The Design and Implementation of a Broadcast Quality Real-Time Aspect Ratio Converter

Set Norman

March 19, 2009
Master's Thesis in Computing Science, 30 hp credits
Supervisor at CS-UmU: Thomas Pederson
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

Aspect ratio conversion of broadcast quality interlaced video has traditionally been performed in a real-time fashion using specialized hardware coupled between expensive video servers or occasionally live at playout. As media management is moving towards digital file-based format and computers are getting faster the interest for software aspect ratio converters that do not sacrifice quality has increased. A software implementation has the potential of being both cheaper and faster than the hardware equivalent.

A background covering broadcasting technology as well as the specific problems of aspect ratio conversion and deinterlacing is presented. It is concluded that it is possible to implement a software aspect ratio converter with real-time performance and quality on par with professional ARC hardware using the combination of a modified spatio-temporal deinterlacing technique coupled with finite impulse response low-pass filtering and spline based rescaling.

Contents

Chapter 1

Introduction

Aspect ratio conversion is the task of rescaling an image of some original proportions to proportions appropriate for a specific display device or transmission channel. In its most basic case the problem is relatively simple, well covered in literature but nevertheless not completely solved. The problem presented in this Master's Thesis is however that of aspect ratio converting interlaced video sequences which makes the problem significantly more difficult. A brief introduction of broadcasting technology will be given in the following sections to explain interlacing as well as other topics. As for why interlacing makes the aspect ratio conversion harder the reader may proceed to the problem description chapter.

1.1 Fundamentals of moving images

We will start by loosely defining a few terms used throughout this thesis.

Video will refer to any sequence of images and will include any kind of recording, transmission and display technique of such images. *Film* is a subset of video where images are recorded primarily on celluloid film and primarily designed to be rendered by a projector onto a movie screen. The most notable difference being that film is never interlaced (which will be described shortly) as an original while video may or may not be interlaced.

To denote a single image in a sequence of images we will use the term *frame*. Virtually all video and film formats require the display device to render a fixed number of frames per second with a constant interval. The number of frames displayed each second is called the *frame rate*. The universally most common film format has a frame rate of 24 FPS (frames per second) while video ranges from 25-30 FPS for most applications.

While moving images are more and more evolving into a digital medium many legacies remain that has an impact on how video and film are handled and stored today. Thus it is relevant to continue with an in-depth description of this technology.

1.1.1 Recording, transmitting and displaying video

Interlaced video

The most distinguishing feature of video versus film is that video is most commonly recorded, transmitted and displayed using a technique called *interlace*. Assume for

example that a video sequence is recorded at 25 FPS and recording is started at $t = 0$ measured in seconds. At $t = 0$ the camera will record every second horizontal line, which we will denote as the set of odd lines. At time $t = \frac{1}{50}$ the camera will record the remaining lines, the set of even lines. At time $t = \frac{2}{50}$ the camera will again record the set of odd lines and will continue in that way to create a sequence of alternating sets of odd and even lines.

One set of either odd or even lines is called a *field*. We also need to extend the definition of frame to include that two sequential fields forms a frame. Thus we can define *field rate* as twice the frame rate. It is common to refer to fields in a frame in many different ways including odd/even, top/bottom and first/second. We assume that if the top field is the first field then this applies to every frame of the video.

The main reason for recording video field by field remains to be explained but first we need to understand how the most common display device, a cathode ray tube (CRT) works. A cathode ray tube works by beaming a ray of electrons onto a phosphorus covered glass window. The ray will make the phosphorus glow for a brief period of time but only a single spot of phosphorus can be lit at a time. However, by moving the beam quickly across the screen while varying the intensity of the beam a pattern can be rendered. Virtually all CRT:s of today move the beam in straight lines from left to right starting with the top line and moving down line by line. Now assume that we would render the video in a non-interlaced manner (progressively) at 25 FPS, then any point on the screen will be lit once every $\frac{1}{25}$ of a second and then slowly fade. This will give a flickering appearance since the human mind is capable of perceiving intensity variations well above 25 Hz [11].

However, if the image is rendered field by field, i.e. any small region of the screen will be rendered once every $\frac{1}{50}$ of a second which will reduce the most severe flickering. This refresh rate of 50 Hz was chosen as an acceptable compromise between quality and bandwidth requirements [11].

Note that interlaced video can be displayed effortlessly on a cathode ray tube but other display devices that do not employ an electron beam for the rendering of images will have quite a bit of trouble. This is because any other display device will most likely need to display two fields that are $\frac{1}{50}$s apart at the *same* time to form a complete frame, i.e. such devices normally light every single pixel at the same time. If sequential fields are simply stitched together then artifacts will be apparent where motion has occurred.

Progressive video

The alternative to interlaced video is called progressive video. If a video camera records both fields at the same time progressive video will be recorded. Note that progressive video may in a standard definition broadcasting context nevertheless, and most likely will, be transmitted and rendered as if it was interlaced.

Video broadcasting standards

There are a great many video broadcasting standards for analog consumer television but it is interesting to look at differences in the properties of the main alternatives. These are called NTSC[1] and PAL[2]. NTSC is used predominately in the US while PAL is used in most parts of Europe. They have different signal encoding techniques which give them

[1] NTSC - National Television System Committee
[2] PAL - Phase Alternating Line

different advantages and drawbacks but in the context of this thesis the interesting properties are the frame rate and number of horizontal lines which can be seen in the following table.

Standard	Lines	Active lines	Frame rate
PAL	625	576	25
NTSC	525	480	29.97

As can be seen NTSC has sacrificed vertical resolution to gain a higher refresh rate. The pixel count per second is equal between the two formats, $25*576 = 30*480 = 14400$.

1.1.2 Film fundamentals

Traditionally film has always been captured on a moving film reel making it quite difficult to do anything but full frame exposures. Nor does a film projector benefit from interlacing as any part of an image is displayed at constant intensity for the duration of the frame. Thus film is always progressive. As was noted earlier almost all classical film formats are recorded at 24 frames per second.

1.1.3 Telecine

The process of converting film to a video format suitable for display on a television set is called telecine. The main problem is the difference in frame rate between the formats. The fact that we are moving from a progressive to an interlaced format is however more of an advantage since it allows for some clever rearrangements to be done to achieve the conversion.

The most interesting telecine conversion is that of converting 24 FPS film to 29.97 FPS NTSC. We may assume that NTSC is 30 FPS since the small resulting error can be ignored. The idea is to rearrange the available material so that the number of frames are increased by $\frac{5}{4}$ or in other words, add one frame for each group of four. Assume that we have four film frames A, B, C and D. Since our output format is interlaced we may split the original film frames into paired fields A1-A2, B1-B2, C1-C2 and D1-D2. Increasing these 8 fields to 10 fields require us to double two fields. A common method is to add a new frame formed by the pairing of B1-C2 into the middle of the sequence as illustrated in figure 1.1 on the following page. While this might not provide the smoothest of motion it is a simple and robust technique.

In the case of film to PAL telecine the conversion is from 24 FPS to 25 FPS. As there is no acceptable rearrangement of fields that will achieve this change smoothly enough another approach needs to be used. The standard solution is to simply play the film slightly faster, this will also slightly increase the pitch of the audio but that can be corrected. This will also decrease the length of the movie by four percent.

As a side note we can consider temporally interpolating across frames as a means of resampling the material. This is a very interesting and difficult problem and can most likely be shown to be an even harder problem than aspect ratio conversion which will be covered in the following sections.

1.1.4 Aspect ratio

The term *aspect ratio* refers to the relationship between the width and height of an image. More formally the aspect ratio of an image is the resulting quotient when dividing the

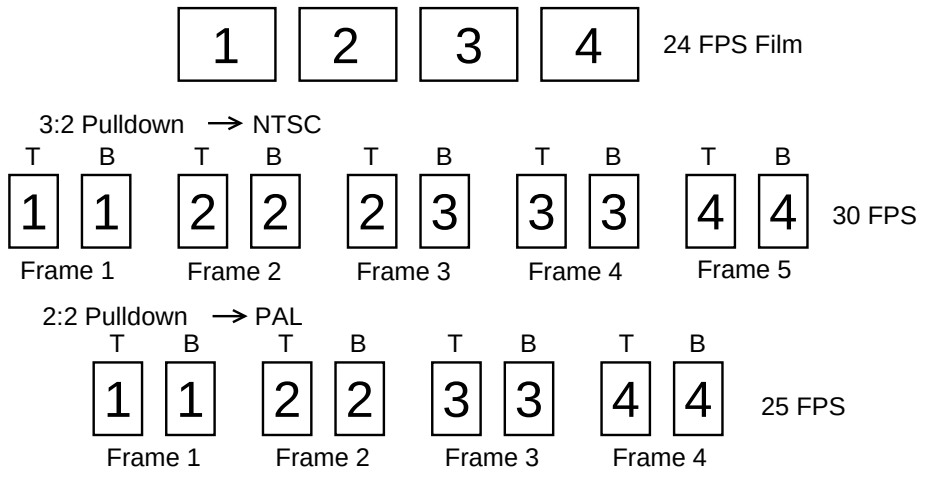

Figure 1.1: Telecine

image's width by its height. A perfectly square image thus has an aspect ratio of 1. We may also write the aspect ratio as the simplified result from the following equation $width/D : height/D$ where D can equal the height or the greatest common divisor of the width and height depending on convention. A square image would for example be written as 1:1.

A few aspect ratios are especially common for television sets. 4:3 (1.33) is the aspect ratio used in almost all television sets before widescreen sets started to become available. A widescreen television set has an aspect ratio of 16:9 (1.77). Film commonly has aspect ratios as wide as 2.35:1 (CinemaScope).

1.1.5 Digital video

When digitizing a video sequence it is interesting to look at a few of the properties which will affect how the digitization is done. First let us assume we are digitizing PAL video. PAL transmits 625 lines at 25 FPS. However, not all lines are *active*, that is, in the visible region of the screen. If we limit ourselves to active lines only 576 remain. The remaining question is which sample rate to use when sampling the signal. This will directly affect the width in pixels of the digitized image. One choice would be to sample at 14.75 MHz so that an aspect ratio of 4:3 is yielded (768x576). This would be useful since the sampled images would have square pixels which is common on LCD monitors. It was however decided in the ITU-R BT.601 standard [16] that 720 pixels was sufficient which is the result of a 13.5MHz sample rate.

If a larger region of the PAL transmission needs to be stored an extended resolution of 720x608 is commonly used. For NTSC the equivalent resolutions are 720x486 (standard) and 720x512 (extended). The extra 32 lines in the extended resolution format allows for storing the most interesting information found in the off screen area called the vertical blanking interval or VBI.

1.2 Aspect ratio conversion

The problem of aspect ratio conversion originates in the recent evolution of broadcast television standards. The widescreen aspect ratio (16:9) is growing more popular and the solutions used to take advantage of this format creates backward incompatibilities that needs to be handled appropriately. The problem is made more difficult by the fact that video media is stored in a format that is beneficiary to analog transmission and output requirements and thus difficult to modify, i.e. with interlace.

1.2.1 History

During the first half of the 20th century, most movies were recorded on a 35 mm film format. This film format has an aspect ratio of 1.37:1. As television started to become available it was conveniently decided that television should use an aspect ratio of 4:3 (1.33:1), i.e. almost the same as the contemporary movie format.

The availability of television resulted in a decrease in movie theater visitors and Hollywood created the widescreen format so that movie theaters could again provide something unique. From the 1970's and forward the predominant film format was the 2.39:1 which is still common today. The widescreen movie format nevertheless uses the original 35 mm film format (1:37:1). This is achieved by squeezing the recorded film in the vertical direction using what is called an anamorphic lens. When the film is played back another anamorphic lens performs the inverse conversion back to the correct aspect ratio.

The next generation television-standard HDTV uses a 16:9 (1.78:1) aspect ratio. The aspect ratio 16:9 is now commonly referred to as widescreen television and is not only a HDTV format but is also used in standard television sets (which previously only received 4:3 images).

1.2.2 Anamorphically stored video

The television broadcasting standards PAL and NTSC described previously were designed strictly for use with a 4:3 format and has no special means of handling any wider format. However, if the cathode ray tube (CRT) would simply be widened and the ray would be made to move faster across the screen in the horizontal direction the picture rendered could be made to look wider than intended. If the input signal would be tweaked to accommodate this change at the output (to something that regular TV set would display as a horizontally squashed image) a widescreen television-set would render it correctly. So with a few simple changes the original format can be used as a widescreen format. Video stored in this non-standard manner is called anamorphic video.

Commonly 16:9 video is stored anamorphically but as noted if such video would be rendered directly on a 4:3 display it would look horizontally squashed. This makes it difficult for broadcasting companies to use anamorphic video since a large group of viewers would be unable to view the broadcast at the correct aspect ratio. This is often solved by broadcasting all video at 4:3 but with black horizontal borders thus enabling widescreen displays to perform an aspect ratio conversion where the black borders are removed. Such technology is included in virtually all 16:9 television-sets. This implies that the vertical resolution is reduced by one quarter compared to a true anamorphic transmission.

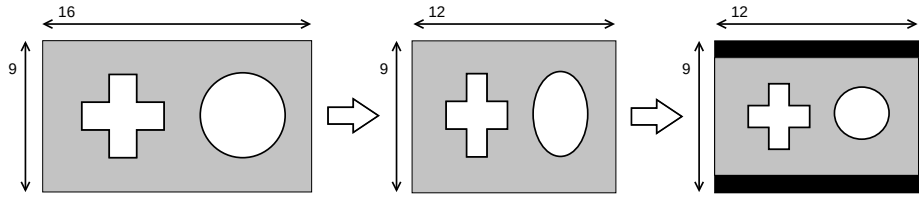

Figure 1.2: A 16:9 image store using anamorphic coding in a 4:3 format and then properly displayed on a 4:3 screen.

The aspect ratio conversion needed to perform 16:9 to 4:3 conversion (anamorphic pulldown) is vertical scaling by 3/4. As the rescaled image is only three quarters as high as the original the bottom and top parts of the image are painted in black. The technique of rendering the top and bottom part in black is called letterboxing, see figure 1.3. Some conversions downscale the image in the horizontal direction, such an image is called pillarboxed because of the vertical black pillars or bars.

For example an anamorphic downconversion implies that if our 16:9 image is stored anamorphically in an image of resolution 720x576 pixels (and is to be displayed at that resolution) the image needs to be rescaled to fit within 720x432 pixels if it is to be displayed with the correct aspect ratio at 4:3. The remaining $142 = 576 - 432$ lines will be drawn as black.

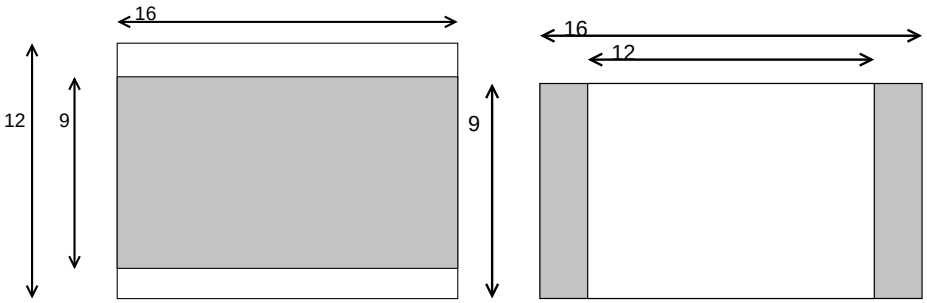

Figure 1.3: To the left, a letter boxed 16:9 image. To the right, a pillar boxed 4:3 (equivalent to 12:9) image on a 16:9 display

1.3 Applications of aspect ratio conversion

1.3.1 Digital broadcasts

A very interesting approach to aspect ratio conversion is taken in the Digital Video Broadcasting (DVB) standard of digital television. Pictures transmitted may be in a wide range of formats and are aspect ratio converted by a set-top box locally at each receiver. The big advantage of this technique is that the set-top box can be told the aspect ratio of the display device used. This makes it possible to display true anamorphic 16:9 video on a widescreen TV without unnecessary conversions. If the viewer only has a 4:3 display the video can be “letter boxed” so that is is appropriately rendered. The drawback of this approach is that the quality of the aspect ratio conversion algorithms

in the set-top boxes is most likely not nearly as good as if the conversion had been performed prior to broadcast by professional hardware.

If the set-top box does not have an aspect ratio converter the image can be sent unconverted with additional burnt in video data called wide screen signaling (WSS) in line 23 of the frame. This will communicate which conversion needs to be applied to get the correct aspect ratio. Most widescreen TV sets are capable of reading the WSS and performing the necessary conversion.

1.3.2 Analog broadcasts

In an analog broadcast the aspect ratio conversion needs to be performed before broadcast since virtually no 4:3 TV sets can perform the conversions necessary to correct the aspect ratio. However, as most TV-sets being sold today have a widescreen format and HDTV is strictly widescreen it makes a lot of sense to use 16:9 as the main recording format. In any case this is true for channels which are distributed digitally as described previously. While a 16:9 recording needs to be carefully made to suit analog broadcast, most analog channels are also distributed digitally which increases the advantage of doing all new recordings in a 16:9 format.

The conversion done for the analog broadcast is mostly done using special hardware aspect ratio conversion boxes. They convert video signals in real-time and can also be controlled to switch conversion on-line during playout. Conversion may also be performed during editing on what is called non-linear editors or NLE:s. If all video is stored as digital video files it may also be converted by a special software aspect ratio converter.

1.3.3 Cathode ray tube conversion

As has been mentioned above most widescreen television sets can perform aspect ratio conversion. The technique employed in such displays is however different from the normal type of aspect ratio conversion performed by special hardware or set-top boxes. A CRT will perform the conversion by directly altering the path of the cathode ray depending on the desired video aspect. If the input is for example letter boxed the CRT may simply drop the transmission of the black bars and run the beam in a vertically wider pattern across the screen when actual video is transmitted to achieve a full screen 16:9 view.

1.3.4 Future of aspect ratio conversion

As it is possible that most future recordings will use a 16:9 aspect ratio the need for conversion could be eliminated when the analog 4:3 transmissions are closed down. Nevertheless, there will still be huge amount of archived material in 4:3 format that will need to be converted to 16:9 for use in for example HDTV. The problems of up and down conversion needed to convert between HDTV and SD formats are very similar to the problems of aspect ratio conversion and knowledge acquired in either field is applicable in the other.

1.4 Aspect ratio conversion in the context of media management systems

The main application for the aspect ratio converter that has been implemented as part of this Master's Thesis is to be integrated in the ARDOME Media Management System. While the implementation may be used standalone a short introduction to media management system is given to introduce the context of the implementation.

A media management system is exactly what is sounds like, a system to organize and handle media in a way that allows for some work flow to take place in a sensible way. In the context of this thesis the media handled are digital video files and we will therefore be discussing file based media management systems.

The fundamental idea of a file based media management system is to replace a physical tape based work flow with a digital file based work flow. This would apply to any television company that records, edits and transmits video. For example, a tape based work flow might include going down to an archive, physically retrieving a video tape and then performing editing and playout of the video directly from tape. A media management system strives to simplify that procedure by digitally storing files on hard drives or in tape archives. This allows for example the archive retrieval operation to be performed by a simple network file transfer instead of the physical retrieval of a video tape.

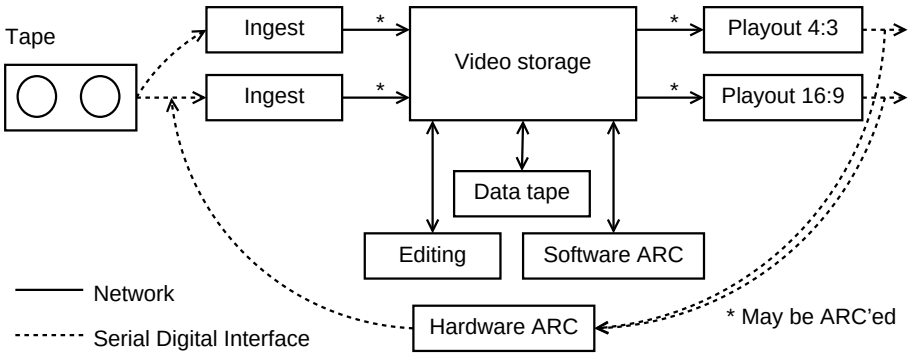

Figure 1.4: Basic layout of media transport in a digital file-based media management system

There are many advantages of digital media asset management systems including the way in which a software aspect ratio converter can be incorporated within the system. In figure 1.4 a hardware ARC based work flow for an aspect ratio conversion operation is depicted. Video is played out from a video server over SDI[3] through a hardware ARC which will incur a minimum delay of one field. At the same time the output video is recorded by a second video server. Please note that while a hardware ARC is fairly cheap a video server is usually extremely expensive. Aspect ratio conversion may also be used at the time of recording and playout with a much smaller cost since at that time the hardware ARC is the only extra component needed. However, at the time of recording we may not know what format we will need the file to be in the future and would thus be unable to make a choice. Aspect ratio conversion at playout also

[3]Serial Digital Interface, a coaxial based digital video cable.

has potential problems, switching aspect ratio conversion mode on-line must be done at exactly the right frame and without generating suboptimal frames during the switching transition.

The software aspect ratio conversion operation consists of acquiring one or more free CPU nodes with sufficiently fast access to the media to be processed. The file is processed on the node(s) at approximately real-time speed depending on available CPU resources and then written back to hard disk or a network drive.

1.5 A short review of broadcast quality digital file formats

To understand some of the problems dealing with compressed digital video it is important to understand the compression technique used. It is also important to have a clear understanding of the layout of a frame stored in a video file as not all lines contain active video and aspect ratio conversion tends to move lines around.

1.5.1 Compression techniques used in digital video

Intraframe coding

Intraframe coding is coding within one frame. A very common technique is to subdivide the image in blocks and apply the discrete cosine transform or DCT across each block. When decoded the difference in encoding across neighbouring blocks may become visible and a blocked pattern appear. This pattern gives sharp edges in the video material and such edges may become enhanced if improperly handled by an aspect ratio conversion algorithm.

Interframe coding

A universal coding technique used for digital video is that of interframe coding. By interframe we mean across frames in a sequence. It is based on the idea that once we have established what a complete frame looks like at one point in time we may describe the next frame by reusing (square) picture elements or blocks from the previous frame to create the next frame. In other words the next frame is described by a set of block motion vectors. In the case of for example MPEG this can be even further elaborated if motion vectors are allowed to refer to information in future frames. In strongly compressed MPEG the effect of this compression technique can become clearly visible with artifacts such as streaking and moving block artifacts. The block artifacts from interframe coding need not be aligned to a specific grid as is the case with intraframe coding.

Chromatic undersampling

Video formats commonly store data in a format called YUV. Y represents luminance information, i.e. the value or brightness of color while UV represents chrominance, a 2D plane which decides the actual color. Thus three separate values needs to be stored for each and every pixel of the image. However, it is more difficult for the human eye to notice intensity variations in the chrominance values than the luminance values and it is therefore possible to decrease the resolution of UV data while maintaining an apparently high resolution image[11]. This technique is almost always used in digital video files even

in the most high quality formats. Without going into detail the formats 4:1:1, 4:2:0, 4:2:2, 4:4:4 all describe slightly non-intuitive ratios of luminance and chrominance values.

1.5.2 Digital video formats

File formats

While there are many file formats used to store video two of them are by far the most common; DV and MPEG. Both of them are common in both professional and consumer electronics. DV is common in consumer digital video cameras and MPEG is used in for example the DVD format. The professional versions of DV are called DVCPro25 and DVCPro50. The MPEG-2 standard is versatile enough to encompass an extremely wide range of applications including consumer and professional technology. The versatility of MPEG-2 comes at the price of increased complexity which may impact the interoperability of the format.

The DV format is never interframe encoded which is a requirement for allowing reasonable editing and reverse playback on magnetic tapes. It is also possible to restrict an MPEG encoder to never use interframe encoding which is the case in the MPEG-2 D10 standard which is used by Sony IMX digital video tapes, among others.

Frame layouts

MPEG and DV formats feature different frame layouts which makes software aspect ratio conversion slightly more complicated. In table 1.1 the 'active' column illustrates if a specific line has no active video, half a line of active video or a full line of active video. The half lines at the top and bottom are not visible on a normal TV as they fall just outside the visible area but may become visible after aspect ratio conversion.

Table 1.1: Layout of a PAL frame

Line	Field	Active	MPEG	DV
7	1		extended resolution first line f1	
320 (7)	2		extended resolution first line f2	
⋮	⋮			
2235 (22)	2			first line f2
23	1	———	first line f1	first line f1
336 (23)	2	———————	first line f2	
⋮	⋮	———————		
622 (309)	2	———————		last line f2
310	1	———————	last line f1	last line f1
623 (310)	2	———	last line f2	

1.6 Related parties

This Master's Thesis is done at Ardendo AB[2], a software company focusing on complete Media Asset Management solutions for the Broadcast Industry. Ardendo AB has provided office space, salary and an very expensive broadcasting monitor necessary to

facilitate the development. BBC Broadcasting has been helpful in providing a hardware ARC for comparison testing as well as sharing experiences on hardware ARC:ing. Vlaamse Radio- en Televisieomroep (VRT) has provided test material which contained material which was very difficult to convert correctly and that was absolutely essential for testing.

The software resulting from the work done in this Master's Thesis is currently in production at Red Bee Media, Vlaamse Radio- en Televisieomroep and FOX Sports. In some cases the software is also used to perform HD to SD conversions.

1.7 Thesis overview

The problem as well as motivation for the problem that this Master's Thesis attempts to solve is described in chapter 2. Chapter 3 introduces the concept of deinterlacing in more detail including several different techniques to solve the problem. In chapter 4 the process of understanding and implementing the aspect ratio converter will be presented. Chapter 5 offers the results and chapter 6 the conclusions.

Chapter 2

Problem Description

Going back to the title of this Master's Thesis "The Design and Implementation of a Broadcast Quality Real-Time Aspect Ratio Converter", which states that the goal has been to create an aspect ratio converter that is sufficiently fast (i.e. real-time) and have a quality on par with working technology already being employed in broadcasting environments. By real-time performance we are referring to the capability of being able to process one second of material in one second or less.

The problem of performing such a conversion can arguably be divided into two separate steps, a deinterlacing operation and a rescaling operation. The distinction between the two is always clear in the approach taken in this Master's Thesis even though it is possible that some yet to be conceived algorithm may blur that line. Since we thus have had to solve two distinct problems two different fields has been covered. Most of the focus has however been on deinterlacing as far more trade-offs are required to be made when solving that problem. Optimal rescaling on the other hand is comparably easy to implement and the best solutions may be done in near real-time which is not the case for deinterlacing.

2.1 Problem statement

The central problem of developing a real-time aspect ratio converter has thus been in finding a high quality, real-time deinterlacing algorithm. The most important quality factors are sharpness (maintaining high vertical resolution), minimal artifacts and maintaining good quality even on worst case inputs. The field study in chapter 3 clearly points us in the direction of non motion-compensated spatio-temporal deinterlacing techniques and this area has therefore been evaluated so that a specific method or hybrid method could be implemented.

The secondary problem of actually scaling the image was achieved by finding and imlementing a well established high quality algorithm to do so, namely the Catmull-Rom scaler. An additional problem of scaling is the risk of aliasing and to avoid this we have evaluated the need for applying a low-pass filter when performing downconversions.

2.1.1 Fundamental problems with scaling

Assuming we have a progressive[1] video frame the problem of aspect ratio conversion consist of scaling an image along the vertical and/or horizontal axes by some scaling factor in the range of 2:1 - 1:2. This can be done in quite a few ways including for example removing/doubling lines, linear, cubic and spline based interpolation. In the case of interlaced video we have apply our spatio-temporal deinterlacing technique to yield a progressive image suitable for input into the rescaling algorithm.

A problem when rescaling video comes from the fact that an image recorded by a video camera is not necessarily bandlimited below the Nyquist-frequency. The rescaling processes also requires low-pass filtering to avoid aliasing in the resampled image. Ideal low-pass filtering in the context of avoiding aliasing is an opposite requirement of avoiding ringing artifacts along sharp edges and thus a trade-off needs to be made. Designing and implementing such filters is non-trivial and as this is beyond the scope of this Master's Thesis the subject will only be briefly discussed.

2.1.2 Quality considerations

As the quality of an aspect ratio conversion algorithm is of utmost importance it is essential to determine how to measure quality. The best way to measure quality would be to compare the output of the algorithm with a known correct result and measure the difference as a signal to noise ratio. It is however not easy nor necessarily possible to create a correct in and output video to which comparisons can be made. An alternative is to use a group of experts to compare two different aspect ratio converters and then ask them to determine for example which converter gives the least annoying artifacts.

The fact that it is extremely difficult to acquire or generate video representing all reasonably different kinds of input is also a problem related to quality. Thus it is very important that the algorithm is robust and well thought out to be reasonably proficient in handling any kind of input.

2.1.3 Runtime considerations

The performance of a software aspect ratio converter can potentially supersede that of a hardware equivalent as a hardware ARC will always run at real-time speed. However, super real-time performance is rarely a requirement.

General purpose PC:s

The cost of hardware can also be considered a factor. It would strike a nice balance between software and hardware encoders if the cost of a real-time software encoder system performing at real-time speed would equal that of a hardware aspect ratio converter. At the time of writing the cost of a cheap hardware aspect ratio converter is close to the cost of a fast dual core machine. This would imply that a price competitive software algorithm should at least run at half the speed of real-time on one CPU.

It might also be relevant to mention the benefit of purchasing general purpose PC hardware instead of special purpose aspect ratio conversion hardware since general purpose hardware can be used for other computing tasks while idle such as video transcoding or similar tasks.

[1] Progressive in this context means not interlaced, i.e. lines ordered plainly from top to bottom.

Parallel processing assisted computation

Some of the massive parallel processing benefits available for the hardware aspect ratio converters are also becoming possible in a general purpose PC. This can be achieved by using a modern graphical processing unit (GPU) to perform some of the calculations. Modern GPU's relay on massive parallel processing to be able to render and texture millions of polygons efficiently.

To be able to access the processing power of the GPU there needs to be a framework that allows an application to send and receieve data as well as flexibly specify the calculations that should be applied. OpenCL (Open Computing Language) is such a framework which is supported by all major 3D card vendors. Nvidia also provides the CUDA API which contains additional Nvidia specific features.

Finite impulse response filtering can be implemented as GPU operation [13] and texture scaling is performed natively on all texture rendering GPU:s so it seems likely that it is possible to implement a majority of aspect ratio conversion operations on the GPU. A few deinterlacing algorithms may also be possible to implement as pixel shaders. More investigations would be needed to determine if using a GPU would also increase the performance of the calculations.

2.2 Goals

The goals of this Master's Thesis are summarized below;

1. Implement a spatio-temporal deinterlacer that should have,
   (a) quality comparable to other real-world implementations.
   (b) real-time performance.
2. Find and implement a scaler solution which;
   (a) maintains image sharpness.
   (b) does not introduce aliasing or other artifacts.
   (c) runs in real-time.

A scaler is needed for a complete aspect ratio convertion workflow but it remains a secondary goal and will not be the main focus of this Master's Thesis.

2.3 Related work

As the most closely related work are all proprietary implementations, a quick review of such implementations will be given.

2.3.1 Hardware aspect ratio converters

A few of the major companies in the hardware aspect ratio conversion business are Snell & Wilcox, AXON and Vistek. The algorithms employed are all proprietary implementations. One might attempt to understand the constraints under which the algorithm operates and the problems such an algorithm faces by reading through the specification. A few items from a few different fact sheets are;

- Unique spatio-temporal filtering algorithm for transparent alias-free resizing.
- 12 taps vertical FIR-filter with 3 field processing
- Variable aspect ratio.
- Video and film modes.
- Control of picture size and position.
- Y black adjustment.
- Fixed 1 frame delay for easy installation.
- Total delay between input and converted outputs is one frame +0..-7 lines.

Some list items are related to issues such as signal levels, for example the black level which indicates the voltage below which the signal is considered completely black. In the case of a software implementation this would already have been taken care of by the recording device that generated the digitized video. As for details on the algorithm itself it is in one example referred to as a spatio-temporal filtering algorithm and in other cases as a vertical-temporal filtering algorithm. This would imply that no motion compensation is used as vertical/spatio-temporal filtering is a direct technique as will be described in the chapter on deinterlacing.

We can also see that the delay is commonly exactly one frame. This implies that the device has time to receive one and only one field beyond the field currently being converted. This field can be used to improve the quality of the conversion. Video and film modes refer to the fact that an ARC may take advantage of knowledge of the telecine process.

2.3.2 Software aspect ratio converters

To the author's knowledge there is no professional stand-alone software aspect ratio converter application. However, several software non-linear video editors can perform aspect ratio conversions such as Final Cut Pro and Avid Xpress. Note that these applications are capable of handling broadcasting quality video material which is not the case for most consumer oriented software programs.

Chapter 3

Theory of Deinterlacing

To fully appreciate what is meant by deinterlacing the reader must understand the concept of interlacing which is described in the following section. Deinterlacing will nevertheless be briefly outlined below.

Deinterlacing is the process where a sequence of interlaced images are converted to their non-interlaced or progressive form. The problem can be described as trying to maintain all of the vertical video resolution while avoiding artifacts. Video sequences will range from where deinterlacing is trivial to where the problem becomes so ill-posed that it is theoretically impossible to produce better than half the vertical resolution. Maintaining high vertical resolution therefore incur a high risk of creating artifacts as data needed to do so may easily be misinterpreted.

3.1 Interlacing

An interlaced sequence is a sequence of alternating odd and even fields. An odd field contains only odd lines and vice versa. The positioning of the odd and even lines is such that the odd and even lines are the odd respectively even numbered horizontal lines recorded by a recording device which samples images using a two dimensional matrix. The fields in the interlaced video sequence are commonly sampled at a fixed interval i.e. odd and even fields are not sampled at the same time.

Capturing and displaying images using the technique called *interlace* stems from the need to reduce visible flicker in display devices which relay on rendering an image using pulsing light as opposed to maintaining a continuous light intensity at any given pixel. In the case of cathode ray tubes (CRT), the image is rendered using a beam of electrons which quickly moves across the screen lighting individual phosphorus pixels which slowly fade after being energized by the electron beam. By varying the intensity of the beam the desired image may be rendered. The frequency at which the CRT image is rendered is called the refresh rate. The refresh rate needs to be high enough to fool the observer that the light has a constant intensity or the image will appear to flicker.

When creating the first television standards a few trade-offs had to be made. Since bandwidth is a limited resource there was a need to constrain the frame rate and resolution of the video signal to some reasonable level. As the frame rate is directly proportional to the refresh rate tests were made to see which refresh rate was needed to reduce flicker to an acceptable level. An acceptable frame rate for creating fluent motion is 25 FPS but a refresh rate of 25Hz caused serious flickering in the display device.

Doubling the refresh rate to 50Hz alleviated the worst problems. However, transmitting images at 50 FPS would require a big reduction in resolution to maintain the bandwidth requirements.

The solution to the problem was found by transmitting television as a sequence of half frames or fields containing only every other line. Every other line was then rendered on the CRT display, alternating between sets of odd and even lines. This technique proved effective in reducing flicker while maintaining the bandwidth requirements and the 25 FPS frame rate. [11]

3.1.1 Camera

As a camera is most often the source of video material it is interesting to look at a few properties of it. A film camera registers images by exposing the light sensitive film to light by opening a rotating shutter. The shutter needs to move because the film is constantly moving as well. The shutter is open across a large portion of the frame time which means that the image stored will be sampled over a large proportion of the frame time [14]. A video camera on the other hand utilizes a CCD array registering the image at a short interval and in the case of interlaced video only across every other line. We can also assume that the CCD will always generate fields which are slightly vertically interpolated due to limitations of CCD technology [1].

3.1.2 Monitor

While interlacing has a big impact on such modern display devises as LCD and plasma TV:s we will focus on the common cathode ray tube technology which, as was previously described, was the reason for creating an interlaced video format in the first place. A few interesting properties of video transmission other than interlace are caused by limitations of CRT displays. One of these properties is that the transmitted signal needs to be black while the cathode ray moves back across the screen to render the next line. The signal also needs to be black during the vertical blanking interval to enable the CRT to properly sync the beam to the incoming signal. If this synchronization fails the viewer will see the characteristic rolling image of old TV sets. A few video lines are also above or below the edge of the visible screen area which makes it possible to transmit extra information such as teletext or closed caption subtitles.

As the line moves down the screen in discrete steps it is obvious that the vertical resolution is determined by the number of steps but what is the horizontal resolution? A first guess would be that the horizontal resolution is as high as the representable sampling rate or bandwidth which we can fit in the incoming signal. While this is perfectly true there is also one more thing which needs to be considered and that is the shadow mask. Just behind the phosphor layer, which consists of groups of three sub-pixel elements, is a surface with a single hole per pixel. The purpose of this mask is to make sure that each of the three adjacent electron guns beam directly on their respective red, green and blue sub-pixel. It can be concluded that the resolution of a CRT is determined by the shadow mask in conjunction with the placement of sub-pixel elements.

One relevant problem that must be considered when displaying images on a CRT is that of interline flicker. If a single line is much brighter than the adjacent lines, this general area will only be lit every second field or at 25Hz and will be flickering quite visibly. If, however, *two* adjacent lines are much brighter than their surroundings the

flickering will be greatly reduced while still noticeable. Thus the smoother the light intensity of an image is (in the vertical direction) the less likely it is to flicker when displayed.

3.2 The need to deinterlace

The need to deinterlace a video sequence arises in many different situations both on the consumer side and before broadcast. We will look at a few of them to explore what the specific problems may be.

3.2.1 Display device deinterlacing

As was described in the chapter introduction interlacing is a technique designed to reduce flickering in CRT displays. Plasma and Liquid Crystal Display (LCD) devices do however have different display techniques and requirements. We will use the term flat screen display to refer to these new technologies.

On a flat screen display every single pixel is lit at virtually the same time and the light intensity does not vary over time if a still image is displayed. This means that still image flicker is not a problem for the family of flat screen displays. However, how does such a device go about rendering an image if it is only provided with a sequence of fields? In fact, it needs to deinterlace the image so that frames can be displayed progressively. So virtually all flat screen television displays needs to include a hardware deinterlacer and the quality of that deinterlacing implementation has a strong impact on the quality of the displayed image.

3.2.2 Video deinterlacing

In a broadcaster context some operation may need to be applied to the video. Not all of these require deinterlacing to take place but every operation that rescales, distorts or moves an image around is likely to require some form of deinterlacing before the actual operation takes place. In a broadcaster context we will also need to return the video to an interlaced format after the operation has taken place so that the video can be transmitted in the correct format. In this case it is common to create one frame for each field which effectively doubles the frame rate of the video but makes the re-interlacing operation simple by allowing the re-interlacer to simply remove every other line from the edited sequence to create interlaced material.

3.3 Quality of deinterlacing

An interesting aspect of deinterlacing is how to define and determine if an algorithm is successful and what the goals of the deinterlacer should be. In this Master's Thesis a *perfect deinterlacing* will be defined as restoring an interlaced video sequence into the form it would have been if it had been originally recorded using a progressive camera. However, a perfect deinterlacing is rarely possible nor necessary to reach an acceptable result. The requirements of an acceptable result varies of course with the audience. A few common requirements could be;

1. Should not turn straight lines into jagged lines.

2. Minimal decrease of vertical resolution.
3. Diagonal lines should remain straight.
4. Circles should remain smooth.
5. Small average impact on luminous intensity.
6. Noise insensitivity.
7. Minimize inter-line flickering.

3.4 Basic deinterlacing methods

3.4.1 Single field deinterlacing

Single field deinterlacing creates a complete frame from a single field thus eliminating many of the motion artifacts that may occur when several fields are involved.

Line doubling

Line doubling is one of the simplest deinterlacing algorithms. As the name implies every odd line of a field is doubled to create a complete frame. Thus we accept a loss of vertical resolution by 50% on each frame. This is especially apparent since every other fields needs to be discarded to maintain the frame rate. The advantages of the line doubling algorithm is that it avoids most of the artifacts likely to appear in other algorithms and it is efficient.

Vertical interpolation

Vertical interpolation is an improvement on the line doubling technique. We start with a single odd field and thus every odd line is known. Every even line is unknown and needs to be determined. For each unknown value there will be a known value immediately above and below that value. The vertical interpolation strategy is to determine the unknown values by looking at these vertically adjacent values. In the most simple case this would simply be the average of the value immediately below and above.

$$f(t,x,y) = \frac{f(t,x-1,y) + f(t,x+1,y)}{2} \forall x \mod 2 = 0$$

Note that this strategy as well as many other deinterlacing strategies generates as many frames as fields. We may thus discard every other field if we intend to maintain the frame rate. One advantage over line doubling is that diagonal lines appear smoother.

3.4.2 Multi-field or spatio-temporal deinterlacing

With the single field deinterlacing techniques we can be confident to never create any artifacts caused by motion in the image as all of the data used in the operation were taken from a single point in time. When doing multi-field deinterlacing there is no such guarantee as we will be using fields from different points in time to create a complete frame. However, the gain is that the vertical resolution of the output can in the best possible case be doubled. The main problem is when motion has occurred. If motion

is simply ignored the resulting image will end up with jagged edges around all objects which has moved.

We are also currently describing methods which has no knowledge of the direction or size of motion in the image. The algorithms may nevertheless try to determine if the image has changed in some manner between two adjacent fields or between two fields of the same type. Formal descriptions of the methods described in the following sections can be found in Bellers et.al [5] among others.

Median filtering

Median filtering is a method to eliminate artifacts based on the local color gradient. First the current and previous fields are merged as is into a frame. If a pixel from the previous field is outside of the range described by the vertically adjacent pixels from the current field, the value will clamped to be within that range. Median filtering is very efficient in removing artifacts from the image but will also remove some naturally occurring high frequency patterns. A few interesting properties of median filtering will be described in the results chapter.

Edge detection

Edge detection works by defining an edge detection function. This function will determine for each pixel whether that pixel is part of an edge or not. If the pixel is part of an edge, vertical interpolation is used. If not, then intra-field averaging is used. The reasoning behind this method is that visible artifacts most commonly appear around clearly visible edges.

Weighted filtering

Weighted filtering works by applying a weighted function across a vertical segment of pixels from the previous and current frame. The weighted function may for example be the average of three vertically adjacent pixels. The trade-off is between making the image artifact free or maintaining vertical sharpness.

3.5 Motion compensated methods

The motion compensated (MC) methods first and foremost rely on acquiring information regarding the motion of objects displayed. This includes determining global motions (camera movement) and motion of individual objects. The motion compensated algorithm then tries to use the motion information to stitch moving objects from adjacent frames back together. Even with perfect motion information including rotations, overlaps and other complex motion types, it is impossible to do a perfect full vertical resolution deinterlacing every time. A very simple example is when the camera is panning down with a speed of one vertical line per frame. In this case the best possible progressive restoration will only have half the vertical resolution since the odd and even field from one frame will potentially describe exactly the same data.

3.5.1 Block-based motion search

Block-based motion search is a simple motion detection algorithm that works by trying to match blocks of data with nearby blocks in the adjacent frames. This method is

commonly used in MPEG encoding to find motion vectors. The motion vectors allows 16x16 blocks from previous frames to be reused at a new position in the following frames. It works well for encoding because it is easy to measure the error of a specific motion vector. However, the motion vectors does now have to describe *true motion*[1] as there might be a non-true motion vector with a smaller error. This makes block-based motion search some what unsuitable for deinterlacing as non-true motion vectors will have a negative impact on the result [7].

3.5.2 Gradient-based motion estimation

Gradient-based motion estimation is a mathematical approach to motion detection which features sub-pixel accuracy. The basic idea is to calculate the gradient of every pixel of the image. By maximizing the cross-correlation of the two gradient sets we can find a motion estimation. The cross-correlation can be calculated with multiplication in the frequency domain or by solving an ordinary least squares problem [3].

Argyrio et.al [3] cites Girod who has concluded that "A key performance issue in motion estimation is sub-pixel accuracy". Accurate gradient based motion estimation is however too slow to do in real-time [8].

3.6 Alternative deinterlacing algorithms

3.6.1 Majority selection

Majority selection is based on the idea that by selecting the median value from a number of deinterlacing algorithms, the bad cases can be avoided and an acceptable solution can be achieved. See for example Bellers et.al [4]. The cost of running multiple deinterlacing algorithms may be prohibitive.

3.6.2 Image inpainting

Image inpainting is based on the idea that we may turn a single field into a complete frame using some sophisticated algorithm which is capable of repairing shapes in the image. A few very impressive examples are available in [15] and [6]. Good image inpainting in general is however far to expensive to be done in real-time.

Tschumperlé et.al [14] elaborates on the idea of image inpainting by calculating motion vectors for the inpainted frames and blending pairs of such frames to appropriately blur high motion objects. This allows the result to properly emulate the way images are captured in a film camera, i.e. over a longer period of time than in a video camera.

[1] A true motion vector is aligned with the actual motion of the object it describes.

Chapter 4

Procedure

In brief the work which has been undertaken as a part of this Master's Thesis can be divided in two parts; deinterlacing algorithms and rescaling algorithms. The procedure of implementing a deinterlacer started with acquiring information about research implementations of deinterlacers which fitted the scope of real-time processing, implementing these and exposing any flaws that they may have had and finally formulating, testing and re-iterating a design of an efficient high quality algorithm with very low acceptance for artifacts regardless of video material.

In the case of scaling algorithms the problem is not as ill-behaved as deinterlacing and the research consensus appears to be that there are a few good algorithms. Thus one well known algorithms was selected mainly because it was simple to implement and vectorize.

A few of the most interesting deinterlacing algorithms and why they were rejected will be given in the conclusion of this chapter after the chosen solution has been described.

4.1 Preliminaries

4.1.1 Video clips

The following section will refer to a set of video clips used to test the implementation. These clips are briefly described here.

- Diagonal lines – diagonal lines with different angles. Both stationary and moving.
- Zoneplate 1,2,3 – zoneplate type patterns in different shapes and colors. A zoneplate is a pattern which increases to maximum frequency in some direction. When rescaling, a subset of the image will have too high a frequency to be properly represented in the result. Such areas are expected to become completely gray (if the source was alternating black and white).
- Scrolling text – scrolling and crawling text (vertical/horizontal) both monochromatic and in color.
- Carpet – rare natural high frequency content with camera movement.
- Pans – a set of more ordinary type video clips used to determine sharpness.

4.2 Deinterlacing in the context of aspect ratio conversion

The initial task given by Ardendo was to implement a software aspect ratio converter. A customer of Ardendo was the original party interested in this technology and the in-house knowledge of the problems of aspect ratio conversion (ARC:ing) was limited. It was however our understanding that it would provide an academic challenge to implement such an algorithm based on the input from broadcasting professionals stating that "there are no good software aspect ratio converters". Comparison with such software as well as hardware converters will be presented in the results chapter.

Once the goal of the Master's Thesis was fairly clear, the initial phase of the project was aimed at acquiring knowledge as how to achieve that goal. A lot of time was also spent trying to understand the fundamentals of the problem. The most straight forward solution idea is arguably to properly deinterlace the input video which would yield a frame for each field. Then perform frame based aspect ratio conversation and then re-interlace the material. This led to a lot of deinterlacing papers coming into focus during this initial phase. It was soon clear that to do really good deinterlacing motion compensated algorithms were needed. Motion compensated deinterlacing algorithms depend on an underlying optical flow algorithm to calculate the optical flow of a sequence of images. Calculating the optical flow is the task of assigning a 2D vector (motion vector) to each pixel of an image describing the motion that has occurred since the previous frame, i.e. tracking the motion of moving objects.

As optical flow is quite a big field and there are many different algorithms, quite a few weeks were spent trying to implement a robust optical flow algorithm. Robustness turned out to be the main problem of optical flow algorithms. If some motion vectors are incorrect they are likely to create artifacts in the deinterlaced material. Optical flow algorithms are also very costly to calculate and performance analysis indicated that they would incur a slow down at least an order of magnitude larger than real-time. This study into the field of deinterlacing algorithms was however not a wasted experience partly because of the way it required a complete understanding of the problem of aspect ratio conversion and also in the way that the final algorithm came to be implemented.

As it was still not clear to the author of this thesis how hardware aspect ratio converters worked and what constraints they operated under it was quite necessary to learn more about hardware ARC:s. By reading the specifications one could conclude that the hardware ARC used exactly three sequential fields to create one aspect ratio converted field. It also performed the task in real-time. Since a hardware ARC is an FPGA implementation or similar it can perform a huge amount of calculations in parallel and it seemed unlikely that general computing PC could perform a similar amount of computations as quickly.

At this point the research aspect ratio converter implementations moved in to second phase which emphasized simplicity and speed. Quite a few different techniques were tested during this phase. The basic idea was to explore the relationship between temporal and vertical interpolation to get a better idea of the fundamental problems of deinterlacing. Another important problem that was explored was how to deal with high frequencies occurring in the direction of rescaling as such patterns could cause aliasing problems in the output video.

The more interesting points of the conclusions made, papers read and implementations developed are covered in the following sections. The first part aims to allow the reader to quickly dive into the finer points of deinterlacing without requiring that all

referenced papers are read. As such, it may not describe exactly the order in which ideas in this Master's Thesis were originally formulated. This part will also be the foundation for a short review of the advantages and disadvantages of the techniques described in the theory chapter.

4.2.1 Naive deinterlacing

To get a better understanding of the problem it is very helpful to explore a few more or less naive ideas. We begin by assuming that we are to perform an aspect ratio conversion from 16:9 anamorphic to 4:3 letter box. This requires us to reduce the image in the vertical direction by 75%. A very simple way of achieving this is to remove every fourth line from the image. Assume further that the input video is interlaced. In figure 4.1 we study a set of 8 sequential lines. As can be seen some lines are temporally reordered which is a very undesirable result.

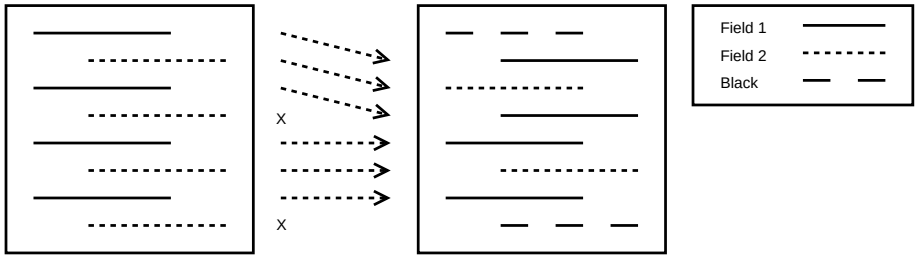

Figure 4.1: To the left, the original frame. To the right an aspect ratio converted frame which has undergone a naive aspect ratio conversion that results in a temporal reordering of some field lines

As this approach was unsuccessful we could attempt to solve this by applying our rescaling on only one field at a time. This would ensure that no information is reordered temporally. Figure 4.2 illustrates the result of an image that has been rescaled field-by-field, it is clear that the previously diagonal lines now carry a wave like pattern.

Figure 4.2: Left: Normal rescaling of diagonal lines. Right: Field by field rescaling of diagonal lines.

As has been shown, field-by-field scaling has some problems with diagonal lines. A more advanced approach would be to stitch together sequential fields to form frames using some technique that avoids creating interlacing artifacts. The interlacing artifacts

appear wherever motion has occurred in the picture sequence. Handling these artifacts requires a good understanding of how sequential fields are related and how that relationship depends on the motion of the objects depicted.

4.2.2 Stationary versus moving objects

As the naive implementations were shown to be insufficient we will need to look at the class of implementations which perform some sort of initial deinterlacing. To do this we will first discuss a few properties of a video sequence. Assume that we have a stationary camera depicting stationary objects. In this case, interlaced video is indistinguishable from progressive video. That is we may simply rescale two sequential fields as a frame and thus acquire the best possible rescaled result.

In the opposite case we have a camera moving quickly and/or quickly moving objects. In this case the relationship between sequential fields is potentially non-existent or at least very low. This would imply that involving previous or future fields in the calculation of a rescaled field is completely unnecessary since no information can (easily) be acquired from it to enhance the quality, i.e. no information is available in the temporal direction.

If we also decide that no motion compensation technique will be employed to stitch moving objects together across fields, the subset of an image which can be considered to be in motion (and thus have the property that no information in the temporal direction can be used) is greatly increased. This decision is based on the conclusion in the preliminaries that such algorithms are to costly to run in real-time.

Given this we have limited the set of useful algorithms to deinterlacing algorithms without advanced motion compensation. Algorithms with simple motion detection algorithms are still to be considered. We will leave the explanation of how motion can be determined using simple algorithms to the following section.

In summary, in the case of motion (more than a few pixels per frame), as was argued, we should not use the previous and future field in the calculation of the current frame. So, in this case only half the frame is known, i.e. the lines from the current field. The only remaining approach is to use some kind of spatial or statistical interpolation to fill in the unknown lines of the frame.

4.2.3 Properties of color and motion

To introduce what is meant by motion, in the context of the class of simple motion detecting algorithm, we first need to discuss a few properties of color or more specifically luminance and chrominance. Luminance is the value or intensity (black and white) property while chrominance is the actual color. A well established fact is that humans are much more sensitive to variations in luminous intensity than chromatic intensity and many video formats use this to their advantage [20].

What we are trying to achieve is to determine motion by looking at some subset of an image. As was stated previously we wish to avoid motion compensated techniques and we are therefore unaware of any true motion occurring in the image sequence. What we are forced to do in this case is to compare sequential pixels in the exact same position and from that observation determine whether motion is present or not. A complicating element is that we cannot determine the direction of motion nor easily differentiate other intensity variations from motions because of other factors such as changed lighting conditions in the image sequence. Hence it would be more sensible to say that it is not motion per se that is identified but rather apparent motion based on

perceived changes. The perceived change would preferably need to take into account the non-linear sensitivity to intensity variations and weight luminance changes versus chrominance changes. If the color change is then mapped to a linear function we may use it as a metric of apparent motion across every pixel of an image in a sequence. A few examples of simple motion sensitive algorithms are available in Bellers et.al [5].

4.2.4 Directions of interpolation

Before we elaborate on the task of trying to find a suitable algorithm we will define what is meant by temporal and vertical direction of interpolation in the context of aspect ratio conversion. Figure 4.3 offers a visual perspective. Information from the previous and future fields are acquired through temporal interpolation. Retrieving information in the vertical direction implies that we use spatially adjacent information sampled at the same time as the unknown sample that we seek the value of.

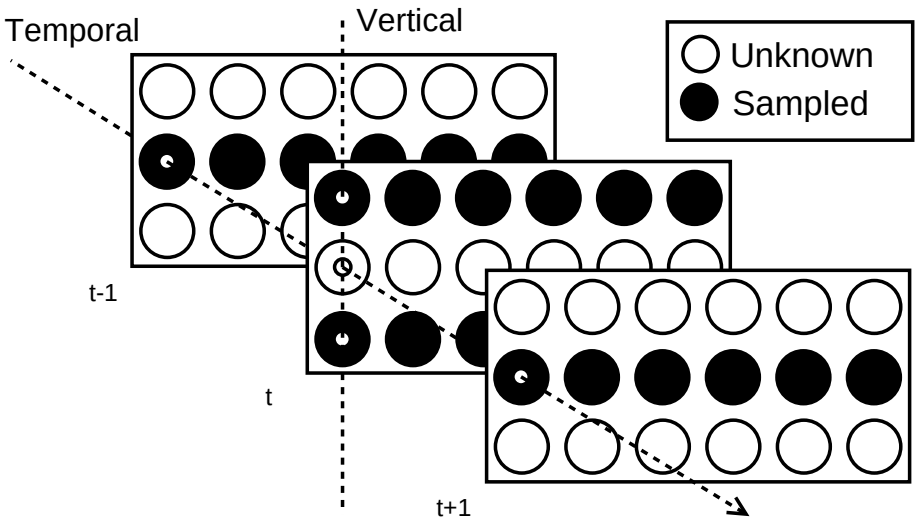

Figure 4.3: Directions of interpolation over an interlaced sequence of images

4.2.5 Other possible algorithms

Before exploring a detailed formulation of a motion detecting deinterlacer we need to mention other deinterlacing algorithms from the theory study which fit in the class of algorithms that we have decided to look at.

Edge detection

Edge detection is based on the idea of identifying areas where artifacts are likely to occur according to some function which should trigger in areas where the human eye is likely to perceive artifacts. Exploring such functions and their impact on the visual quality is most likely a huge task and beyond the scope of this Master's Thesis.

Median filtering

Median filtering is another algorithm which will act in areas where deinterlacing artifacts are likely to occur. Median filtering was found to be a very interesting alternative to a motion weighted implementation and was the algorithm which was used for most of the early work. The results section will explain why a motion weighted algorithm was found to be a better alternative.

4.2.6 Formulating an equation

We will now detail an equation that describes a simple motion weighted algorithm that will glue together the two extremes of moving and stationary objects based on the information of apparent motion. For this we will need a function that maps the apparent motion to a reliability parameter which will directly affect the way that the unknown samples are interpolated. This will be a spatio-temporal interpolation algorithm, the word spatio is derived from the fact that we use spatially adjacent information when we consider the temporal information unreliable and we use the temporal information otherwise. We will define r as a function measuring the reliability of data in the temporal direction where $r = 0$ implies that the information is unreliable and $r = 1$ implies that the information is completely reliable. The temporalValue() function is a temporally interpolating function and the verticalValue() function is a vertically interpolating function.

$$sample_{pos} = \mathrm{r}(pos)\ \mathrm{temporalValue}(pos) + (1 - \mathrm{r}(pos))\ \mathrm{verticalValue}(pos)$$

The given equation will yield the approximation x for a single sample value. Applying this equation to every unknown sample will result in a deinterlaced frame. This frame will be rescaled using a progressive frame scaling as will be described in the following sections.

The basic idea of the algorithm is in summary, based on apparent motion use the most high quality information (temporal) if it is considered reliable otherwise use the safe and artifact free spatial information.

4.2.7 Filling in the blanks

We now have an equation for which we need to define the component functions. In a first attempt at doing so we will aim to do it in an as simple way as possible and discuss possible problems. The temporal and vertical interpolation functions will be chosen to be the average of the closest adjacent values in the direction of interpolation. The r function will be defined as a parameterized function where the parameters serve to adjust the sensitivity to changes in luminance and chrominance respectively. We will separate luminance from chrominance based on the earlier discussion. It is however not clear how to correctly handle chrominance components as it is a pair of values (U, V). To measure the difference between chrominance value pairs we will simply use the Euclidean distance.

$$
\begin{aligned}
r_l(x,y) &= Y(n-1,x,y) - Y(n+1,x,y) \\
r_c(x,y) &= \sqrt{(U(n-1,x,y) - U(n+1,x,y))^2 + (V(n-1,x,y) - V(n+1,x,y))^2} \\
r(x,y) &= S_l r_l(x,y) + S_c r_c(x,y) \\
K_v(n,x,y) &= K(n,x-1,y) - K(n,x+1,y), K \in Y,U,V \\
K_t(n,x,y) &= K(n-1,x,y) - K(n+1,x,y), K \in Y,U,V \\
K(n,x,y) &= \mathrm{r}(x,y) * (K_t(n,x,y)) + (1 - \mathrm{r}(x,y)) * (K_v(n,x,y)), K \in Y,U,V
\end{aligned}
$$

The reliability equation has the parameters S_c and S_l which are the chromatic and luminous sensitivity parameters.

4.2.8 Possible problems

As the equations for building a motion weighted deinterlacing algorithm has been outlined, we will describe a few possible problems that has been found with this approach. Assume that we have an extremely high frequency pattern such that vertical interpolation will yield the color white and temporal interpolation will yield the color black. Now add to this pattern a random Gaussian noise with sufficient intensity to trigger the motion detection algorithm to intermittently do a complete vertical interpolation. As this occurs a relatively low intensity noise has generated a very high intensity noise in the output image. While the author has only found such patterns in special test sequences it is not acceptable that some video inputs look extremely bad as it is very hard to predict when this behaviour might appear.

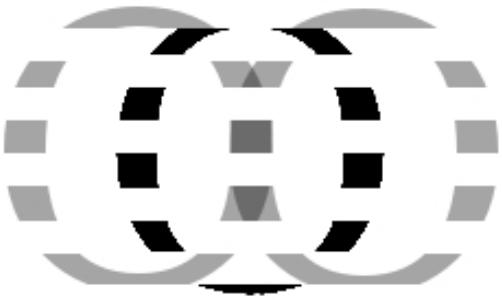

Figure 4.4: The letter 'O' is traveling from left to right. The position of the letter in three fields is shown and this is all the data used to perform the deinterlacing. In the dark gray area overlap occurs which may fool the motion detection algorithm that the pixels are stationary

Another problem that is likely to occur is that pixels may mistakenly be measured as stationary. Assume for example that the capital letter 'O' is moving across the screen in the horizontal direction with a velocity of half a letter per field. In fields 1 and 3 the left and right hand sides of the 'O' character will line up. When calculating the unknown rows in field 2 we will look for differences in field 1 and 3 to detect motion. At the point where pixels accidentally line up the pixels will be considered stationary. As stationary

pixels are temporally interpolated, a black pixel will appear in the center of the 'O' on every second line, see figure 4.4 on the previous page for an example.

Real world experiences also show that noise can cause unwanted fluctuations in the apparent motion variable r. Therefore it was needed to investigate if the r_l and r_l parameters should be considered equal to 0 given that they were below some noise threshold value.

4.2.9 Solutions and trade-offs

Before we try to measure the quality of the image resulting from spatio-temporal interpolation we need to address the problems given above to ascertain that we have a solution that handles these cases and also determine if such a solution has a negative impact on the non-special case video.

Median filtering post processing

Post processing the motion weighted imaged with a median filter would solve many problems that occur when handling scrolling text including the problem described above. Since the apparently stationary white pixel is surrounded by black in the example described it is clear that median filtering would remove the offending pixel. A post processing median filter would however still have the drawbacks of median filtering deinterlaced.

Multi-field motion sensitivity

Since the scrolling text problem described above is caused by the incorrect assumption that pixels are stationary we might try to avoid it by some other means. Just increasing the sensitivity to intensity variations is not sufficient since the mistaken pixels are likely to have a very similar value since scrolling text is most likely in a single color and would thus fool even the most sensitive algorithm. A more elegant way would be to store information about motion at each pixel from previous renderings. Since we are trying to distinguish stationary pixels from moving and it is clear that by definition a stationary pixel will remain in the same position we can conclude that it would make sense to only mark pixels who have been stationary for a few iterations as stationary and mark all others as moving.

A simple way to implement this idea would be by using the following equation, $r = (r_{curr} + r_{old})/2$. The equation is however two-sided in the sense that it also makes stationary pixels less likely to become moving. Hence a mistaken pixel will have a stronger negative impact than it had previously. With a small modification we get:

$$r = \begin{cases} r_{curr} & r_{curr} > r_{old} \\ (r_{curr} + r_{old})/2 & r_{curr} \leq r_{old} \end{cases}$$

We could try to solve the first problem described in the section on possible problems as well. Since the problem was quick switching between vertical and temporal interpolation, some of that is alleviated by slowing the switching in one direction. Slowing the switching in the other direction would mean that more pixels would be temporally interpolated which increases the risk of other artifacts.

Limiting temporal and vertical interpolation

Since the first problem we are trying to solve is caused by the algorithm switching between two extreme values we might attempt to solve it by only allowing the algorithm to operate in a subrange of the possible interpolation strategies. Since perfect vertical interpolation is needed to avoid artifacts in high motion video it is not likely that we can disallow the algorithm to perform vertical interpolation to any degree. It is however more likely that we may disallow the algorithm to do perfect temporal interpolation and for example limit it to always include a minimum of 50% of the vertical interpolation value.

A major drawback of this approach is that when displaying perfectly still images we have accepted not to use all of the information that is available to us and thus have a reduction in vertical resolution. Another problem with limiting the temporal interpolation is that diagonal lines can never be truly straight. This is similar to the previously described issues with field-by-field scaling.

This method is a rather drastic measure and its main purpose is to remove some problems found in a rather extreme test pattern. Whether there is actually a real need to avoid these problems, i.e. that they might occur in real video is a very interesting question that unfortunately cannot be answered in this Master's Thesis.

4.2.10 Summary of algorithm parameters

The parameters which can be adjusted in motioned weighted deinterlacer are the range of the vertical interpolation parameter (r) and the parameters of that function, chromatic sensitivity (S_c) and luminous sensitivity S_l.

The procedure for finding a good value for the chromatic sensitivity parameters was done by looking at fast moving objects in all primary colours and trying to spot interlacing artifacts triggered. If such artifacts could be seen the parameter was dialed up until no artifacts were present. The colour red turned out to be the color were such artifacts could be most easily spotted.

In the case of luminous sensitivity the same procedure was applied except with black and white material.

As described in the previous section a possible way to avoid artifacts is to limit the range of the vertical interpolation parameter to a subrange of the normal $[0, 1]$ range. For example $[0, 0.5]$ could be used to make sure temporal interpolation never exceeds 50%.

4.2.11 Deinterlacing conclusions

The approach to deinterlacing described is a simple and realizable algorithm based on the requirement that it must run much faster than real-time. It is important to determine that the algorithm is indeed robust enough to avoid generating noise or other artifacts in the deinterlaced without sacrificing too much vertical sharpness. The results section will show which of the described problem solution approaches that will actually work, as well as a comparison with the deinterlacers in other aspect ratio converters.

4.3 A review of common deinterlacers

In the light of the problem cases described in the previous section we may continue with a brief review of the algorithms described in the theory chapter.

4.3.1 Line doubling

Line doubling is not surprisingly one of the better algorithms if your criteria is to minimize artifacts. In the case of line doubling all scaling will be applied to lines from a single field and thus all deinterlacing artifacts will go away. The obvious drawback is the decreased vertical resolution. Also line doubling does not gracefully handle diagonal lines which become jagged.

4.3.2 Vertical interpolation

Strict vertical interpolations is an improvement over the line doubling algorithm which better smoothes vertical and diagonal gradients. One input where line doubling could be considered better is if the video material shows a black square on a white background. The vertical interpolation will introduce gray lines at the top and bottom of the black square where white lines meet black lines. This will not happen with the line doubling algorithm.

4.3.3 Median filtering

Median filtering is complicated to summarize and describe but there are a few properties which should be obvious without actually conducting quality comparative testing. The median filtering algorithm clearly preserves monotonically increasing vertical gradients with full vertical resolution. It also efficiently removes most deinterlacing artifacts which may appear since the median filter acts as a hard vertical low pass filter and these artifacts most commonly appear as vertical frequencies at the Nyquist frequency. Median filtering does not suffer from the black, gray, white class of problems caused by vertical interpolation.

There are more advantages to median filtering than can reliably be described here but for this review it will suffice to elaborate on the problem with median filtering described in the previous section. The problem was discovered when median filtering was tested on material containing zoneplate type patterns. The median filtering creates an interference pattern which is caused by the fact that the median filter will darken lines in between where vertical dark lines approach a distance of one separating line. The opposite applies to bright lines. This inability to preserve the local average light intensity is most likely the cause of the interference seen. Whether it is important to avoid creating such patterns is another question. The input pattern required to trigger this behaviour has vertical frequencies well above the Nyquist frequency and images with these properties are by definition impossible to filter so that scaling may be performed safely without aliasing [11].

4.3.4 Weighted filtering

Weighted filtering can be designed as an efficient low pass filter. The fact that the function is constant implies that we need to find a trade-off between vertical resolution

and artifact avoidance which works for all input types and given the requirement of no visible artifacts it is likely that such a filter would excessively sacrifice vertical resolution.

4.4 Image interpolation

An important part of an aspect ratio converter is of course the actual rescaling algorithm which is applied after the deinterlacing step. The implementation of most common interpolation algorithms are quite straightforward with the exception of B-splines as will be described in this section. However, the real problem of the interpolation step lies not in the algorithm per se but rather in the need for good low-pass filtering before rescaling. A short note on why this is needed will be given in the section covering sampling theory.

4.4.1 Sampling theory

While sampling theory is a huge subject a short crash course is given so that the topics of aliasing and low-pass filtering may be better understood. For a better reference see any good text book on sampling theory.

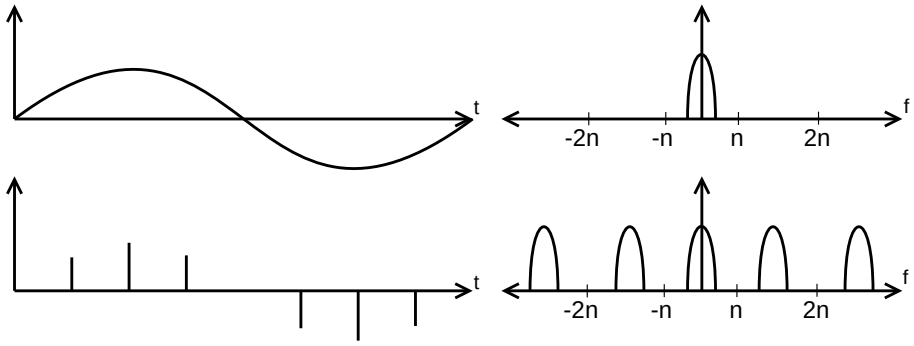

Figure 4.5: Comparison of the frequency response of an analog and sampled signal

Before a signal such a audio or images are sampled they are analog. To store such signals digitally they need to be sampled. Sampling is performed by measuring the signal at discrete intervals and storing the sample values. The sampling frequency is usually measured in Hertz ($\frac{1}{[s]}$). As the signal has been sampled we may compare the frequency response of the analog and digital signals, see figure 4.5. As can be seen the frequency response of the analog signal has been duplicated and lie centered around $\pm n, \pm 2n$ and so on where n is the sampling frequency. This implies that if a signal has a maximum frequency of N and the signal is sampled at $2N$ the repeating waveforms (spectrum replicas) will not overlap. The Nyquist-Shannon sampling theorem states that if a signal is sampled at twice the maximum frequency of the original signal we may also perfectly reconstruct the original signal from the sample values.

A good examples is the common CD where audio is stored with a sampling frequency of 44.1kHz. The sampling theory would imply that to perfectly reconstruct the signal when played out the audio being sampled needs be band-limited to 22.05kHz. As most human listeners are unable to hear such high frequencies a limit of 22.05kHz is an acceptable compromise.

If a signal which is not band-limited below half the sampling frequency (the Nyquist frequency) is sampled, overlap of the spectrum replicas may occur and this is called aliasing. Aliasing results in high-frequency signals being misrepresented and stored as signals at lower frequencies. Aliasing may also occur when downscaling is performed, i.e. a sampled signal is reduced to a slower sampling rate. Assume for example that we have an image where a column is sampled as $[255, 0, 255, 0, 255, 0]$. If we want to rescale the image to half of its length we may simple remove every second sample and get $[255, 255, 255]$. Now if the input signal would change ever so slightly to $[0, 255, 0, 255, 0, 255]$, a change all to common in a video camera, the result would be $[0, 0, 0]$. This illustrates an extreme case of aliasing which could have been avoided if the input signal would have been appropriately low-pass filtered below the Nyquist frequency which would optimally yield the pattern $[127, 127, 127, 127, 127, 127]$ and the aliasing problem would be removed.

Another kind of aliasing may occur as the analog signal is reconstructed from the sample values. If the reconstruction function includes a wider frequency range than intended, the non-overlapping spectrum replicas may be included in the reconstruction and cause what is called post-aliasing.

4.4.2 Low-pass filters

As could be understood from the sampling theory introduction, care needs to be taken when downscaling images. An image that is is downscaled by a factor of α needs to be low-pass filtered to half the sampling rate multiplied by α to make sure that aliasing will not occur. Digital low-pass filters are commonly implemented as infinite impulse response (IIR) or finite impulse response filters (FIR). IIR filter are more accurate but are also more unstable and computationally complex. FIR filter are relatively fast, very stable and can easily be vectorized. The aspect ratio converter implemented utilizes FIR-filters exclusively for performance reasons.

A FIR-filter is a set of P coefficients b_i (or taps) applied to an input signal $x(n)$ in the following manner;

$$y(x) = \sum_{i=0}^{P} b_i x(n-i)$$

A very simple example of a three tap FIR-filter is $b = \{\frac{1}{4}, \frac{2}{4}, \frac{1}{4}\}$. If we apply it to the input signal from the previous example $[255, 0, 255, 0, 255, 0, 255, 0]$ and solve the equation using periodic boundary conditions we get $[127, 127, 127, 127, 127, 127]$ which would indicate that the sample values have been reduced in frequency (by some yet unknown amount). We may find the frequency response of the FIR-filter by using Matlab, see figure 4.6. As can be seen the transition between the pass-band and the stop-band is very smooth. In an ideal filter the transition between the pass and stop-band is sharp so that frequencies very close to the desired cut-off frequency are either perfectly preserved or strongly attenuated. A sharp transition is difficult to define with a FIR-filter and requires more filter coefficients (also known as taps) which implies increased computational complexity.

Designing a FIR-filter is however not done by hand, special purpose software exist to facilitate this task. The FIR-filter is designed by specifying some parameters of the frequency response. The parameters vary greatly between different filter creation methods but commonly the end of the pass-band and the beginning of the stop-band as well as the desired stop-band attenuation may be given. Before developing the subject of

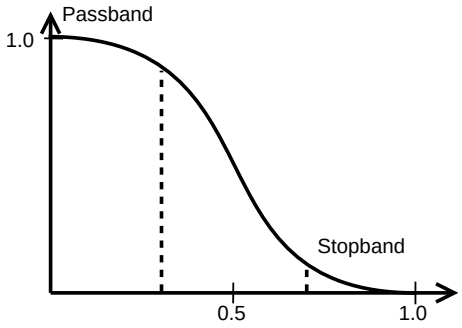

Figure 4.6: The frequency response of the (1,2,1)/4 filter. The diagram illustrates the strength of attenuation at different frequencies, a value of x along the abscissa represents the frequency x times the sampling frequency. The filter illustrated is a low-pass filter since low frequency signals (in the passband) are unaltered and the high frequency signals (in the stopband) are attenuated

FIR-filter creation in more detail, we will discuss some of the problems and requirements of the filters.

4.4.3 Filtering problems

While the task of low-pass filtering is to reduce artifacts such as aliasing, the process may introduce other artifacts which will be described in this section.

As the purpose of a low-pass filter is to remove high-frequencies and since high-frequencies are required to represent sharp transitions in the image, the image will effectively be blurred by the low-pass filtering. Blurring can be minimized by using an as ideal as possible filtering because such a filter will minimize attenuation of signals in the safe pass-band and will remove any signal which may cause aliasing.

There are however a few problems with ideal filters. As was noted earlier they are more computationally expensive since they require more taps and they are also likely to introduce ringing artifacts. Ringing artifacts are apparent as ripples extending out from sharp transitions in the image.

Some of the properties of a FIR-filter can be found through visual inspection of the filter. The filter $[0.025, -0.02, 0.99, -0.02, 0.025]$ has for example a very small impact on the image as the center coefficient is so large. An example of a filter that will cause ringing is;

$$[-0.001, 0.11, -0.05, 0.07, -0.6, 0.69, -0.6, 0.07, -0.05, 0.11, -0.001]$$

This filter will create ringing because of the large coefficients near the beginning and end of the filter. A single vertical line of intensity 255 filtered horizontally will for example be rendered with $0.69 * 255$ at the original position, $0.07 * 255$ at an offset of two lines and with the intensity of $0.11 * 255$ at an offset of four lines.

4.4.4 Designing filters

As designing good FIR-filters is a process of iteration and testing it is not necessarily possible to give a recipe for creating such a filter. We will however try to offer a few good

pointers. First of all it is necessary to determine the cut-off frequency of the low-pass filter. As was stated earlier we need to filter frequencies above the Nyquist frequency of the destination image. If we test such a filter on a video stream known to cause problems, we may determine whether the filtering is sufficient or not. In a video file we may have a sampling rate of 576 Hz and may be rescaling to 432Hz, in this case the Nyquist frequency of the resulting image would be $432/2 = 216$. By repeated testing a cut-off at 216Hz has however turned out to be inadequate for removing aliasing when the filter are created using the method found to be most suitable. A more suitable cut-off appears to be around 170Hz where adequate alias suppression was found for the *carpet* test-material. The blurring effect at that cut-off has been determined acceptable.

The sharpness of the transition between pass and stop-band is mainly determined by the number of taps that we are able to use. Acceptable performance is possible to achieve at 15 taps, so we may attempt to create an as sharp as possible transition using 15 taps centered at 170Hz and inspect how strong the ringing effect is. By applying the filter at sharp lines and text it was clear that ringing was excessive and very visible.

A special post-filtering method called windowing can be applied to the FIR-filter itself to smooth out the filter and lessen the impact of large coefficients in the edges of the filter. After windowing the filters the frequency response will be smoothed out and the transition band will be made smoother. While it might have been possible to specify constraints that would have created a smooth filter in the first place, windowing was found to be a reliable and fast method of achieving the same goal.

The filter used at the moment in the aspect ratio converter for 3/4 conversions is designed using the Parks-McClellan method (160Hz, 180Hz, 2dB, 12dB), rescaled to 0 gain and windowed with a Kaiser window with an alpha of 4.

4.4.5 Interpolation problems

There are a few fundamental problems with image interpolation. Artifacts such as aliasing (post-aliasing), anisotropy, blurring and ringing may be introduced by the process [10].

Blurring may be caused by poor fit to the given sample values or poor frequency response of the interpolating method, see [12] for the frequency response of a few relevant methods.

4.4.6 Separability

A scaling algorithm is separable if a two dimensional rescaling can be performed with identical result by applying two orthogonal one dimensional scaling operations. All interpolating methods used in the implementations part of this Master's Thesis have this property and it is most likely a requirement for achieving good performance. Separability also simplifies the implementation considerably.

4.4.7 Simple interpolation methods

Nearest-neighbour

The nearest-neighbour interpolation algorithm determines the interpolated values by selecting the nearest sample value. When upscaling this will result in blockiness since groups of adjacent pixels will be sampled from the same pixel. See figure 4.7 on the next page for an example.

Figure 4.7: Nearest, Linear, Cubic and Lancoz scaling by a factor of 2 as implemented in GIMP

Linear

Linear interpolation translates the given samples into a continuous piecewise linear function which will allow us to reconstruct samples at the desired points. See figure 4.7 for a linearly interpolated image.

4.4.8 Spline interpolation

A spline is a function defined piecewise by polynomials. Splines are appreciated for their global smoothness and flexibility. Splines may also be employed to perform image interpolation. Under the assumption that the original sampled data was smooth, a spline is more likely to better approximate that data than a linear function which is non-smooth [19].

While we do not aim to define splines mathematically, a few terms will be introduced to help the casual reader. A spline is controlled by its control points. A spline of degree n has $n+1$ control points which determine the shape of the spline over some subset of the control points extension. Some splines intersect their control point and are thus called interpolating splines as they may easily be used for interpolation. Other splines do not intersect their control points and require precomputation before they can be used for interpolation [19].

Interpolating splines

The Catmull-Rom spline is an interpolating spline and also very easy to implement into a software program. A degree of 3 has been chosen for convenience. Using the following equation,

$$q(t) = 0.5 * \begin{pmatrix} 1.0 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{pmatrix} * \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

where $0 < t < 1$ determines a point between the second and third control points. To yield a continuous spline across all control points we simply reuse three control points and shift a new sample into the equation. Using numerical analysis, Keys [9] showed that the Catmull Rom spline is the best interpolating spline for image reconstruction.

Non-interpolating splines

When studying what the best image manipulation software uses to perform high-quality rescaling, the cubic B-spline appears to be a very common choice. Most notable it is used in Adobe Photoshop for high-quality rescaling. However image rescaling algorithms

may be optimized for doing upconversions with large scale factors which is not the case in an aspect ratio converter.

As was described earlier for a non-interpolating spline, the control points are different from the sample values that it fits to. Unser et.al [17][18] describes the algorithm needed to convert samples into control points. While the author of this Master's Thesis does not pretend to understand the algorithm it is nevertheless implemented in the aspect ratio converter software to facilitate a comparison between the Catmull-Rom spline and cubic B-spline.

4.5 Practical work

The more practical task of implementing the algorithms that needed to be tested during the course of this Master's Thesis was done using Matlab and as standalone C programs. The advantage of Matlab is significant when only a quick test is required and computed images can be judged on a still frame basis. The major drawback of Matlab is that if video streams are to be processed, Matlab is far too slow and the work flow for doing so far too time consuming. Hence C was used to implement all algorithms which needed to process video streams.

4.5.1 Implementation

The main implementation of the software aspect ratio converter has from the start been an evolving piece of work as new ideas and approaches have been tested. The main conceptual idea as described earlier in this chapter has nevertheless always been held firm. The parts of the algorithm that have been in flux are the different approaches to solving the problems described in the possible problems section and also how to do proper filtering and rescaling. Before going into those topics a few more practical issues will be covered.

Decoding/Encoding and data format

The decoding and encoding of the MPEG bitstreams are handled by a commercial MPEG-2 codec from MainConcept. The output from the decoder is a decoded frame of video data stored in UYVY format which is a 4:2:2 format, i.e. there are twice as many luminance samples (Y) as chrominance samples (U/V) in the respective channels. More specifically UYVY describes the byte layout across 4-bytes of data. The first byte contains a U channel sample which applies to both Y samples. The second byte is a Y sample which describes the luminous intensity of one pixel. The following two bytes are interpreted similarly. The YUV samples are limited to a subrange of 0-255, for example Y is limited to 16-236 as this maps directly to the volt level of the cathode ray.

Data transport and implications for parallel processing

As an aspect ratio converter sometimes apply both vertical and horizontal scaling and sometimes only one of the two, a pipeline style implementation is used. This makes it possible to create a chain of processing steps. For example the vertical scaler may either output to the encoder or the horizontal scaler. This also implies that each part is run by an independent thread and a two-way scaling operation would be able to utilize four CPU:s at once, one for encoding, two for scaling and one for encoding. Still the scaling

operations are the most expensive and performance would improve on a four-way system if the scaling operations were split into more threads.

4.5.2 Performance

As has been described speed is essential if a software aspect ratio converter is to be competitive in a real world application. The goal was to achieve real-time performance on some reasonably priced hardware.

Choosing a data type

To achieve good performance, Intel®Streaming SIMD Extension also known as SSE was utilized. SIMD stands for Single Instruction Multiple Data and this means that a single operation can be carried out in parallel on several input values at once. As SSE registers are 128 bit long we may choose between doing 2 double precision floating-point operations, 4 single precision floating-point operations, 2 64-bit integer operations, 4 32-bit integer operations, 8 16-bit integer operations or 16 8-bit integer operations. This implies that it would be advantageous to use a data type in the calculations that would allow for as many parallel operations per instruction as possible. While the input values are discrete 8-bit integer samples, many algorithms used require at least a few bits of additional decimal information. Hence it would seem that a conversion to single-precision floating-point would be required. This can be avoided using a technique called fixed point. This allows decimal numbers to be stored as integers and with some additional algorithmic complexity allow for integer operations to correctly operate on the fixed point numbers. The fixed point approach will therefore store the 8-bit integer samples as 16-bit numbers during the calculations.

Fixed point math

Fixed point math works by splitting an integer type into two parts; integral and fractional. In this application 16-bit integers are used. The integral part is the first 8 bits and the fractional part is the 8 last bits, this is commonly denoted as '8.8'. A few examples of how integer operations needs to be altered to handle fixed point numbers is given here;

- Integer to 8.8: shift integer left 8 bits.
- Add: normal integer add.
- Multiply: shift result of multiplication left 8 bits.
- Division: as integer SSE lacks division, a multiplication by a 0.16 fixed point integer may be used. The intermediate 32-bit result needs to be shifted 16 bits to the right.

Capabilities and limitations of SSE

As SSE has a limited instruction set it is important to determine if the operations are sufficient to implement the algorithms in question. It is also important to determine if it is possible to implement the algorithms efficiently. A problem that might strongly hinder efficiency is if the data is laid out in an inappropriate manner so that data needs to be rearranged to a large degree before processing.

On the topic of capability of the instruction set it is important to remember that if a desired operation is not available it should always be possible to implement such an operation without SSE instructions. The main drawback with this approach is if the operation needs to be done between two sequential sets of SSE instructions. In this case, data needs to be written out from the SSE register, the custom operation applied and then read back into SSE register. The integer operations of interest that are available in SSE are 128-bit aligned move (register/memory to register/memory), unaligned move (slower), add/sub (with or without saturation), and/nand/xor, shift left/right, compare/max/min, multiply store high result (high 16 bits), multiply store low result (low 16 bits) and shuffle (rearrange 16-bit samples across a 128-bit register). This small set of instruction has turned out to be enough to implement virtually all the algorithms used. Great care has to be taken to avoid overflow when using a data type with such a narrow numerical range.

As for problems with respect to data layout there is a clear difference between rescaling an image in the vertical direction compared to the horizontal direction. If data is loaded into a register during a vertical operation, every sample is to undergo the same type of operation usually involving (vertically adjacent) values loaded into an other register. In the horizontal case data would be loaded into a register so that values within the same register needs to be added together to perform the appropriate horizontal operation. However such operations do not exist in SSE. Virtually all operations accept a pair of registers, for example addition is carried out as (a1,a2,a3,a4) + (b1,b2,b3,b4) = (a1+b1,a2+b2,a3+b3,a4+b4). Thus no support for adding a1 and a2 together is available and that operation is most likely required for horizontally operating implementations. There is however an acceptable solution to the problem which can be achieved by simply transposing the image buffer so that all operations can be carried out in the vertical direction. While this is expensive it is far less expensive than not using SSE-operations for the horizontal conversion.

Notes on the SSE implementation

As was described previously, data is originally stored in an 8-bit per sample UYVY format. Converting this to three separate buffers of 16-bit fixed point samples is required before the 8.8 SSE instructions can be applied at full speed. SSE can also be used to to perform this initial conversion from 8 bit to 8.8 bit style samples. SSE is also used to do all data operations throughout the deinterlacer, FIR-filter and rescaling algorithms. Some flow control code is written as assembly code but most flow control is done as regular C code which increases the readability and maintainability of the code.

4.6 Software implementation overview

The software aspect ratio converter is written in C/C++-code and can be compiled to a threaded library on linux standard base 1.0 systems. The code is not object oriented but uses C++ for function templates and other syntactical sugar.

4.6.1 Application programming interface overview

The API provided by the library is extremely simple.

- `int arc_init(ardarc_t *ardarc);` Initializes the ardarc_t struct with default parameters.

- `void arc_clearframe(uint8_t *buf, size_t bytes);` Prepares a buffer for repeated use by the getframe call.
- `void arc_putframe(ardarc_t *ardarc, uint8_t *buf, vitc_t *vitc);` Adds a frame for processing by the converted.
- `int arc_getframe(ardarc_t *ardarc, uint8_t *outbuf);` Retrieves a processed frame.

Additional control of the conversion operation is available by manipulating the `ardarc_t.arc_options` struct after initialization.

```
uint16_t width;       // 720
uint16_t height;      // {576, 608, 480, 512}
uint16_t input_format; // 0 - UYVY, MPEG, Normal field order
uint16_t output_format; // 0 - UYVY, MPEG, Normal field order
uint16_t line23margin; // 1 number of lines cleared below line 23
uint8_t video_mode;   // 0 video, 1 reserved, 2 film
uint8_t wss;          // Wide screen signaling
uint8_t conversion;   // 255 = use custom settings from below
float x_scale;        // Scale in horizontal direction
float y_scale;        // Scale in vertical direction
int16_t v_shift;      // Shift image in vertical direction
int16_t h_shift;      // Shift image in horinzontal direction
```

The `x_scale` and `y_scale` parameters have a wide legal range but only certain ranges trigger the use of low pass filtering. This is since the low pass filters are built manually and optimized for very specific downconversion scenarios.

4.6.2 Internals overview

Internally the aspect ratio converter uses a series of frame queues connected between the up to four processing nodes. E.g. decoder → vertical scaler → horizontal scaler → encoder. In this example three queues would be used. The queues are flexible in the sense that we may retrieve frames as fields or frames depending on the need of the processing node. The flexible frame queues greatly simplify the implemention because some processing nodes such as the scalers need to look a three fields in sequence for optimal performance (as is discussed in the deinterlacing section) and the encoder needs to retrieve a frame (two fields).

Vertical scaler internals

When performing vertical scaling the image needs to be deinterlaced before processing. Virtually all deinterlacing code is written as assembly code to be able to use SSE2 operations for performance reasons. The deinterlacer function is called blender for historical reasons.

```
void blender(arc_internal_t *ai, YUV_matrix_t frame,
     YUV_matrix_t *left, YUV_matrix_t *right, int fieldCount);
```

The frame parameter is preprocessed so that the the current field is correctly filled in. After successful processing the remaining lines will be filled in based on the data from the three fields given.

The deinterlaced image rendered will then be fed to the FIR-filter function which will use SSE2 to apply the most appropriate low pass filter to the image. This step is only applied when doing downconversions.

```
template <int IMG_WIDTH>
void fir_filter_SSE(arc_internal_t *ai, matrix_t *in,
     matrix_t *out, int filter_type);
```

In the last step the actual scaler will run. The scaler implementation is a fairly straight forward SSE2 implementation of Catmull-Rom spline interpolation.

```
template <int M_STRIDE, int IMG_WIDTH>
void rescale_SSE(matrix_t *img, matrix_t *img_out,
     double line_step, int m_start);
```

The scaler and filter functions are implemented as C++ template function where vertical and horizontal resolutions are parameters of the template. This allows the compiler to create several versions of these function where template parameters can be written into the code as assembly code immediates.

Horizontal scaler internals

The horizontal scaler works much like the vertical scaler but has some interesting implementation quirks. As mentioned previously there is no need to deinterlace when scaling in the horizontal direction. There is also a huge difference from a vectorization prespective since the direction of operation of the algorithms are now aligned with the structure of the data where as in the vertical case the direction was perpendicular. This makes it much more difficult to write efficient vectorized code as for example additive operations must now added vector elements to each other instead of adding vectors to vectors. Such operations are called horizontal vector operations are a few of them are introduced in SSE3. However for the target platform only SSE2 was available. To solve this the horizontal scaler applies a transpose operations of the image matrix before and after applying the vectorized scaling operations used by the vertical scaler. This has been found to be more efficient than using a non vectorized implmentation of the horizontal scaler.

4.7 Quality testing

The quality testing done in-house at Ardendo utilizes a professional broadcasting CRT monitor. As the algorithm outputs interlaced video the need for a CRT monitor is clear despite the problems with a 50Hz CRT display. Also as hard cases include quickly moving text the video must be played back at full speed. This is due to the fact that a still frame of interlaced video containing moving text is completely illegible.

Chapter 5

Results

The first part of the results section will cover a comparison between some of the most interesting parameters that can be tuned in the algorithms implemented. The second part will cover a comparison of the best parameters found with a few different commercial implementations. We will also cover external testing done by customers of Ardendo.

5.1 Parameters chosen for the deinterlacing algorithm

As was described, the parameters tweaked when testing the motion weighted deinterlacer are the range of the vertical interpolation alpha value (r) and the parameters of that function chromatic sensitivity (S_c) and luminous sensitivity S_l. The minimal value of chromatic sensitivity which was found to be artifact free was $\frac{1}{4}$ and in the case of luminous intensity $\frac{1}{3}$.

Finding a good value for the vertical interpolation limiting is on the other hand completely non-trivial. As mentioned earlier the limiting serves the purpose of avoiding instability in the deinterlacer when faced with a noisy input. If the input is of high quality the limiting may not be necessary. It is however the author's opinion that if the value is clamped to the range $[0, 0.5]$ no significant image quality degradation can be detected and this also sufficiently attenuates the effects of quick switching between maximum vertical interpolation and maximum temporal interpolation.

As mentioned in the procedure section possible thresholds for the luminous and chromatic change parameters (r_l, r_c) were considered. It was subjectively found that variations below four intensities were better to ignore than to include in the calculation of the total apparent motion parameter (r) as such variation commonly occurred in video material due to noise in the recording.

5.2 Results from scaling comparisons

Very limited testing has been done to compare different scaling algorithms. In summary, it was found that Catmull-Rom scaling far outperformed nearest-neighbour or linear scaling. Regarding bi-cubic spline scaling no quality improvements could be discerned and considering the performance impact of bi-cubic splines the Catmull-Rom spline based scaler was selected as the scaler to use.

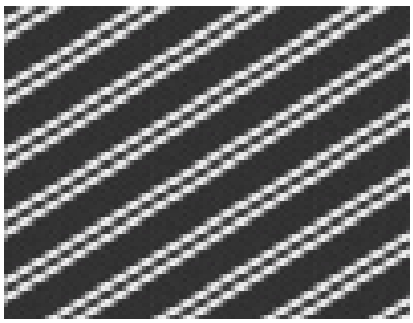

Figure 5.1: Diagonals, Original image

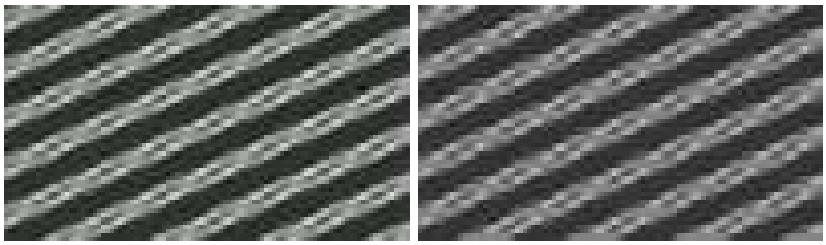

Figure 5.2: Diagonals, Author's implementation left, Avid Newscutter XP right

5.3 Results from comparing other implementations

Three commercial broadcast quality ARC implementations were applied to certain key video sequences and compared to the motion weighted Catmull-Rom ARC. The first one is Avid Newscutter XP which is a heavily used video production suite in the broadcasting industry. The second implementation is Apple's Final Cut Pro (FCP) which is also a broadcasting quality non-linear editor. The third implementation is the hardware ARC from AXON described previously.

There are is also one well known open source implementations which could have been used in the comparison called FFmpeg. However, it turned out that it was not possible to generate a properly interlaced output after scaling had been performed. Most likely because this is not needed when used with personal computers which rarely have the capability to properly send an interlaced video signal to a TV set.

5.3.1 Deinterlacing

This section will cover some difficult video sequence which could potentially break the deinterlacing part of an ARC implementation.

Moving diagonals

Moving diagonals is a good example of motion artifacts as it is clear to the observer what the output should be and their sharpness makes it easy to see any artifacts.

In this comparison the motion weighted Catmull-Rom ARC and Avid ARC produces the least straight lines. Final Cut Pro produces the straightest lines but it is also the blurriest and the thin black lines tends to disappear.

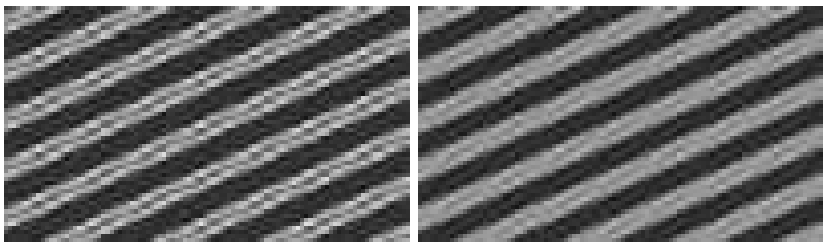

Figure 5.3: Diagonals: AXON left, Final Cut Pro right

Figure 5.4: Flowers, Original image

In the motion weighted ARC the direction of interpolation is always vertical. A possible idea that may be worth exploring to improve the results seen here is if the direction of interpolation would be adaptive and follow a direction perpendicular to the local image gradient. This should make interpolated values follow the direction of the line more closely.

Sharpness

The image compared in this case is a detail from a larger close up of a house. The camera is zooming in and there is no other motion in the image. Despite the extremely high bitrate there are noticeable compression artifacts and errors may be introduced differently since all clips have been encoded using different DV encoders.

The most interesting problems is the slight blurriness and lack of saturation in the Avid case. The FCP version suffers from color bleeding around the red areas.

Figure 5.5: Flowers, Author's implementation left, Avid Newscutter XP right

Figure 5.6: Flowers: AXON left, Final Cut Pro right

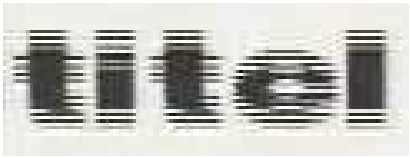

Figure 5.7: An example still frame from an interlaced clip of rolling (vertically scrolling) text

Scrolling text

To compare the performance of the ARC:s with scrolling or rolling text it is not possible to look at still frames from the material. An example of why this is can be seen in figure 5.3.1. The image looks very distorted but this is caused by the fact that the rolling text has moved several video lines between the first and second frame. So when this video sequence is displayed on a CRT monitor the text looks absolutely correct.

When comparing scrolling text it is important to look closely at the area the text is passing through. Jumping pixels may appear near the text. Larger characters may also appear to shimmer.

From comparison of this material displayed on a CRT monitor it has been found that the motion weighted ARC and the AXON have problems with flashing pixels when white text is scrolled very quickly in the vertical direction. The FCP ARC features a mild shimmering especially in characters with narrow edges like ”e”. The Avid ARC seemed very robust with all kinds of scrolling text.

Zone plates

A brief comparison of ARC:ing zone plates reveals that the Avid ARC seems to be sensitive to noise in non-moving high frequency areas as there is visible popping and minor artifacts visible. The weighted motion Catmull-Rom ARC tends to introduce blurred horizontal lines. Final Cut Pro produces the blurriest version as the highest frequency areas have turned completely gray. This blurriness might not be a disadvantage as it does not seem to appear on less extreme patterns.

5.3.2 Rescaling

Image filtering and scaling algorithms are quite difficult to evaluate. A few results will be summarized which have strongly shaped the implementation.

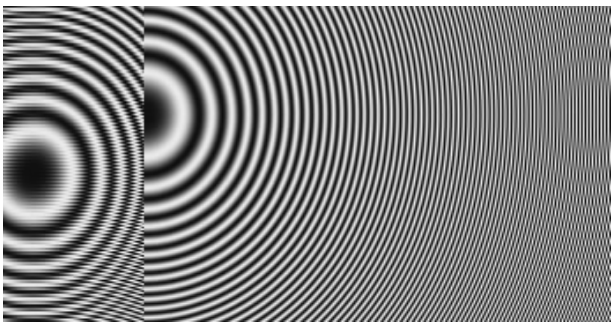

Figure 5.8: Zoneplate pattern where the left part is moving and the right is stationary

Low-pass filtering

To evaluate the low-pass filtering step we need to determine if the low-pass filter used manages to eliminate aliasing while still avoiding to blur the image. To show this still images are not sufficient, instead actual video material must be used. The problem is made harder by the fact that only very specific patterns will yield aliasing when downconverted. However all types of material can be evaluated to determine if excessive blurring has occurred.

The test sequence used is the "carpet" material described earlier. The aliasing suppression performance is compared with the AXON ARC.

For this material the aliasing suppression is far from perfect as there is still a visible moire pattern after filtering and scaling has performed. A pessimistic estimate is that 60% has been removed. The AXON ARC performs better but it also noticeably blurs the material in the affected regions where as the implementation made as part of this Master's Thesis maintains a high degree of sharpness throughout.

Scaling

The Catmull-Rom spline based scaling algorithm has been compared mainly with cubic splines which is a far more well known scaling algorithm. However, it has not been possible to identify any measurable advantage for either algorithm when doing side by side comparisons. As it is near impossible to find artifacts that are attributable to the actual scaling algorithm no comparison to the other complete ARC implementations can be made.

5.4 External testing

As part of using the ARC in production at Red Bee Media the quality was tested using 50 hours of material and the output was reviewed by senior QC staff. Except for a few initial omissions related to how the bottom and top lines were handled in different resolutions, only one artifact required changes in the implementation. That artifact can be seen in figure 5.4 on the following page. The artifact was most likely caused by a combination of misaligned chroma samples in the scaler and the fact that the chroma was scaled as is, i.e. at the subsampled (4:2:2) resolution. The artifact disappeared once the chroma samples were upconverted to the resolution of the image before being

Figure 5.9: Left image is the (manually scaled) original and right is created by a previous version of the algorithm described in this Master's thesis. The right hand image has an has an unwanted artifact in the form of a gray shade in the bottom right corner of letter e

provided to the scaler. Once this problem was solved and the remaining material was screened the ARC was approved for production use.

5.5 Performance

The speed of processing of the implementation has been evaluated on an a Xeon 3050 2.13GHz dual core machine. Run-time measurements for a sequence of 1000 frames of PAL material (25 fps) are available in the table below.

Frames	Avg. wall clock time (s)	Frame rate	Operation
1000	37.45	26.7	Vert. + Horiz. scaler & filtering
1000	29.55	33.8	Horizontal scaler
1000	32.10	31.2	Vertical scaler

5.6 Summary

In total the result section shows that there are measurable differences between the implemented algorithm and other professional software and hardware implementations. These differences show that all these implementations have advantages and disadvantages depending on the video material tested and no clear winner can be determined.

Chapter 6

Conclusions

We conclude that it is possible to implement a software aspect ratio converter with real-time performance on par with professional ARC implementations using the combination of a modified spatio-temporal deinterlacing technique coupled with finite impulse response low-pass filtering and spline based rescaling. Concluding scientifically that the quality is on par with professional ARC implementations is more difficult. The result section shows that there are artifacts and possible areas of improvement but that is a fact which is shared between all tested implementation. The strongest argument with respect to quality is that the implementation has been tested and found sufficiently good to please professional viewers at Red Bee Media (formerly known as BBC Broadcasting) during special quality control across a range of material spanning a total of 50 hours. This is the main motivation for the completion of goals 1a, 2a and 2b all relating to the quality of the conversion. The completion of goals 1b and 2c relating to performance are shown in the results chapter.

The statement that there are no good software aspect ratio converters has not been found true. The hardware and software ARC:s compared performed on a similar level. The statement may be based on the still frame evaluations of deinterlaced material which can easily fool an observer that artifacts are present. Since still frame observation are much more likely to occur during video editing (which uses software ARC:s) the software ARC:s may have been evaluated in an unfavourable light.

6.1 Limitations

As the implementation is used in production it can be considered quite complete in the context it is used. This means that it supports a limited set of resolutions mainly the standard PAL and NTSC resolutions of 720x576 and 720x480 respectively. The algorithm itself has no such limitations and it has in fact been used to implement HD (1920x1080) to SD (720x576) downconversions in other applications. The only necessary modification was to determine good filter parameters for the scale factors used.

6.2 Future work

Regarding the scope of the Master's Thesis we conclude that the original goals have been achieved although not in the normal 20 weeks but rather in 30-32 weeks. There are of

course still areas where much additional work can be done. A better understanding of good low pass filtering of images is needed to better make sure that aliasing or blurring is avoided. Different scalers, for example bicubic interpolation, should be more carefully evaluated to find the most suitable algorithm. Of course many different approaches on deinterlacing can be used and with faster computers more time can be spent analyzing complex information in the image like global motion and motion flow.

Chapter 7

Acknowledgements

I would like to thank the following for their assistance in completing this Master's Thesis. Ardendo has provided me with the time to write this report and the equipment needed to facilitate the development. Ian Wimsett and Iain Stoddart at Red Bee Media have provided valuable ARC insights and feedback on the ARC testing. My internal supervisor Thomas Pederson has provided many useful suggestions for improvement of the academic quality of this Master's Thesis and my external supervisor Isak Jonsson has helped integrate the software in several Ardendo applications. Ben Norman has provided help with English grammar and punctuation.

References

[1] John Amanatides and Don P. Mitchell. Antialiasing of interlaced video animation. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 77–85, New York, NY, USA, 1990. ACM Press.

[2] Ardendo AB. Webpage. http://www.ardendo.com accessed March 19, 2009.

[3] V. Argyriou and T. Vlachos. Using gradient correlation for sub-pixel motion estimation of video sequences. In *IEEE Proc. ICASSP, Vol. III*, pages 329–332, 2004.

[4] E. B. Bellers and G. De Haan. Majority-selection de-interlacing : An advanced motion-compensated spatio-temporal interpolation technique for interlaced video. In *Proceedings of Image and video communications and processing 2000*, pages 386–395. International Society for Optical Engineering, 2000.

[5] E.B. Bellers and G. de Haan. Advanced de-interlacing techniques. In *Proc. ProRISC/IEEE Workshop on Circuits, Systems and Signal Processing*, pages 7–17. IEEE Computer Society, 1996.

[6] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[7] Yen-Kuang Chen. *True Motion Estimation - Theory, Application, and Implementation.* PhD thesis. www.geocities.com/ykchen913/thesis.pdf.

[8] Francis Kelly and Anil Kokaram. Graphics hardware for gradient based motion estimation. In *Embedded Processors for Multimedia and Communications*, pages 92–103, San Jose, California, 2004.

[9] R. G. Keys. Cubic Convolution Interpolation for Digital Image Processing. *IEEE Transactions on Acoustics Speech and Signal Processing*, 29:1153–1160, 1981.

[10] Don P. Mitchell and Arun N. Netravali. Reconstruction filters in computer-graphics. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 221–228, New York, NY, USA, 1988. ACM Press.

[11] Charles Poynton. *Digital Video and HDTV Algorithms and Interfaces.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[12] Miklós Póth. Image interpolation techniques. In *Proceedings of the SISY 2004*, 2004.

[13] A. Smirnov and T. C. Chiueh. An implementation of a fir filter on a gpu, 2006. www.ecsl.cs.sunysb.edu/fir/fir.ps.

[14] David Tschumperlé and Bernard Besserer. High quality deinterlacing using inpainting and shutter-model directed temporal interpolation. In *Proceedings of ICCVG 2004*. Kluwer Verlag, 2004.

[15] David Tschumperlé and R. Deriche. Vector-valued image regularization with pdes: A common framework for different applications. citeseer.ist.psu.edu/581387.html.

[16] International Telecommunications Union. Recommendation itu-r bt.601, encoding parameters of digital television for studios, 1992.

[17] M. Unser, A. Aldroubi, and M. Eden. B-Spline signal processing: Part I - Theory. *IEEE Trans. Signal Process.*, 41(2):821–833. citeseer.ifi.unizh.ch/unser93bspline.html.

[18] M. Unser, A. Aldroubi, and M. Eden. B-Spline signal processing: Part II—Efficient design and applications. *IEEE Transactions on Signal Processing*, 41(2):834–848, February 1993.

[19] Michael Unser. Splines: A perfect fit for signal/image processing. citeseer.ist.psu.edu/unser99splines.html.

[20] John Watkinson. *The Art of Digital Video*. Focal Press, 2000.