient method to use DCT coefficients for the block matching, instead of the pixel data. The method brings the computational cost from $O(n^4)$ to $O(n^2)$ for a NxN block. It is targeted at high-end real-time applications. It is described as follows:

> It works on the DCT coefficient of the input image by computing the pseudo phases and applying sinusoidal orthogonal principles.

3.2.7 Entropy Coding

To minimize the cost of transmitting the motion vectors, we want to represent motion vectors with the least number of bits possible. By using statistical data, it is possible to give the more common vectors a short bit-sequence while the less common is given a longer bit-sequence. The

most common motion vector is the vector (0,0), indicating no movement. Short motion vectors are in general more common then longer ones.

Camera movements like panning and zooming cause several adjectant vectors to be somewhat similar. Large moving objects also result in several motion vectors pointing in the same general direction. This can also be used to further compress the motion vectors.

Implementing an algorithm to compress motion vectors and bidirectional signals has been considered outside the scope of this thesis.

3.3 Summary

This chapter gives a description of how bidirectional coding and motion compensation work, and alternative implementations. This thesis will explore the most basic methods for both bidirectional coding and motion compensation.

Only block based motion compensation will be considered from this point on. Sub pixel accuracy has also been considered too complex too pursue any further. For motion compensation the remaining task will be to find a suitable block size, and choose a search pattern, search length, and matching criteria.

As for bidirectional coding, dynamic GOP structures had to go. Dynamic frame sequences is also considered outside the scope of this thesis, as it may prove difficult to implement without being in conflict with the layered approach used in the VSCBN prototype. For bidirectional coding the remaining task will be to find a suitable frame sequence, determine the unit for which a direction code should apply, and choose what information a direction code may carry.

Compression of motion vectors and bidirectional signals are considered outside the scope of this thesis.

Chapter 4

Design

This chapter begins with a more technical description of the parts of the VSCBN prototype that is of interest considering the proposed extensions. The following two sections discuss a scheme for bidirectional coding and motion compensation that can be used for the VSCBN prototype. Two new optimizations are suggested in section 4.3.2. These are made specifically for the VSCBN prototype.

4.1 VSCBN Prototype

In our project a user does not necessarily receive all frames unless that user subscribes to all time-layers. This means that the information in a frame a user does not receive cannot be used as a base for creating a frame that the same user receives. So, to enable subscriptions to alternative frame-rates, a frame sequence must follow certain restrictions. A layer may only use frames from an equal or lower layer as references. The frames must be placed into layers so that all subscription levels produce a video with a constant framerate.

The prototype makes use of the original images from the input stream as reference images i.e, no decoding is done at the sender side. This leads to inaccuracies for the differential coded frames.

4.1.1 Frame-sequence

To achieve temporal granularity (varying framerates) each frame in a GOP is divided into layers where each layer only depend on lower layers for references. In this way each layer is only dependent on lower subscriptions in the temporal dimension. Each additional layer will increase the framerate of the video, and the new frames are added so that the old and new frames are evenly distributed in time, creating a steady flow of images for all subscriptions.

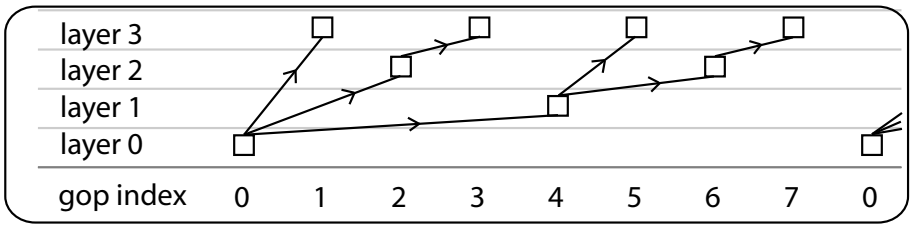

Figure 4.1: How reference frames are chosen using forward prediction only

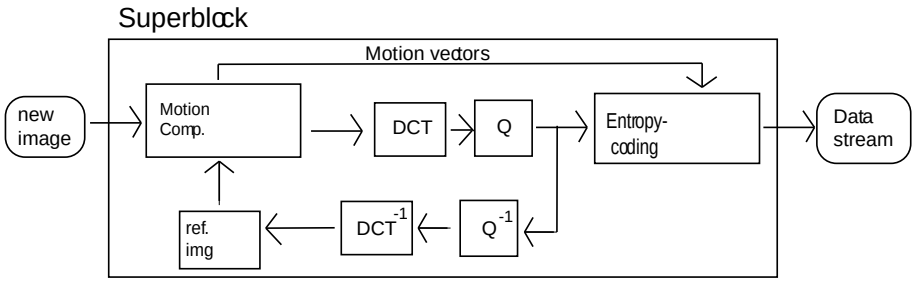

Figure 4.2: An encoder that also decodes data to keep a correct reference image for future temporal compression

The frame sequence used in the current prototype is a simple hierarchically structure seen in figure 2.5 on page 21, and repeated here as figure 4.1 for the convenience of the reader. For each additional temporal subscription level the framerate is doubled by adding a new frame between every existing frame.

4.1.2 Accuracy of Reference Images

For predicted frames, the sender side transmits the differential between an image a receiver already have, and the next frame. Since frames are not compressed lossless there is a small difference between the image a user receives and the originally captured image. In order to compensate for this inaccuracy an encoder may also include a decoder, and use the encoded-decoded frames for references. In this way the encoder keeps a reference image that is an exact replica of what the receiver has. An example of such an encoder is illustrated in figure 4.2.

Although the difference between a compressed image and the original is hardly noticeable, the difference in a video may build up over time if each image is based on a prior image. This is specially true for lower quality video since the highly compressed images differ more from the

original images than less compressed images do.

To avoid this inaccuracy in the prototype without sacrificing granularity in the signal to noise ratio dimension, one would have to decode each frame in every quality a user may subscribe to. Instead of coding the differential between two frames, and storing the data in several quality layers the encoder would have to code each layer separately. This is an issue concerning all layered video formats. Attempts have been made to solve the above problem for a limited number of layers by using a separate correction layer for the base layer [2].

4.2 Bidirectional Coding Scheme

There are alternative ways to add bidirectional coding to the current prototype. How frequent bidirectional frames should occur, and how they are distributed are two aspects. There is also the choice whether to allow combinations of references as described in section 3.1.5, or only use strict forward or backward references. The chosen frame-sequence effects areas like memory usage, processing speed and end-to-end delay. The layered structure that enables granularity in the time dimension must be preserved.

In this thesis, support for only one predefined frame-sequence was originally considered. After trying different sequences it seemed that each sequence had some advantages over the others, and so a more general approach was needed.

References that involve bringing information from a future frame to a current frame will throughout this text be called a future reference. References bringing information from a prior frame will be called a past or former reference.

4.2.1 Criteria for Frame-sequences

The VSCBN prototype uses a binary tree structured frame-sequence illustrated in figure 4.1 with eight pictures in each group of pictures (GOP). Before looking at specific bidirectional frame-sequences this section will try to identify what qualities a frame-sequence should incorporate.

Independent Group of Pictures (GOP)

In the current implementation, the client drops a GOP if it is incapable of processing frames at the rate at which they are received. Hopefully, scalability on the client side will perform more seamlessly than this in a future release. There may be other reasons to drop a number of frames,

in example synchronization with audio or to limit the effect of an error to only one GOP.

Length of References

The length of a reference refers to the number of frames between a frame and its reference frame. Longer references generally produce larger differentials since similar frames are more likely to be found close to each other.

Length of Differential Chain

A frame coded as a differential to another frame may be coded more efficiently, but will contain errors if the reference frame contains errors. A frame coded against a reference frame which itself is coded against a third frame, contain errors from both its reference, and the top level reference. The number of references a frame may directly or indirectly inherit information from will be described as *chain length*.

Avoiding lengthy chains lead to the use of longer references, which again lead to less efficient differential coding. To create a combination of error resilient code and efficient compression, a balance must be made between minimizing chain lengths and minimizing reference lengths.

Reference Frames

in addition to compression rate for reference frames three considerations can be taken into account. First of all, bidirectional coding creates a delay that is proportional to the length of the longest future reference. Secondly, all frames that can be used as a reference must be kept in memory. For example, the frame sequence used in MPEG-2 minimizes the number of frames kept in memory by not using B-frames as references. Thirdly, by adding a second reference to a frame, more processing is required at the encoding side.

4.2.2 Alternative Frame-sequences

Following are some considered frame-sequence structures with comments on expected performance and resource consumption.

Preserving Independent Group of Pictures

For a GOP to be truly independent no references can cross between two picture groups. In effect that means that no references may cross from

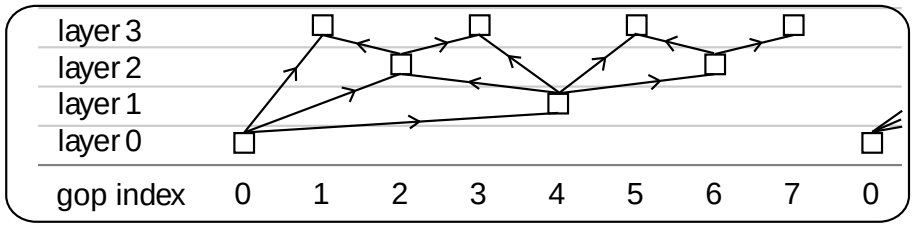

Figure 4.3: Bidirectional coding preserving independent GOPs

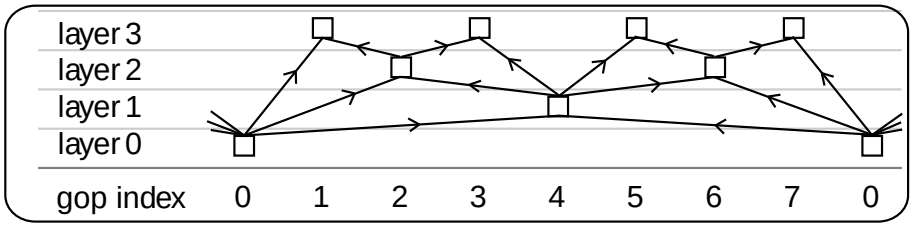

Figure 4.4: Frame structure using only I and B-frames

one GOP to another. Figure 4.3 is an example of a frame-sequence that preserves self contained GOPs.

I and B-frames Only

When replacing all P-frames in the original sequence with B-frames, both former and future I-frame are used to code frame number four simultaneously. This can be seen in figure 4.4. Because of this, frame zero and eight must be kept in memory simultaneously. The first I-frame can only be removed from memory once the dependent fourth frame has been fully encoded. This sequence is not supported by the extended prototype due to its extra complexity.

A Combination of I, P and B-frames

By changing frame four to a P-frame, holding two I-frames in memory simultaneously can be avoided. The delay between capture and transmission is also reduced. The frame-sequence is illustrated in figure 4.5.

Structure Commonly Used by MPEG

The frame-sequence in figure 4.5 is quite similar to one of the most common sequence used by MPEG-2. The main difference between the two is that MPEG never uses B-frames as references. The MPEG approach

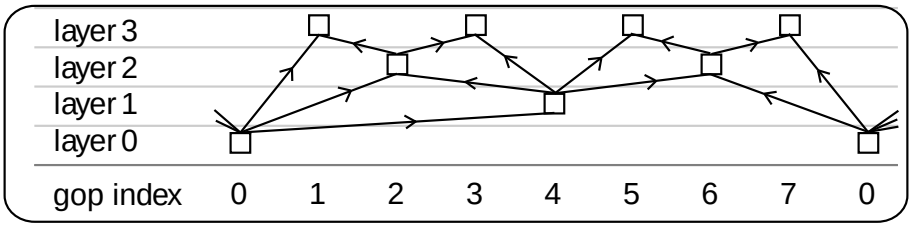

Figure 4.5: Frame structure using a combination of I, P and B-frames

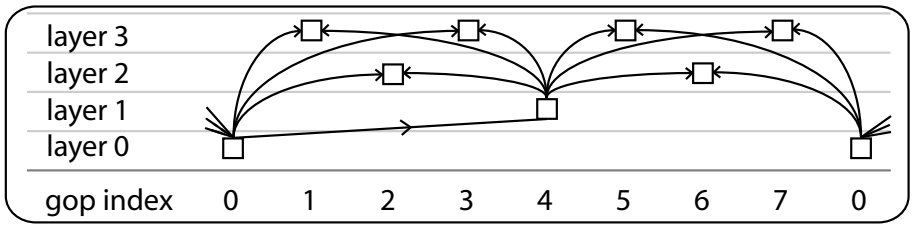

Figure 4.6: Alternative frame sequence that reduce chain length, at the cost of longer references

use longer references, but reduce chain length. Figure 4.6 show a legal MPEG-2 sequence that preserves independent time layers, and thus can be used by the prototype.

4.2.3 References

A frame-sequence only describes which frames that may be used as references, not what information a reference may include or what units references are associated with.

Information Contained in a Reference

For simplicity, only forward or backward references are allowed, no combinations. The prototype will use a single bit to signal a forward or a backward reference for each macroblock. Figure 4.7 show all possible references for a group of pictures, and show an example of the references actually used on a macroblock level.

Frequency of References

A reference can be sent per macroblock, superblock or for each frame. It is possible to limit a subscription to a given number of superblocks, so even if a reference apply to an entire frame a reference would have to be

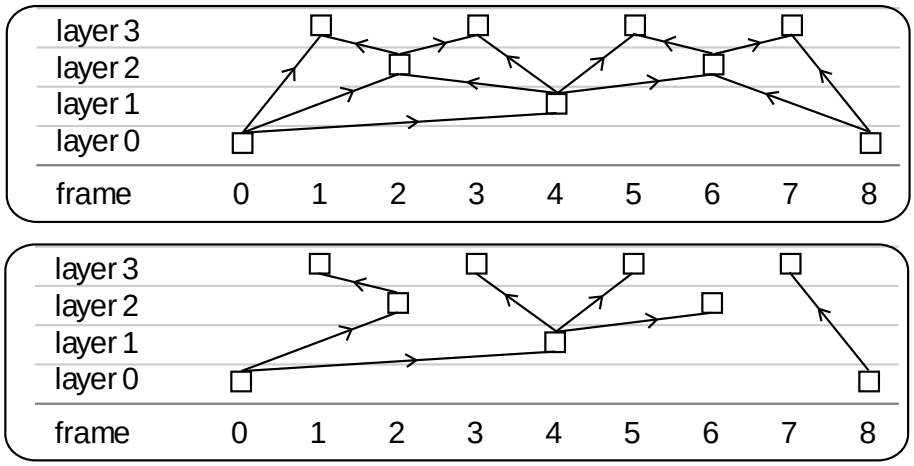

Figure 4.7: Top: All possible references using bidirectional coding. Bottom: An example of references actually used on a macroblock level.

included in each superblock. This limits the choice to macroblocks or superblocks.

If a superblock is likely to contain both information that is only present in a future frame, and information only present in a former frame, references should be sent per macroblock. If not, a reference per superblock is preferable due to less overhead.

Each reference contain little data (one bit) compared to for example, motion vectors (16 bits). The choice has therefore been to use one reference per macroblock without investigating the second option any further.

4.2.4 Transmission Order

At the encoder side a frame cannot be encoded before its depending reference frame has been captured. Similarly at the decoder side a frame cannot be decoded unless its depending reference has been received. To minimize buffer usage at the receiving side frames are transmitted out of order so that every received frame is the next one to be decoded. This is the same technique used in MPEG-2.

4.2.5 Efficiency Gain

The true effect of bidirectional coding depend on the video source. The amount of compression on average is linked to the number of possible references and their lengths. We believe short references produce better results, and by introducing two possible references more suitable

matches can be found. But the goal is not only to maximize compression, but also to provide stability and reliability. One error should not affect more than a limited part of the video.

There is a large disadvantage compression wise in the original prototype by not using bidirectional coding. Imagine in figure 4.1 on page 40 that a large change occurs between frame one and frame two followed by small changes in the next three frames. At the macroblock level the large differential is transmitted for frame two, but cannot be reused by the following frame since this frame is on a lower temporal layer then frame one. The same applies to frame four, which has a higher temporal layer then frame one, two and three. This means that the large difference occurring between frame one and two will have to be transmitted three times. Bidirectional coding will help to minimize the amount of redundant code in this example.

4.3 Motion Compensation Scheme

Motion compensation is provided with two separate search algorithms, and two matching criteria. More can easily be added at a later stage. Two optimizations are also suggested. The first is a speed optimization that allow the search algorithm to start in a position more likely to contain a good match. The second is a possible more efficient method to store the generated motion vectors. It involves using separate Huffman generated VLC tables that vary with the likely lengths of motion-vectors.

4.3.1 Search Algorithm

The purpose of the search algorithm is to find good matches in a time period that is appropriate. The most effective way to design a motion compensation scheme is to include as many search algorithms as feasible, and choose the one offering the best matches, or perhaps one that provide good matches in less time.

Two block matching algorithms were considered implemented to validate the effectiveness against each other. Two separate matching criteria were also implemented for the same reason.

Motion estimation is done in the luminance specter only, since most of the video signal consist of luminance data. For complexity reasons, this is the normal approach. An example of using both the luminance and chrominance specter is discussed in [1].

Search length determine the speed of the search algorithm and is set specifically for each search.

4.3.2 Matching Criteria

The approach for deciding which matching criteria to use is to implement as many as feasible, and use empirical testing to decide the most efficient one. According to [15] the various matching criteria produce nearly equal results when it comes to compression ratio.

4.3.3 Using Small Vectors to Generate Start Positions

The following strategy uses temporal dependency to generate start positions. It is only effective for hierarically structured frame-sequences, and this is probably the reason this technique has not been found described elsewhere by the author.

Motivation

A user may choose not to subscribe to every frame. Hence a frame may have to use reference frames that are further apart in time than otherwise possible. When using references that are so far away in time an object may have moved a considerable distance. To detect a large movement a larger search area is needed, which results in slower motion detection. By setting the start position of the search to a position where the match is more likely, good matches can be found in less time.

Method

Searches using frames next to each other are executed first. The results from these calculations can be used to calculate likely vectors for longer searches. In many cases a movement continues in the same direction from frame to frame, for example a camera panning a static background. If this is the case, vector u1 and u2 in figure 4.8 should be of approximately equal size, pointing in opposite direction.

Vector v1 should be twice the size of u1 and u2 since it covers movement from the first to the third frame instead of from the first to the second. More precisely, the combined movement of u1 and u2 should be a good approximation of v1. (u2 must be subtracted from u1 since it is a backward prediction). By using u1 - u2 as a starting point for v1 a match can be found faster then otherwise possible. In the same manner v1 and v2 can be used to generate a start position for vector w1. Note that motion compensation is done at the server side, so this will not cause information to flow between the temporal layers.

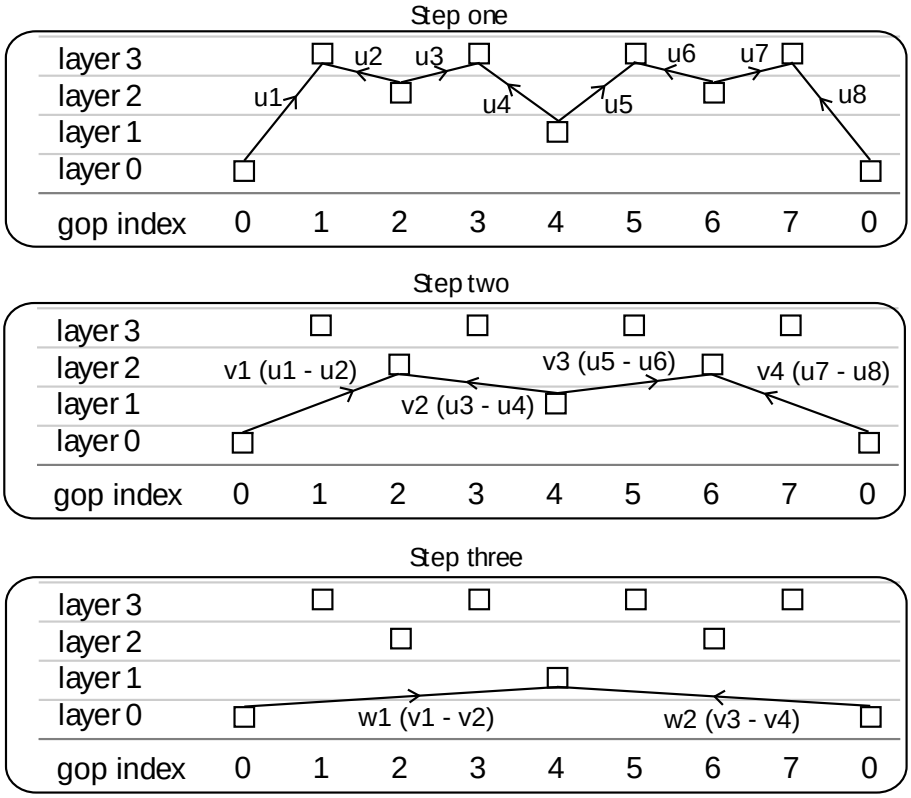

Figure 4.8: Three steps for calculating bidirectional motion vectors in a group of pictures. Vectors in parentheses can be used as start position.

Expected Results

Implementing this method should in most cases provide good matches in less time. One problem of calculating short reference motion vectors first in a live transmission is that the long reference frames are transmitted first. This is because the short reference frames are dependent on the other frames for decompression. By calculating short references first the transmission of longer references which are needed first at the client side are delayed, which results in a slightly increased overall delay.

4.3.4 Using Several Variable Length Code (VLC) Tables

Motion estimation between frames next to each other contain the least movement. To encode motion vectors efficiently it might be a good idea to use separate Huffman generated VLC tables for frames likely to contain long and short motion vectors.

4.4 Summary

This chapter has discussed design related issues of the extensions to the VSCBN prototype.

The suggested bidirectional coding scheme calls for user specified frame sequences. The sequences must follow certain restriction. Three of the four frame sequences described comply to these restrictions. The effectiveness of these sequences will be verified by empirical studies in chapter 6. A single bit will be used to signal the use of a future or former reference.

The motion compensation scheme simply suggest adding as many search algorithms as possible and use empirical tests to determine the most useful. Two optimizations have been suggested for the motion compensation algorithm. One that will speed up the block matching algorithm and another to compress the motion vectors more efficiently. The suggested techniques are designed specificly for the VSCBN prototype and may not be of any relevance to other projects.

Chapter 5

Prototype

This chapter gives a description of the prototype and how the additions of motion compensation and bidirectional coding are implemented. Bidirectional coding changes a lot of the structure of the prototype and there are a limited set of alternatives. Extra buffers and queues were added, but the overall design remains mostly unchanged. For motion compensation, the existing design was changed very little. Making it work processing wise, has been more important then making it work efficiently in terms of speed and memory usage.

The prototype has been developed on and for the Linux operation system. The code is written in java 1.4, with the addition of some Java native C code used to provide an interface to Video4Linux. The implementation is published under GNU public licence.

5.1 Motion Estimation

Temporal layering and regions create extra challenges for motion compensation implementations. Low temporal subscriptions provide a stream where an object move a greater distance between two following frames. There is also a greater number of edges due to region coding. Similar to image boarders, no information may cross between two superblocks.

A new class named *MotionEstimation* calculates motion vectors. It compares the block to compress to a reference image containing pixel data from the referenced super-block. The vector of the best match is returned. It includes two alternative search pattern, and two alternative matching criteria.

Search Pattern

Two search methods have been implemented. A full search to give the best possible results, and the faster tree step search. A “greedy” search

method was also implemented, but has not performed as well as tree step search. Full search has been used to validate the effectiveness of the other techniques. Search length is set as a separate argument for each search, and can be adjusted based on the number of frames between a frame and the associated reference.

Search Length

The search length can be set for the full search algorithm. A separate value is set for each temporal layer. In this way a search length can be set that reflect the distance to the referenced frame. If, for example, there are several frames between the frame to compress and the reference frame, it is assumed that an object will have moved a greater distance and a longer search is required. For a real-time compression system it may be a good idea to adjust the search lengths dynamically to control the computational cost of motion compensation. A search length for the three step search cannot be set due to the nature of the algorithm.

Matching Criteria

Two search criteria have been implemented: Mean Absolute Difference (MAD) and Mean Square Difference (MSD). MSD is supposed to be slightly better, but is also the most CPU intensive. The difference between using one over the other should be too small to have a noticeable impact on compression rate.

Threshold

A threshold value can be set to abort further searching once a match within the given threshold has been found. In the current implementation the same value is used regardless of matching criteria.

Motion Estimation with Sending Threshold

The amount of difference between a block and the referenced substitute block is stored by the motion compensating algorithm. If this value lie below a certain threshold, correctional data is not transmitted. The video data being transmitted by the prototype is listed in table 5.1.

5.2 Bidirectional Coding

There are two methods in the *MotionEstimation* class. One for forward prediction and one for bidirectional prediction. The bidirectional

Movement	Change	Data transmitted
No	No	Nothing
Yes	No	Motion vector (X,Y) + no data signal
No	Yes	Motion vector (0,0) + DCT data
Yes	Yes	Motion vector (X,Y) + DCT data

Table 5.1: The data transmitted depend on whether there is sufficient change from a priorly transmitted frame, and whether there is movement or not.

method does two searches, one in each direction and returns the best of the two results. Bidirectional coding requires frames to be transmitted out of order.

Setting the Frame-sequence

Frame-structures are set using two arrays, one for future references and one for past references. The position in the array represent the frames of a GOP. The value indicate what frame to use as a reference. The value -1 is used to indicate no reference. Most frame-sequences described in the design chapter are supported by the prototype. An example for the frame-sequence illustrated in figure 4.3 for bidirectional coding is:

```
forward references  = {-1, 0, 0, 2, 0, 4, 4, 6};
backward references = {-1, 2, 4, 4,-1, 6,-1,-1};
```

Bidirectional coding is disabled by setting all backward references to -1. The sequences are currently constant values in the code. A more flexible solution would have been to give the sequence as an argument to the encoder.

Queues

Two queues control the transmission order, and delay frames that cannot be encoded straight away. It is also possible to add a third queue that controls the encoding order and time of execution.

Waiting Queue A *waiting queue* holds a value containing the GOP index of captured frames, and the index of its future reference. When the corresponding future reference is captured, the frame is moved from the waiting queue to the execution queue.

Execution Queue An *execution queue* may be implemented that sets the execution order and/or delays the compression of a frame. The priority of the execution order may be based on a *short reference first* strategy to generate start-positions for later motion compensation, or alternatively be based on transmission order to create minimum delay. The current implementation simply execute all frames whenever they are ready.

Transmission Queue The *transmission queue* delays the transmission of some frames so that the client side can decode each frame as they arrive. The transmission order is generated from the currently used frame-sequence. For the frame-sequence in figure 4.3 the transmission order would be: (by GOP index) 0, 4, 2, 1, 3, 6, 5, 7.

5.3 Program Flow

Following is a description of the server side signal handling process from capture to transmission. It is presented in the executed order.

5.3.1 Video Sources

The prototype accepts input video streams in a raw format from both camera and files. FFmpeg has been used to convert video files from its original format to the uncompressed raw format our prototype expects. Depending on the video source a conversion of color model and subsampling of color layers may be required.

Conversion of Color Model

If needed, the prototype does a conversion from RGB to YCrCb. Colors are converted from RGB to YCrCb by the following computations. This conversion is currently done in software, but it is possible to delegate the task to specific hardware to increase processing speed.

```
Y  =  (0.257 * R) + (0.504 * G) + (0.098 * B) + 16
Cr =  (0.439 * R) - (0.368 * G) - (0.071 * B) + 128
Cb = -(0.148 * R) - (0.291 * G) + (0.439 * B) + 128
```

Subsampling of Color Layers

If needed, the prototype performs horizontal and vertical (4:2:0) decimation on the chrominance data. To achieve this the prototype organize each frame into groups of four blocks, each containing 8x8 pixels, defining a 16x16 pixel *macroblock*. The luminance data for a macroblock is

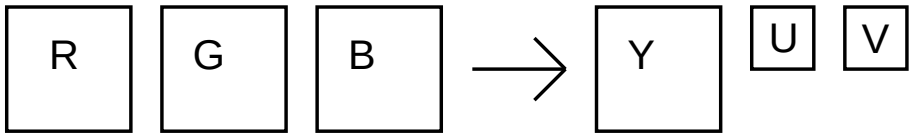

Figure 5.1: Color conversion and subsampling

represented by four 8x8 luminance blocks, while the decimated chrominance data for the entire macroblock is reduced to two 8x8 chrominance blocks. Figure 5.1 illustrate color conversion and subsampling of the chrominance data.

5.3.2 Compression

Motion compensation and bidirectional coding provide a modeled image of the frame that is being compressed. If the remaining difference fall beneath a given threshold it is ignored. If not, it is rectified by using discrete cosine transform (DCT).

Motion Compensation

To minimize the differential code, instead of using the pixels at the same positions in the reference frame, a better match can be found by shifting the candidate block horizontally and vertically.

Example: Frame one does forward motion estimation with frame zero as a reference. The best match for a specific macroblock is found in the reference frame two pixels to the left of the blocks original position. The match in this example has a distortion rate of 1,541 using *Mean Square Difference*.

Bidirectional Coding

The motion compensation procedure is executed twice. Once for a forward reference, and once for a backward reference.

Example: After frame two is captured motion estimation is again calculated for frame one, this time using frame two as a reference. The best match found this time is located three pixels to the right and have a distortion rate of only 268. A motion vector (3,0) is stored along with a signal to show that the block in frame two was used.

Sending Threshold

The block to compress is compared to the best match found in either future or former frame. The difference is only transmitted if it is considered noticeable.

Example: If a distortion rate of 268 fall under the threshold for updating, a *no data* signal replaces the DCT data for the macroblock .

Discrete Cosine Transform and Entropy Coding

If there is a significant change the difference is encoded using DCT. The resulting coefficients are quantized, effectively removing unnoticeable information. Next, the coefficient values are read in a zigzag pattern, and separated into bitplanes, which again are stored using a variable code length table.

5.3.3 Transmission

The content based network makes sure the notifications are sent to the correct subscribers. Each notification carry attribute fields identifying the notification and a video data payload. The signals may be unicasted or multicasted across the network depending on the settings for each network node.

Notifications

A *Siena* notification consist of a number of attributes containing an identifying character string and a value. In the video stream each notification contains a set of attributes describing the type of the notification and a payload attribute containing a chunk of compressed video. All data not used to identify the notification should be encapsulate by this data chunk to prevent extra overhead by using more attributes then necessary. Currently used elements are listed in table 5.2.

Binary Video Element (blob)

Video data is contained in a *binary video element* (blob). The data in most blobs contribute to enhanced quality in one of the video dimensions. The blob that carries the base layer in all quality dimensions have an additional payload of motion vectors and direction signals. The blob also carries the number of bitplanes present. Any notification may contain a new quantizer value if the qescape bit is set. A listing of elements can be seen in table 5.3.

Attribute	Value
sid	Stream identifier
row	Horizontal position
col	Vertical position
ql	Quality layer
tl	Temporal layer
f	Color: luminance or chrominance
sq	Sequence number
time	Timestamp
gopi	Group of picture index
blob	Binary data payload

Table 5.2: Structure of a video notification

Condition	Description	Size
base layer only	numberof bitplanes	4 bits
always	qescape	1 bit
qescape == 1	quantiziser value	6 bits
base layer, B-frames	direction bit	1 bit
base layer, P/B-frames	Motion vectors	16 bits
always	video data	n bits

Table 5.3: Structure of the blob attribute of a video notification

5.4 Summary

In this chapter the implementation of motion compensation and bidirectional coding has been described. It also gives a description of the program flow presented in the executed order.

A motion compensation class implements the *Full Search* and the *Threestep Search* search patterns. It is also possible to choose between two matching criteria. Some search algorithms may abort further searching once a match within a given threshold is found. Matches that produce a differential small enough to fall beneath a sending threshold cause no DCT data will be sent. In the case where there is no movement and the block is considered not to have changed, no data at all is sent.

The frame sequence is set by specifying two arrays containing the referenced frame indexes. A waiting queue hold the indexes of frames waiting for a future frame to be captured, while a transmission queue delay the transmission of some frames to make sure the clients receive the frames it the correct order. An execution queue may also be implemented to set the order of which frames are to be encoded.

Chapter 6

Empirical Results

The extended prototype has been tested with commonly used video clips using the existing codebase as a baseline for comparison. The tests have been conducted in three phases. In the first phase the extended prototype is measured against the original VSCBN prototype. The following phase measure the effect of motion compensation and bidirectional coding individually, while the last phase measure the combination of the two techniques. All measurements were done at the video receiver side.

6.1 Test Conditions

The tests in this section were carried out to measure how much the differential data is reduced when applying motion compensation and bidirectional coding. Motion compensation and bidirectional coding add two types of meta-data to the video stream, motion vectors and direction signals. To calculate the differential data, the bits required for carrying these meta-data have been subtracted from the total amount of video data.

Earlier tests of the prototype [9] has used a *quantizer scale* (qscale) value of four. The qscale value determines the quantization of the DCT coefficients. The tests here use the same qscale value to achieve comparable results.

The motion estimating algorithm uses a threshold value to abort further searching once a sufficiently good match has been found. This value has been set to match the sending threshold value used for transmitting correction data. If the amount of correction lie within the sending threshold, no differential data will be sent. An even better match from the motion estimation algorithm will therefore not effect the compression rate and will in these tests be considered a waste of effort. The sending threshold in the VSCBN prototype was set to 128 with a note to rethink the value.

For searches in the first temporal layer a search length of 10 is used. For the second temporal layer the length is 16, and for the third layer the length is 24. The lengths are set to reflect the distance between a frame and its reference. No thorough testing has been conducted to verify the effect of changing these values, but the author believes that by increasing the search lengths very little will be gained.

To tests the extended prototype against the original VSCBN prototype a dummy function was created that simply return the same reference that would have been used in the original prototype. By also using a forward predictive frame sequence the differential data generated should be equal for the two runs. By comparing the two results the amount of meta-data can be verified.

6.2 Video Clips

The video clips used for testing were downloaded from the University of Missouri-Columbia [17]. The clips are commonly used for testing video compression. Out of several video clips four clips were chosen, two containing much movement, and two containing very little movement. All videos contain uncompressed data in the PAL CIF format measuring 352 x 288 pixels. Each video frame was divided into six superblocks, each containing 176 x 96 pixels. The video clips used are called *Foreman*, *Coastguard*, *Container* and *News*.

Foreman - a freehand captured video clip that contain much movement and some noise.

Coastguard - moving camera from a solid mounted tripod. The clip contains much movement, and there is continuously shifting water.

Container - video clip without camera movement capturing a slow moving container ship on a calm sea.

News - video clip that contain little noise and where movement is limited to small areas of each frame.

For the initial tests, and for measuring the effect of bidirectional coding and motion compensation the *Foreman* clip was used. In the final tests all four video clips were used.

The first 90 frames of each video were used to generate statistics. It should be noted that the first 90 frames in the extended prototype may not correspond with the 90 first frames in the VSCBN prototype since frames can be sent out of order. The test results here will not be affected by this since individual results are presented for each temporal layer.

Foreman video clip				
Encoder	**Temporal layers (bits/pixel)**			
	Layer 0	Layer 1	Layer 2	Layer 3
VSCBN prototype	1.67	1.48	1.31	0.96
Extended prototype	1.67	1.54	1.37	1.03
Difference	0.00	+0.06	+0.06	+0.07

Table 6.1: Test results for the original prototype, compared to the modified version of the prototype with new features replaced with dummy functions.

6.3 Test Results

The initial tests compare the extended prototype to the original VSCBN prototype to verify that the extended prototype has not been degraded by the update. It also measure the amount of meta-data added to the stream to verify that the measured amount match with predictions.

The following test results measure the effect of motion compensation and bidirectional coding independently. Various settings were tested to find the most efficient ones.

Finally, the combination of motion compensation and bidirectional coding was tested using the most efficient settings from prior tests. The final tests were conducted using four video clips with various levels of noise and movement. Each video was tested with various subscriptions in the video quality dimensions.

6.3.1 Initial Tests

An extra overhead is introduced by adding motion vectors and direction signals to the stream. This amounts to 16 bits for the motion vectors, and a single bit for direction signals per macroblock. The introduced overhead is therefore expected to be 0,0625 bits/pixel ($(16 + 1)/16^2$).

Tests have been conducted comparing the updated prototype with a dummy function that return a match at position (0,0) in a former reference, and the original version using the *Foreman* video clip. The results, seen in table 6.1, show that the average amount of meta-data sent per pixel is roughly the expected 17 bits per macroblock, or 0.0625 bit/pixel. Temporal layer zero remains unchanged since it contains I-frames only.

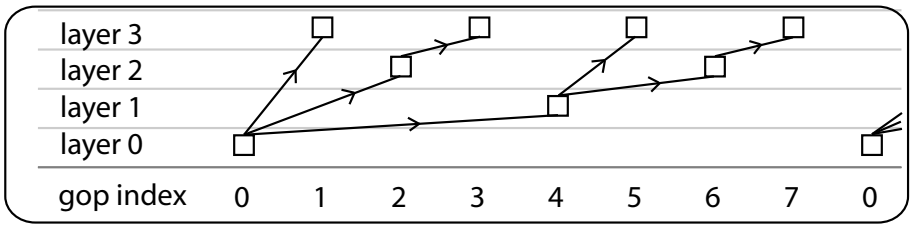

Figure 6.1: Tested frame sequence - Forward Prediction (FP)

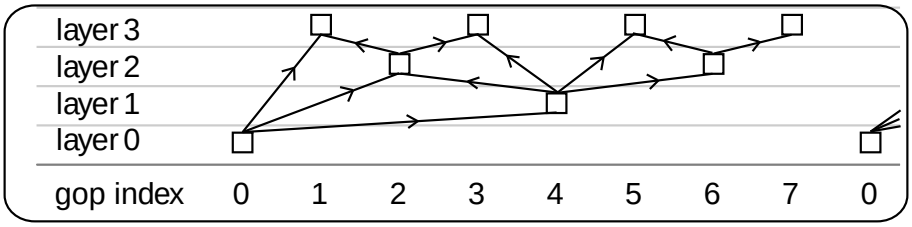

Figure 6.2: Tested frame sequence - Independent GOPs (IGOP)

6.3.2 Bidirectional Coding

Bidirectional coding is a technique that is effective if parts of a video frame are only present in a future frame. The frame-sequence used is a factor that affects the compression rate.

The frame sequences tested were illustrated in figure 4.1, 4.3, 4.5 and 4.6, starting on page 40, and repeated here for the convenience of the reader as figure 6.1, 6.2, 6.3 and 6.4.

Results

As seen in table 6.2, the compression rate on temporal layer zero and one remain unchanged. This is because bidirectional coding is not applied to

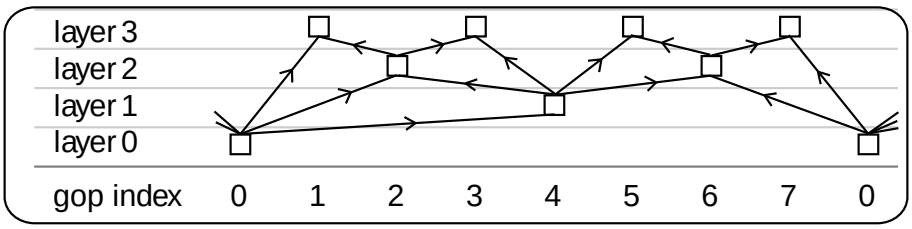

Figure 6.3: Tested frame sequence - Bidirectional Prediction (BP)

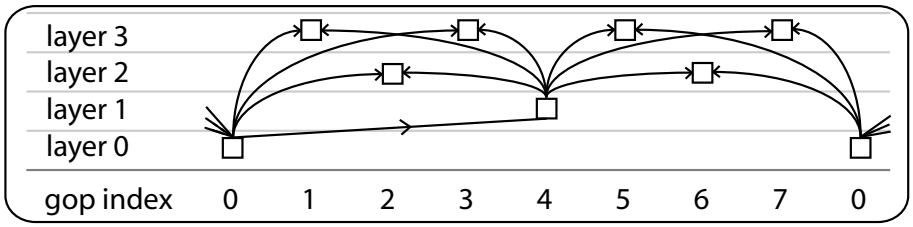

Figure 6.4: Tested frame sequence - MPEG style (MPGS)

"Foreman" video clip				
Frame sequence	**Temporal layers (bits/pixel)**			
	Layer 0	Layer 1	Layer 2	Layer 3
FP, figure 6.1	1.67	1.48	1.31	0.96
IGOP, figure 6.2	1.67	1.48	1.18	0.86
BP, figure 6.3	1.67	1.48	1.13	0.85
MPGS, figure 6.4	1.67	1.48	1.13	0.93

Table 6.2: Test results for bidirectional coded video using alternative frame-sequences. Differential data only on layer 1-3.

these layers by any of the tested frame-sequences.

Layer two consist of frame indexed two and six. Frame-sequence BP and MPGS use the exact same references and therefore have the same performance. IGOP use independent GOP's, and so only frame indexed two is bidirectional. With only half of the frames on layer two bidirectional the compression gain is also halved compared to BP and MPGS.

Layer three consist of the odd numbered frames: one, three, five and seven. IGOP and BP use the same references for most of the frames in this layer, and have comparable results. Due to the longer references used in MPGS this frame sequence produce a much lower compression ratio.

The chain lengths are shorter in MPGS, which should produce better image quality, but BP should produce quality equal to the original VSCBN prototype. In the final tests, the BP frame-sequence will be used because of the compression ratio.

6.3.3 Motion Compensation

As described in section 5.1, the motion estimation algorithm has been implemented with two variable settings. The search method can be set to either *full search* or *threestep search*. The search criteria can be either

"Foreman" video clip					
Method	**Criteria**	**Temporal layers (bits/pixel)**			
		Layer 0	Layer 1	Layer 2	Layer 3
No search	-	1.67	1.48	1.31	0.96
Full search	MAD	1.67	0.73	0.62	0.49
Full search	MSD	1.67	0.73	0.62	0.49
Threestep	MAD	1.67	1.29	1.15	0.91
Threestep	MSD	1.67	1.09	0.91	0.70

Table 6.3: Test results for motion compensated video using various settings. Differential data only on layer 1-3.

Mean Average Difference (MAD) or *Mean Squared Difference* (MSD).

Results

As seen in table 6.3 the full search method (row two and three) improves compression very well. For full search the results are equal for the two matching criteria MAD and MSD. The threestep search (row four and five) contribute to compression, but the performance is, as expected, lower than the full search algorithm.

There is a very big difference between the two matching criteria MAD and MSD when using three-step-search. It was at first believed that this was a mistake in the code, but after examining the code this seemed unlikely. It can only be concluded that for a good match, the two matching criteria perform equally well. MAD however does not evaluate matches that lie close to a good match as well as it should. This sets the matching algorithm off in a wrong direction, which lead to less optimal results.

Nevertheless, a full search will always perform better then a sub optimal search, and so the final tests will use this search method. As for matching criteria, MAD will be used since the performance is equal to MSD, while being less computationally complex.

6.3.4 Final Tests

During the final tests both motion compensation and bidirectional coding were used with the most efficient settings from prior tests. Four test samples were used to see if the compression gain varies with the characteristics of the video clip. Each video was tested with various user subscription levels in the video quality dimensions.

Meta-data and Correction Data

The original VSCBN prototype aims to be a compression algorithm that scales from low to high bandwidth. Inter frames consist of two types of data, meta-data and correction data. One way to produce a low bandwidth subscription is to cut away some of the least significant bits of the correction data. This is done by not subscribing to all layers in the *signal to noise* dimension. By doing so, the meta-data will obtain a large portion of the overall signal, and will therefore contribute less to compression. This leads to a compromise between sending too much meta-data for low subscriptions, and too little for high bandwidth subscriptions. Since the meta-data part of the video stream is not included in the test results of this thesis, it is difficult to determine what is an appropriate amount of meta-data. Varying subscriptions in the temporal dimension is unaffected by this since subscriptions to low frame-rates also cut away the meta-data associated with the missing frames.

Subscription Settings

The difference in compression rate is expected to change unevenly for different client subscriptions. There are too many combinations to test all of them, so only a few combinations were selected.

Instead of comparing different subscriptions in the temporal dimension, the test results are displayed for the various temporal layers. The tests have been conducted with and without color information, combined with the highest and lowest level in the *signal to noise* dimension. Different regions of interest is not a part of the testing because the introduced meta-data transmitted is proportional to the number of regions subscribed to.

Results

Table 6.4 and 6.5 show the results for the two video clips containing much motion, while table 6.6 and 6.7 show the results from the low motion video clips.

The improved compression ratios are in the text presented as a percentage improvement compared to the dummy function. To calculate the total amount off differential data within a group of frames the result from each layer must be multiplied with the number of frames in that layer. Total differential data for a GOP is therefore $Layer1 + (Layer2 * 2) + (Layer3 * 4)$.

The best results came from the *Foreman* clip presented in table 6.4, with a reduction of the differential data 60% to 70%. The *container* clip

"Foreman" - dummy algorithm					
Settings		**Temporal layers (bits/pixel)**			
Color	Quality	Layer 0	Layer 1	Layer 2	Layer 3
Yes	High	1.67	1.48	1.31	0.97
Yes	Low	0.68	0.39	0.33	0.21
No	High	1.33	1.37	1.23	0.92
No	Low	0.49	0.35	0.30	0.19
"Foreman" - extended prototype					
Color	Quality	Layer 0	Layer 1	Layer 2	Layer 3
Yes	High	1.67	0.74	0.46	0.37
Yes	Low	0.68	0.15	0.09	0.07
No	High	1.34	0.67	0.43	0.34
No	Low	0.49	0.12	0.07	0.05

Table 6.4: Test results enabling both bidirectional coding and motion compensation. Video clip is *Foreman*, containing much movement. Layer 1-3 show differential data only.

(table 6.7) was the clip that was the least compressed. Differential data was reduced by 16% to 20%.

Compression is higher for low subscriptions in the *signal to noise* dimension. For the *Foreman* clip the lowest subscription compression ratio was reduced by 70%, while the highest only had a 60% reduction.

The compression rate is about equal for subscriptions to chrominance and luminance part of the video signals. It is however interesting to see how little information is needed to provide color signals. The author expected about one third of a color video to be chrominance data, based on the sub-sampling rate. The results here indicate that the chrominance data obtain less data then expected.

The combination of the two techniques roughly equals the sum of the gained compression rate for the two techniques individually. For the *Foreman* clip with color and high quality, bidirectional coding reduced the video data with (bits/pixel) 0.0 on layer one, 0.18 on layer two, and 0.11 on layer three (from table 6.2). The effect of motion compensation on the same clip is for same layers are 0.75, 0.69 and 0.47 (from table 6.3). The combination of the two techniques result in a compression gain of 0.74, 0.85 and 0.60 (from table 6.4). This is almost exactly the same as the sum of the two compression techniques individually, which is 0.75, 0.87 (0.69+0.18) and 0.58 (0.47+0.11).

"Coastguard" - dummy algorithm					
Settings		**Temporal layers (bits/pixel)**			
Color	Quality	Layer 0	Layer 1	Layer 2	Layer 3
Yes	High	2.07	2.10	1.98	1.75
Yes	Low	0.71	0.55	0.48	0.32
No	High	1.77	2.04	1.94	1.73
No	Low	0.53	0.53	0.46	0.31
"Coastguard" - extended prototype					
Color	Quality	Layer 0	Layer 1	Layer 2	Layer 3
Yes	High	2.07	1.50	1.17	0.83
Yes	Low	0.71	0.30	0.17	0.10
No	High	1.77	1.47	1.15	0.82
No	Low	0.53	0.28	0.16	0.09

Table 6.5: Test results enabling both bidirectional coding and motion compensation. Video clip is *Coastguard*, containing much movement. Layer 1-3 show differential data only.

"News" - dummy algorithm					
Settings		**Temporal layers (bits/pixel)**			
Color	Quality	Layer 0	Layer 1	Layer 2	Layer 3
Yes	High	1.55	0.40	0.28	0.17
Yes	Low	0.65	0.14	0.10	0.05
No	High	1.18	0.33	0.24	0.14
No	Low	0.46	0.11	0.08	0.04
"News" - extended prototype					
Color	Quality	Layer 0	Layer 1	Layer 2	Layer 3
Yes	High	1.55	0.35	0.20	0.12
Yes	Low	0.65	0.11	0.07	0.04
No	High	1.18	0.28	0.16	0.09
No	Low	0.46	0.08	0.05	0.03

Table 6.6: Test results enabling both bidirectional coding and motion compensation. Video clip is *News*, containing little movement. Layer 1-3 show differential data only.

"Container" - dummy algorithm					
Settings		**Temporal layers (bits/pixel)**			
Color	Quality	Layer 0	Layer 1	Layer 2	Layer 3
Yes	High	1.84	0.60	0.30	0.14
Yes	Low	0.74	0.14	0.07	0.03
No	High	1.49	0.58	0.30	0.14
No	Low	0.55	0.13	0.07	0.03
"Container" - extended prototype					
Color	Quality	Layer 0	Layer 1	Layer 2	Layer 3
Yes	High	1.84	0.34	0.29	0.14
Yes	Low	0.74	0.07	0.07	0.03
No	High	1.49	0.32	0.28	0.14
No	Low	0.56	0.06	0.06	0.03

Table 6.7: Test results enabling both bidirectional coding and motion compensation. Video clip is *Container*, containing little movement. Layer 1-3 show differential data only.

6.4 Summary

In this chapter the extended prototype has been tested with the original VSCBN prototype as a baseline for comparison. By using a dummy function in the motion estimation algorithm in combination with pure forward prediction the extended prototype produce the same amount of differential data.

The frame sequence illustrated in figure 6.3 gave the highest compression ratio of the tested sequences. For motion compensation, full search gave the best results, as expected. Both matching criteria gave equally good results. The combination of motion estimation and bidirectional coding enhance the compression rate with between 16% and 70%.

Chapter 7

Conclusion and Further Work

The goal of the work described in this thesis has been to increase the compression rate of the VSCBN prototype made in the DMJ project by adding two new compression techniques. The new techniques are called motion compensation and bidirectional coding.

The added compression techniques reduce the differentially coded data with up to 70%. As expected, video with little or no movement however have very little to gain

7.1 Reviewing Project Goal

The VSCBN prototype provides scalability at the sender side, receiving side as well as across the network. The video signal in the prototype is divided into several video quality dimensions, so that a viewer may choose appropriate quality in each dimension. To achieve this a specific layering is used. Existing compression implementations were considered, but none provided a satisfying quality layering, and so a new compression algorithm was created.

Motion compensation and bidirectional coding have been added to the existing compression algorithm as part of this thesis. The reason for adding the two techniques, and the goal of this thesis, is to achieve better compression rates than the original VSCBN prototype.

Motion compensation and bidirectional coding is used at a large scale for video compression, and is considered basic compression techniques. Implementing the two techniques should therefore yield a significant increase in compression rate without lowering the perceived quality.

7.2 Results

The author has currently achieved an improved compression of the differential data by 16% to 70% depending on video characteristics and subscription settings. The increased compression rate comes at the cost of higher resource usage, and a delay in the encoding/decoding process.

7.3 Discussion

The VSCBN prototype has the potential to solve a big problem with todays video content providers on the Internet. Servers providing live coverage on the Internet, for example NASA TV, warn their users that a *Server has reached capacity* error message might occur if there are too many viewers. The scalability of the prototype has the potential to reduce the workload of the servers, and also reduce the overall usage of bandwidth. The downside is that there would have to be a permanent content based network built on top of the Internet.

By adding bidirectional coding, a delay is created in the encoding and decoding process. This delay is tied up to the video stream, and cannot be compensated with increased processing power. It also uses more memory due to extra buffering. This extra resource used must be compared to the compression gain.

In the original implementation the predicted frames with the longest references compress poorly. For some videos, coding these frames from scratch would actually increase compression rate. This show that there was a great potential for further compression in the original prototype.

Compression rates were expected to be significantly lowered for videos with a large amount of movements. The author considers this goal to have been met. A compression of up to 70% of the differential data is considered a good result.

7.4 Criticism

In addition to adding bidirectional coding and motion compensation, a method to store pre-compressed video could have been implemented. Without pre-compressed video, the frame rate using the most demanding compression settings is very low.

Block-based motion compensation was chosen because of its low complexity compared to the more advanced algorithms. Compression efficiency is however lower then the alternative methods. Bidirectional coding is also implemented in it's simplest form. More advanced solutions may have given better compression efficiency.

The updated prototype copies all pixel data from the original frame-buffers into superblock sized buffers. The new buffers are used for motion compensation. This is an unnecessary operation since the pixel data could be extracted directly from the original frame-buffers. From a programming point of view this is a simplifying factor, but not a very efficient one.

The original code base was quite large, and gaining an understanding of how the code worked was not the simplest task. With the enormous amount of data passing through the encoding process the task of finding out exactly what happened was very difficult. The introduction of synthetic video gave an input that was much easier to debug. Motion could be controlled and noise eliminated entirely. The move to synthetic video should have been made earlier.

7.5 Further Work

In parallel with this work, Frank Jensen has looked into graceful degradation when resource availability is reduced. The use of a database to store video is another idea that is being considered in the DMJ project. This section also suggest some short term improvements.

7.5.1 Short Term Improvements

The current implementation is a small start compared to what can be done. The code probably has a small inaccuracy in the motion estimating algorithm, and there are several possible optimizations that the prototype would benefit from.

Known Bugs

The motion calculation algorithm seem to make mistakes. When compressing synthetic video, the motion estimation code will sometimes report that only perfect matches were found, but still there is a difference between the match found, and the block to compress.

Another problem is that the first few frames the user receives are not displayed correct. If not repairable it would perhaps be better not to display incomplete frames at all. There is also a problem with chrominance data not being reset, or zeroed out, when cancelling a subscription to color information.

Optimizations

The optimizations suggested in section 4.3.3 may increase processing speed or the accuracy of the matches in motion compensation.

Memory usage can be reduced by calculating motion vectors directly in the frame-buffer received from the video source instead of copying the pixel data.

It is possible to reduce the average number of bits needed to represent the average vector based on the fact that some vectors are more common then others. Huffman coding and arithmetic coding are both suitable candidates. Direction signals are coded with one bit per signal. Normally direction signals in some areas of a frame have a tendency to point in the same direction. This can be used to compress the signals by using run-length or similar algorithms.

The threshold value used for aborting further searching in motion compensation should describe the average difference for a single pixel. In this way the same threshold value can be used for various matching criteria.

Calculating motion vectors for future and past references could be done in parallel. A match is likely to be close to the start position, so by calculating both simultaneously a match within the given threshold can be found faster.

7.5.2 Possible New Features

The following improvements are features that are being implemented, or might be considered implemented at a later stage.

A More Dynamic GOP

Dynamic GOP is a possible method to achieve further compression. Frame number four will sometimes be less efficiently compressed then a regular I-frame if there is no redundancy between it and frame zero. Dynamically deciding to code frame four as an I-frame in such cases should increase compression ratio.

Adaption

Subscriptions are the mechanism a receiver use to specify a quality based on resource availability and user preferences. Since resource availability vary over time it will become inconvenient for the receiver to respond to these variations manually. This should instead be done automatically by an adopting mechanism. If there are extra resources, these can be used

to enhance utility. If there is a lack of resources, automatic adaption may contribute to a graceful degradation of the video quality.

Database

A database containing a compressed video stream, may reflect the layered structure of the stream. Where a file stores data sequentially, a database may access the different layers of the video stream independently. This may be useful if the server only sends a subset of all available granularity levels. Meta-data like scene description can also more easily be added to the video.

Bibliography

[1] T. André, B. Pesquet-Popescu, M. Gastaud, M. Antonini, and M. Barlaud. Motion estimation using chrominance for wavelet-based video coding. In *Proc. IEEE Picture Coding Symposium*, San Francisco, USA, 2004.

[2] Randa Atta and Mohammed Ghanbari. An efficient layered video codec based on dct pyramid. Technical report, 2002.

[3] Broadcastpapers.com. Visited 8. July, 2004 `http://www.broadcastpapers.com/sigdis/TektronixMPEGGuide\%20-\%20comp08%.htm`.

[4] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Achieving expressiveness and scalability in an internet-scale event notification service. *Nineteenth ACM Symposium on Principles of Distributed Computing (PODC2000), Portland OR. July, 2000*.

[5] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems - consept and design*. Pearson Education, third edition, 2001.

[6] R. A. DeVore and B. J. Lucier. Wavelets. In *Acta Numerica 1*, pages 1–56. Cambridge University Press, 1991.

[7] Greg Goebel / In The Public Domain. Visited March 2004 `http://www.vectorsite.net/ttdcmp1.html`.

[8] Viktor S. Wold Eide, Frank Eliassen, Olav Lysne, and Ole-Christoffer Granmo. Extending content-based publish/subscribe systems with multicast support. Technical Report 2003-03, Simula Research Laboratory, July 2003.

[9] Viktor S. Wold Eide, Frank Eliassen, and Jørgen Andreas Michaelsen. Exploiting Content-Based Networking for Fine Granularity Multi-Receiver Video Streaming. In Surendar Chandra and Nalini Venkatasubramanian, editors, *Proceedings of the Twelfth Annual Multimedia Computing and Networking (MMCN '05),*

San Jose, California, USA, volume 5680, pages 155–166, January 2005.

[10] Didier Le Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM, 34(4), pp. 46-58*, 1991.

[11] MPEG industry forum. MPEG-4 simple / advanced simple visual profiles brochure. Visited 8. July, 2004, `http://www.mpegif.org/public/index.php?sub=m4if`.

[12] Scott R. Ladd. The dirac video codec: A programmer‘s guide, May 2004.

[13] Sang Uk Lee and Jong Won Kim. Variable block size techniques for motion sequence coding.

[14] Weiping Li. Overwiev of fine granularity scalability in MPEG-4 video standard. *IEEE Transactions on circits and systems for video technology, vol. 11, no 3*, march 2001.

[15] Colin E. Manning. New Media Republic, Visited April 2004 `http://www.newmediarepublic.com/dvideo/compression/`.

[16] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. In *ACM SIGCOMM*, volume 26,4, pages 117–130, New York, August 1996. ACM Press.

[17] University of Missouri-Columbia. Index of free download videos. Visited 11. May, 2005, `http://meru.rnet.missouri.edu/free_download/videos/`.

[18] L. Po and W. Ma. A novel four-step search algorithm for fast block motion estimation, Jun 1996.

[19] Peter Schroder. Wavelet image compression: Beating the bandwidth bottleneck. *Wired Vol. 3, No. 5*, may 1995.

[20] Gidon Shavit. Binary tree motion compensation and alternative image transforms for gtv. Qualifying project at the University of Washington.

[21] Jerzy Stachera and Slawomir Nikiel. Fractal image compression for efficient texture mapping. *WSCG POSTER proceedings*, 2004.

[22] Chong Sze Tong and Minghong Pi. Fast fractal image encoding based on adaptive search. *IEEE Transactions on image processing, vol. 10, no 9*, Septmber 2001.

[23] Akio Yoneyama, Hiromasa Yanagihara, and Yasuyuki Nakajima. One-pass vbr mpeg encoder using scene adaptive dynamic gop structure, June 2001.

[24] Chroma resampling (video and image processing blockset). Visited 23. September, 2004, `http://www.mathworks.com/access/helpdesk/help/toolbox/vipblks/chromares%ampling.html`.

[25] Distributed media journaling. `http://www.ifi.uio.no/~dmj/`.

[26] Gw hannaway and assosiates - helpful info - video basics. `http://www.gwha.com/videobasics.html`.

[27] Jpeg 2000. Visited 23. September, 2004, `http://www.jpeg.org/jpeg2000/`.

[28] Wikipedia, the free encyclopedia - Motion compensation, visited august 2005. `http://en.wikipedia.org/wiki/Motion_compensation`.

[29] Wikipedia, the free encyclopedia - Multicast, visited mars 2005. `http://en.wikipedia.org/wiki/Multicast`.

[30] Wikipedia, the free encyclopedia - Retina, visited july 2005. `http://en.wikipedia.org/wiki/Retina`.

Appendix

CD-rom

A CD-rom accompanying this thesis include the source code for the extended VSCBN prototype. The following classes were added to the prototype:

- MotionEstimation.java
- TransmissionQueue.java
- WaitQueue.java

In addition the following classes were modified:

- Const.java
- MacroBlock.java
- SuperBlock.java
- VideoClient.java
- VideoCoder.java
- VideoServer.java

SIENA is required to run the prototype. The *SIENA* version updated in the DMJ project has been used in the testing phase, so this version is also included on the CD. A copy of this document in the *pdf* format is also included