

Master of Science Thesis
Department of Computer Science
Lund Institute of Technology

Real-time Video Effects

Using Programmable Graphics Cards

Videoeffekter i realtid med programmerbara grafikkort

Klas Skogmar <klas@skogmar.com>

Supervisor:
Lennart Ohlsson <lennart.ohlsson@cs.lth.se>

Abstract

The thesis treats the use of modern consumer graphics cards for doing real-time manipulations with high-resolution video.

A demo program was developed for color correcting video using the graphics card instead of using the processor. The demo program shows the abilities of this method for doing image-specific tasks.

Other programs were also used to test this method's performance of rendering video to the screen, and to test the transfer rate between the card and the processor.

My conclusion is that this is something that has a great potential, but the transfer speed from the graphics card to the memory has to be improved. This is a software issue, but this problem is about to be resolved with driver updates. Already driver updates have enhanced the performance by several hundred percent.

It is already a technique that is capable of enhancing the visualization of the modifications that are made. This is a sufficient reason for making programs that utilize the graphics card instead of the processor.

Contents

REAL-TIME VIDEO EFFECTS	1
USING PROGRAMMABLE GRAPHICS CARDS.....	1
Abstract.....	2
1. Introduction.....	5
Background.....	5
Emergence of high resolution broadcasts.....	6
Higher dynamic range.....	7
Modifying video in real-time.....	8
The problems.....	8
2. Programmable graphics cards.....	10
3. Effects on the graphics card.....	12
Color correction.....	12
Masking.....	13
Color keying.....	14
Compositing.....	14
Transitions.....	14
Painting.....	15
3D-effects.....	15
Limitations.....	15
4. High resolution challenges.....	16
Disc space.....	17
Transfer speeds.....	17
Resolution limitations.....	18
Using the graphics card.....	18
5. Working practice.....	20
Scandvision Interview.....	20
6. Existing technology.....	23
Dedicated hardware.....	23
OpenGL:s Imaging subset.....	23
DirectX 9.....	24
Quicktime.....	25
ATI's technologies.....	25
Nvidia's technologies.....	25
CinePaint.....	25
7. Building media tools.....	27
Why create a demo program?.....	27
Building media tools using DirectShow and graphics cards.....	27
Accessing the graphics card.....	29
Transform filters.....	29

Connecting it with a graphical user interface.....	30
DirectShow-filter.....	30
The MFC GUI.....	31
Testing the program.....	32
8. Performance analysis.....	34
ATI's demo program.....	34
OpenEXR's EXRDisplay.....	35
Render to memory.....	36
Matt Craighead's test program.....	36
Test results.....	37
9. Conclusions.....	40
The potential of using the graphics card.....	40
The future.....	41
Appendix A – Formats.....	43
Open-EXR.....	43
DPX/Cineon.....	43
TIFF.....	44
DV.....	44
Digi-beta.....	44
HDTV.....	44
Appendix B – Graphics cards.....	45
Older cards.....	45
Nvidia's Geforce FX.....	45
ATI's R300.....	45
Bibliography.....	46
Books:.....	46
Papers:.....	46
Internet:.....	46

1. Introduction

The thesis will show how high-resolution, high dynamic range video can be modified in real-time using the new generation of programmable graphics cards.

Background

The need for computer aid in editing/color correcting film and video has been obvious since it was first made possible. Today there are relatively cheap solutions in real-time for DV resolutions (720x576 PAL/720x480 NTSC). For HDTV, the picture is different. Real-time solutions for editing/correcting HDTV cost around 200,000-300,000 USD. This thesis will try to determine if the new high quality programmable consumer graphics cards with programmable pixel and vertex shaders will be able to do HD video manipulating in real-time, or near real-time. The thesis will try to deliver a summary of the research and will also try to foresee some future developments.

Before 2002, 2D image manipulation had to be done by dedicated hardware parts. Using hardware support made the solutions very specific, and new hardware had to be bought, when the need exceeded the current installations. When programmable graphics cards turned up in 2002, this seemed like a good solution that could deliver both speed through hardware, but still serve as a general programmable platform.

For the technique to be useful there need to be real speed advantages when using the graphics card compared to the processor. Testing data will be able to tell if the technique is valuable today, or if this belongs to the future. As part of the master thesis a demo program is developed for testing purposes, and to show what can be done using a DirectShow filter that uses the graphics card for manipulating the pixel values. The results that are analyzed, is the time it takes to get a frame from memory, upload it to the graphics card and then fetched back to the main memory. This time is compared with those obtained when using the processor.

Other times, like how long time it takes to read/write to/from hard drive,

is another potential problem. This thesis assumes that this is not a bottleneck. Hard drive performance is covered elsewhere, but the thesis anyway covers some basic storage requirements for a future low-end video editing station.

For the graphics card usage to become a success, not only needs it to be faster than the processor, but it should also be able to do all the currently available effects. Example of effects that are common today are: color correction, color keying, masking, transitions, compositing, filters and painting. Generally these will quite easily be implemented using the graphics card as well. In chapter two many effects will be analyzed with respect to the implementation on the graphics card.

Emergence of high resolution broadcasts

High resolution is needed when the final product will be shown with a high resolution, that is in cinemas or on HDTV, or when modifications are going to be made on the source video.

Since the broadcasting of television began in the fifties, the resolution of the television has stayed the same. When color TV transmissions replaced the ones in black and white, it stayed compatible by using one black and white channel and two color channels. The television standards are different in different regions. The dominating standards are Americas NTSC (National Television Standards Committee) and Europe's PAL. Some countries have chosen to use an alternative, SECAM. All current television standards are interlaced¹ with PAL having a slightly higher vertical resolution. PAL has a resolution of approximately 720x576, and NTSC 720x486², but PAL has fewer fields (50 fields/s) compared to NTSC's 60 fields/s.

If you want to compare the resolution of TV with film that is used to record movies, you have to compare the resolution of the film when scanned. The most common format³ for recording movies is 35 mm (full aperture), which usually is scanned at either 4K (4096 pixels in width) or 2K (half = 2048 pixels). The height then depends on the ratio between width and height, which usually is 1.85:1 or 2.35:1. This resolution is around 3 to 16 times larger resolution than the one used in today's TVs.

HDTV is emerging as a new standard for television⁴. It supports several resolutions and both interlaced and progressive scan. The highest HDTV resolution is very similar in resolution to the "half" resolution used when scanning 35mm⁵. It is thus sufficient for display on cinematic sized/high

¹ Interlacing video is when half the vertical resolution is showed in each field, using the previous field to fill in the gaps. This way the frame rate is doubled, giving a smoother experience, but viewing each field individually reveals the lower resolution. Showing each frame with full resolution is called progressive scan.

² [1] p 319

³ according to [1]

⁴ HDTV means High Definition TV. Usually the abbreviation HDTV is used where HD should have been used instead. Today's televisions are usually referred to as using SD – Standard Definition.

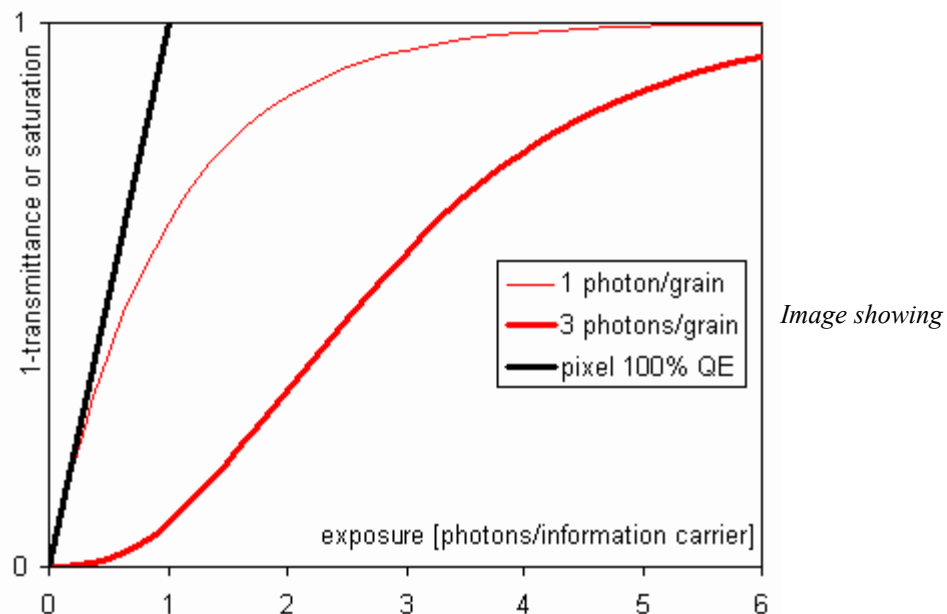
⁵ HDTV has 1920 pixels in width, compared to the half resolution, which usually is called 2k. There is not a defined exact width for 2k, but often 2048 is used since it is a power of two.

resolution displays⁶.

Higher dynamic range

Today, there is an increasing interest in the movie production industry for programs and hardware that can handle HDR (high dynamic range) images. This means that they have more information than can be displayed or perceived; this limit is around 8 bits/color. The extra bits can later be used to change the exposure of the images in post-production, which limits the need for re-scanning the movie. Traditional film has a higher dynamic range than the 8 bits that are common in image manipulation programs like Photoshop. The film's sensitivity is also non-linear. It is more sensible to intensity changes in very bright and dark parts of images, which means that you can usually distinguish differences in colors in an image even if it is taken directly at the sun for example. This also means that digital camera equipment will need the extra bits of information, since then it is impossible to go back and rescan sequences for changes. The digital sensors also have more contrast compared to film.

Digital image sensors have the potential of becoming even better than film, since for digital image sensors “A high dynamic range is not in contradiction with a high sensitivity”⁷. But film still has a much greater tolerance with over exposure. To solve the dynamic range problems in digital sensors, the Japanese company Fuji has developed a new digital image sensor, which is called SuperCCD SR. It alternates sensitive photodiodes, intended to capture shadows and midtones, with less sensitive, that captures the bright areas. This technology will be launched in early summer in consumer cameras. If it works, this concept will soon be used in the movie industry, in which cameras cost several hundred times as much and high dynamic range is very important.



⁶ For example, George Lucas, recorded Star Wars: Episode II using digital HD resolution cameras, and then transferring it to film.

⁷ According to [16] (OEEPE)

Modifying video in real-time

When manipulating images, there is a need for instant feedback. Actually, the industry in some cases still uses analogue editing stations because it can display transitions immediately. Using tapes, it is also easy to track frames, and the response times are negligible.

If the computer is not powerful enough to render the whole image in real-time, most programs utilize a preview mode that shows changes on a small part of the image. This area is called area of interest (AOI). This is a good way of giving instant feedback, but also limits the general impression of how the changes will affect the image. This is especially important when dealing with video, since the preview will consist of a clip, rather than a picture. Since many pictures are involved, the rendering needs to be done on the entire image in the same speed as the frame rate (usually 24-30 fps⁸). This means that each image will need to be loaded, rendered and stored in less than 30-40 ms.

Another way of increasing the speed is to work with a low resolution copy – a proxy⁹. The proxy can be edited and modified with ease, and when the work is finished all editing and algorithms are applied to the original copy.

Why graphics cards?

The idea behind using graphics cards, is that they are developed to do parallel brutal force calculations per pixel basis¹⁰. They are also very cheap, since they are produced in large quantities. If it is possible to use the graphics card, then many programs like Photoshop could start transferring some functionality to the graphics card. This would mean faster programs, with more functionality without having to pay for expensive hardware.

Recent additions to the programmability have increased the flexibility even further. This means that these graphics cards can give hardware accelerated performance of almost any operation. What can be done is covered in the next section.

The problems

The following problems need to be addressed:

- How can graphics cards be used for displaying and altering video?
- What kinds of effects are suitable for a 3D environment?
- Is it possible to speed up effects by using the graphics card?

⁸ Frames per second

⁹ [1] p. 130

¹⁰ The modern graphic cards are also very fast in doing vertex calculations. That feature is important in 3D-games but not when rendering 2D-video.

- How are the transfer rates affecting the system?
- How much disc space is required and how does this affect the system?

2. Programmable graphics cards

Today graphics cards have even more transistors than processors¹¹. This made the graphics card manufacturer Nvidia introduce the notion GPU, an abbreviation for Graphics Processing Unit, to place it at the same level as the processor, the CPU. The reason for increasing the amount of transistors in graphic cards is that the manufacturers have added more flexibility to the cards. Nowadays the cards can do more than just the traditional 3D pipeline.

This added new flexibility opens up new possibilities in 3D, but also for doing image manipulations. In 3D games the programmability of the cards makes it possible to add a lot of new features. This can be for example bump-mapping, displacement mapping, toon-shading and making the environment of the 3D worlds dynamic.

Despite the fact that the technology was created to enhance 3D, it can be used for 2D as well. This is accomplished by applying an image as a texture on a polygon that covers the display area. When the texture is rendered to the screen there is a possibility to use the programmable pipeline to modify the image.

The programmable pipeline consists of two steps: first vertex shaders, and then pixel shaders. Vertices are the corners of the polygons. When each polygon is rendered, each pixel's color value is fetched from its polygon's texture. The pixel shaders can then do a computation for each pixel, like merging it with another texture, subtract a constant from a color component or multiplying it with a transformation matrix.

¹¹ ATI's R300 have 110 million transistors. This can be compared with Intel's

The programmable GPU's started with Nvidia's Geforce 3, and since then all but the cheapest graphics cards have vertex and pixel shaders. Usually there are several parallel pipelines for the shaders, which make the cards handle even more data simultaneously.

The first versions of pixel shaders supported only a very limited set of operations. Only 8 color operations and 8 texture operations could be performed on each pixel, with no support for conditional blocks¹². Recent versions have much more operations, more bits for each color and added support for conditional blocks.

¹² A conditional block is a block of code that is only run if a certain requirement is fulfilled. The reason why they weren't allowed is that then you don't know how many operations you will perform. And since there are a limited amount of operations, this makes it difficult.

3. Effects on the graphics card

There are some differences that have to be considered when doing effects on graphics cards compared to doing them on the processor. The graphics card is more limited, since a limited amount of operations can be performed in one pass. There are also some restrictions on conditional jumps, but the recent cards can do much here as well. Actually the languages for programming the graphics card now resemble those for programming the processor. Cg¹³ and HLSL¹⁴ are very similar to C. This makes it easy to port code written for the processor to the graphics card.

I will describe some effects that are needed in the movie industry, and discuss the advantages/disadvantages of porting them to the graphics card.

Color correction

Color correction is done to change the appearance of the colors in a sequence. Today it is used in movies to make a certain impression on parts of the movie. For example the movie Traffic¹⁵ made the parts that took place in Mexico yellower, and the parts that took part in America bluer. Color correction is also used to correct color faults because of difficult lighting conditions when filming a scene. Some outdoor scenes can have different lighting that makes the color change between shots.

RGB is the most common format for storing images. RGB means red, green and blue, that together can make up all other colors¹⁶. But it is not always the case that one wants to change the perceived colors in red, green and blue. The restriction to these three colors makes it difficult to

¹³ Cg (C for graphics) is a language developed by Nvidia. It supports both OpenGL and Direct3D.

¹⁴ HLSL is an abbreviation for High Level Shading Language and is developed by Microsoft.

¹⁵ Awarded with 4 Academy Awards

¹⁶ Some color spaces have a greater range, like Lab. The color range is also limited by the number of bit each color has (as described in the previous section).

change other perceivable characteristics of the images, like brightness, saturation, hue or luminance. Generating a grayscale image from an RGB image should for example not be the straightforward way of taking the average of all the three colors. Since the colors have different luminance values, taking 29.9% Red + 58.7% Green + 11.4% Blue¹⁷ instead results in much better results. These images have more contrast and are more distinct compared to an averaged image.

There are many ways to make color correction. Usually the Color space axes are changed to some more useful ones. Then the values on these axes are changed, and then the colors are transformed back to the previous ones, usually RGB. Some common color spaces are: RGB, HSV, CMY(K), CIE LAB, CIE LUV, YIQ, YUV, YCbCr, and others. Some of these color spaces can be obtained from RGB by just multiplying it with a matrix, but some require conditional statements by checking borders and min/max values.

For example HSV is a color space that represents the colors as hue, saturation and value. This makes it more intuitive to alter the colors' characteristics because you can change the saturation with one control instead of manipulating three different controls.

Because some conversions require conditional statements, it means that those color spaces can only be obtained with pixel shader version 2.x or later when using graphics cards. To calculate the value (V-part in HSV) from RGB, the maximum values of R, G or B are taken. This operation cannot be performed older cards.

Texture lookups could be used to get even greater speeds when the color correction is decided. This way the texture work as a look-up table, where all precalculated values are stored. Then a simple look-up is made for each pixel¹⁸.

Masking

Masking is the technique of making parts of an image transparent. It is used to combine images or videos with each other. It can also be used to just change the appearance of one part of the movie. For example if the sky in one image needs to be darker, a mask can be used around the edges of the sky. Then the sky can be tuned separately from the rest of the image.

It is easy to make polygon based masks on the graphics cards. It's just a matter of placing polygons on top of the polygon textured with the input video. The polygon can then be rendered to the textures alpha channel to produce a transparent layer in the frame. Movement of the mask is then easy to implement, since it is only a matter of moving, scaling and rotating the polygons. There is plenty of support for this in the graphics card.

¹⁷ Depending on the standard, the values can differ. The example used the standard CCIR 601, whereas CCIR 709 would give 21.3% Red + 71,5% Green + 7,21% Blue.

¹⁸ High dynamic range images can make these tables/textures unacceptably large.

Color keying

Color keying is actually a kind of masking technology, but it takes a color as input. That color is replaced by a mask that is made transparent, so that a custom background can be fitted to the video. Often special backgrounds with a specific color are used as backgrounds. These are usually blue or green and are called blue-screen and green-screen. When the mask has been created using the background's color as a key, any background can be used in that frame. This is often a static painted background or a 3D-generated one.

Color keying is often seen as an area of its own, as it is commonly used and is an important feature in most compositing systems. There are companies that specialize in delivering color keying solutions to the market, like the Swedish company Ultimatte.

It can be difficult to do good color keying, since edges are never razor sharp. An edge between a black and white area, contains pixels that are grey¹⁹. This makes it hard to just take a color and replace it, since then all the averaged colors will not be changed. There are some techniques to get around this problem. For example, you can remove some of the chosen color from the entire image. This can solve the problem on the edges, but it changes the color of the image.

Color keying in graphics card will be implemented in a similar way to the processor. There are several techniques for doing color keying, but the simplest just involves one conditional assignment²⁰. This is easily done on newer graphics cards at high speed.

Compositing

Compositing is the art of combining several images and movie clips to one final clip. A good example of compositing is the movie Titanic, where some of the scenes were composed of up to a hundred different elements. Compositing uses color keying, masks and color correction to give believable end results.

Doing compositing on the graphics card is very nice, since it features Z-buffers, alpha channel and a large color space (16 or 32 bit/channel). This means that if you want one part of an image to dissolve over another, it's only a matter of placing the second image behind the first and to change the alpha channel of the first one. In this way one can use alpha channel textures to make the dissolves even more complex. The 32-bit floating point alpha channel assures that the compositing can be controlled in every way. The loss of information is also minimized, since the information overhead is large enough.

Transitions

Instead of just having cuts between scenes, transitions can be added. The fundamental transitions are fade-in, fade-out, fade-to and several wipe-effects. When a movie starts, it is common to have fade-in from black to

¹⁹ This is called anti-aliasing. [1] gives more information on this problem.

²⁰ [1] p 84. A chapter is dedicated to the extraction of masks (matte).

the movie. And fading between scenes can change the atmosphere of the clip.

Those are not technically difficult to implement, but have to be present in an editing program. Again, the graphics card is very useful, since fading transitions can be controlled by letting the frames be on polygons separated in z-value (depth), and then using the alpha channel to do the transition. Wipe-effects can be performed in any direction by just moving the polygons.

Painting

Painting on a video frame is mostly done using a cloning tool to remove dust and scratches. A cloning tool takes the pixel from a selected area and paints another area. Another possible use for painting is for example painting laser swords, like in the Star Wars movies.

Painting is not as easy to implement as the other effects using a graphics card. One way could be to place polygons with the desired paint stroke texture successively after each other, and render them to one layer (polygon) when the mouse button is released. This could also be a useful way of creating alpha channels that are used when masking and/or compositing as described above.

3D-effects

One obvious possibility of using the graphics card for editing purposes is to use the card for various 3D-effects. These could for example be raindrops falling on the lens, 3D shattering transitions, or distortion of the image. It could be very useful in doing compositing work directly in a 3D-program, for example when characters are interacting with 3D models.

Limitations

The effects that could be hard to implement using programmable graphics cards are probably some painting effects, for example cloning. Most other tasks seems like they are easily rewritten for pixel shaders, and it seems like most will benefit from using graphics cards. This is something that needs to be implemented to make it certain. Other effects that could be hard to create in a graphics card, or where the benefits at least are not obvious, are effects like motion-tracking²¹.

In some of these high-end applications where motion-tracking is a big part of the effect creation will probably continue to be expensive, even though modern graphics card will be able to handle this, since it is usually the software that makes the usage expensive.

²¹ Motion-tracking is used to follow the motion of a certain object between frames (in time).

4. High resolution challenges

There are several problems that have to be solved to be able to use the graphics card for doing video effects, but most of them are related to the amount of data needed for high resolution and high dynamic range movies.

The amount of data needed to be transferred uncompressed depends on the resolution and the color depth of the images being transferred. I have therefore calculated the required data transfer rates (uncompressed) in the following table²²:

Format	Color depth	Frame rate	Data required (per sec)
480p	8 bit	25 /s	$704*480*8*3*25 = 203 \text{ Mbit} = 25 \text{ Mb}$
480p	10 bit	25 /s	$704*480*10*3*25 = 253 \text{ Mbit} = 32 \text{ Mb}$
480p	12 bit	25 /s	$704*480*12*3*25 = 304 \text{ Mbit} = 38 \text{ Mb}$
480p	16 bit	25 /s	$704*480*16*3*25 = 406 \text{ Mbit} = 51 \text{ Mb}$
720p	8 bit	25 /s	$1280*720*8*3*25 = 553 \text{ Mbit} = 69 \text{ Mb}$
720p	10 bit	25 /s	$1280*720*10*3*25 = 691 \text{ Mbit} = 86 \text{ Mb}$
720p	12 bit	25 /s	$1280*720*12*3*25 = 829 \text{ Mbit} = 104 \text{ Mb}$
720p	16 bit	25 /s	$1280*720*16*3*25 = 1111 \text{ Mbit} = 138 \text{ Mb}$
1080p	8 bit	25 /s	$1920*1080*8*3*25 = 1244 \text{ Mbit} = 156 \text{ Mb}$
1080p	10 bit	25 /s	$1920*1080*10*3*25 = 1555 \text{ Mbit} = 194 \text{ Mb}$
1080p	12 bit	25 /s	$1920*1080*12*3*25 = 1866 \text{ Mbit} = 233 \text{ Mb}$
1080p	16 bit	25 /s	$1920*1080*16*3*25 = 2488 \text{ Mbit} = 311 \text{ Mb}$

The frame rate is 30 frames per second in NTSC (or 29.997) countries and 25 frames per second in PAL countries²³.

Compared to DV (which is compressed 1:5), with a required throughput

²² The 'p' after the digit means progressive scan. Interlaced video is denoted with an 'i'.

²³ The data required is the same when interlaced video is used, since it has half the (vertical) resolution but twice as many images per second.

of 3.5 Mb/s, the difference to uncompressed HD video is enormous. This huge amount of information requires very much hardware capacity.

Note that the bandwidth requirements in the table above are one-way only (only displaying it to the user). This means that if the same system should open a video stream, modify it and write it to disc, the requirements will be doubled. It also requires the hard drives to be working constantly, not considering the time that can pass without reading from or writing to disc, because the hardware is modifying the video stream.

There are lossless compressions, that will reduce the sizes on files by as much as 70 %. This is without losing any information. These compression methods reduce time to read and write to disc, but it increases the time it takes to load the images, since a decoding/encoding is needed. Some of the compression formats are faster at decoding than encoding, while some have a similar speed. The compression method used will therefore be selected given the user needs.

Disc space

The currently slowest part in a video editing system is the hard drive. This could be solved through large striped (for example RAID-0) arrays, with at least 4-8 IDE 10,000 rpm drives to get the required speed for uncompressed 12-bit data. When the amount of hard drives increases, then so does the fault probability. Thus, it would be better if the discs were in a RAID 5 array, because that would increase both speed and fault tolerance.

If the above requirements are fulfilled, it is still necessary to consider how much storage is needed to store the movie. A two hour movie in 1080p with 16 bit color depth needs $311 \times 60 \times 60 \times 2 \text{ Mb} = 2.24 \text{ terabytes}$ of data. It can quite easily be compressed without any loss for example by using Huffman encoding on the wavelet transform, thus reducing the size to 30-40% of the original data²⁴.

Transfer speeds

There are some limitations in transfer speed that can give problems. First, the data must be transferred to the computer. If one is working on data that reside on a remote server, the network must handle the data. Then, only dual gigabit Ethernet directly to a RAID-system will be possible for all formats except 1080p with 16 bit color depth. Even if the video is written to the same machine, there is a requirement that the bus (usually PCI), can handle the data. That means that it must be a 66 Mhz, 64 bit PCI-bus, which has a theoretical limitation of $66 \times 64 \text{ Mbit/s} = 4224 \text{ Mbit/s} = 528 \text{ Mb/s}$. In reality, the bandwidth will be much lower, since most systems only have one PCI-bus that should handle all the units attached.

Then the AGP-bus needs to handle all the data. This can seem like a trivial thing, since the AGP bus is much faster than the PCI bus. This is not the case, though. It seems like the drivers are very slow at moving data from the graphics card back to main memory. The reason for this is

²⁴ These numbers are acquired from OpenEXR's web: www.openexr.org

that this have never before been a priority, since that feature is not needed in 3D-games.

To test the transfer speed from the graphics card to the memory, a test called “The Serious Magic Texture Download Benchmark v1.0” by Serious Magic was used. They send the source code on request. A program by Matt Craighead that could give the performance in OpenGL was used.

The conclusions Serious Magic themselves came up with is that the cards with the current drivers are not able to perform well compared to the possible bandwidth of an 4x or 8x AGP bus. The actual transfer speeds are limited to speeds ranging from 10-15 Mb/s, according to tests by Tech Report²⁵.

3D World's reviewer of the Radeon 9700 put it this way²⁶:

“In fact, these especially low transfer rates aren't as much of a problem in Windows 98 or in OpenGL. But in Win2K/XP with Direct3D, AGP texture download rates are slow as molasses.”

Fortunately the speed have increased lately, thanks to driver updates. Results from tests will be available in section 8.

Resolution limitations

The possible resolutions are limited by the texture sizes, which on the 9700 currently is 2048x2048, but it is possible to stitch textures to get even higher resolutions. Stitching two textures halves the performance, though. When implementing 1080p, 2048x1024 sized textures can be used, since there are so few missing pixels (vertically). The textures have to be a power of two. Also there could be some problems with the color depth of the textures, for example if the color space used is not supported by the graphics hardware²⁷. Since the focus of this project is on HD resolutions, the resolution limitation will not be investigated further.

Using the graphics card

Only recently, since Microsoft released their VMR-9 (see section 5), programmers have been able to use pixel shaders when modifying a video stream. Until then, programmers had to apply the frames themselves as textures on a target polygon. Microsoft had an example program that did this in their DirectX SDK.

When developing effects for the graphics card, there are some differences compared to writing effects for the processor. There are some limitations to how many instructions can be written. There are not as many compilers. Neither are there scripting languages. It is necessary to be satisfied with the only high-level languages available, namely Cg and

²⁵ [14] p 2

²⁶ [15] <http://deltaweb73.blogspot.com/>

²⁷ Such a format could be having a luminance channel and two color channels. In that case the image needs to be transformed to a compatible format before being transferred to the graphics card.

HLSL. Both these languages are very similar to C, which makes it easier to develop effects, if the programmer knows C beforehand. Even though there are not any scripting languages for programming the graphics card yet, it will not pose a big problem to introduce it.

5. Working practice

Industry today uses expensive dedicated hardware for certain tasks. Usually techniques that are available for high-end users eventually become available to all users. The DV format is a good example of this. It is a format that is aimed at ordinary consumers, but it can still deliver the same quality as commercial ones. There is little difference between DV and DV-CAM for example²⁸.

Currently, it is common to use proxies when working with high-resolution images. A proxy is a low-resolution version of the movie that is used for editing and transitions. When the editing is done using the proxy, an EDL (Edit Decision List) is created. This is then used to create the final movie using the original video.

When using an EDL it is easy to rescan the material. This is often done when editing is finished, and is referred to as a fine-scan. The process takes a lot of time. Since the scanning machines are extremely expensive, the longer time spent on those machines, the more the final movie will cost.

When doing compositing²⁹, the computers and the programs used are expensive, and still it is common to define areas of interest (AOI), because it is too slow otherwise.

Scandvision Interview

I wanted to see how industry currently works when it is editing, compositing and color correcting in practice. Therefore I asked Scandvision, in Malmö, Sweden, if they could show me around. Mikael Kotanidis (Partner/Manager), Anders Borg (Film Editor) and Kalle Shew (3D Artist), gave me their opinions on current technologies, and what they would have liked the technologies when compositing video/color correcting to be like.

²⁸ The difference is that the speed of the tape is slightly higher with DV-CAM making it possible to have less compression.

²⁹ Compositing is when several movie clips are combined with each other or with still images to create a combination – a composite.

The software they used were mainly Smoke, Avid and After Effects, all of them with their own advantages. When suggesting features, they compared the programs with each other and came up with ideas that they thought would be nice, but which none of the programs had.

When color correcting, they thought an important feature was the ability to compare pictures/clips before and after a color correction had been made. This could either be accomplished by having the clips side-by-side or by having a divider of the picture that could be moved to show the differences in color before and after. This feature could also be accomplished by turning on and off layers like in Photoshop.

The colors should be controllable separately, like turning the green bluer for example. When color correcting, it is also common to control shadows, midtones and highlights separately. They thought it would be nice if one could have default settings for doing some rough corrections, and then have the possibility to fine-adjust manually. The colors should be adjustable per clip-basis, without having to adjust percentage sliders like in Smoke. When they were pleased with a clip, they would like to be able to save their setting so that they could apply them on different clips. Copy and paste-like behavior would be nice, too.

When controlling the colors, they wanted to be able to change the images/clips in several color spaces, like RGB, CMYK, HLS, YUV and others. A mode for changing colors like variations in Photoshop was wanted. In Photoshop it is possible to see previews of all 6 different directions (R, G, B, C, M, Y), which makes it easy to get an overview of the changes done. Changing the contrast is vital, as is the brightness. Easier controls for special purposes could be making it more “outdoor-like”, or maybe altering the perceived time of the day.

A feature related to color-correction is color-keying. They thought this feature was important, because it could be used for changing specific colors, or compositing with greenscreen or bluescreen. Color keying is a kind of masking feature. This often results in problems along the edges of the color to be substituted. If this could be done more smoothly, it would be a good thing. Masking in general is something that is used a lot, for example when compositing. It can be used to color correct certain parts of images separately. When masking they preferred Bezier-curve masking, but it would be sufficient with polygon based masking.

When working with scanned 35mm or 16mm they needed the ability of removing noise/dust/scratches on frames, either automatically or partly manually. This will probably require some sort of painting tool, like Photoshop's clone tool. Filters that are often used are sharpen/blur filters.

The best thing for Scandvision would be if all of their requirements had been incorporated into one of the programs that they used the most. But it is common in the movie production industry that several stations are used during the production of a movie/commercial. Often the editing station is not the same as the compositing station. Therefore they could consider buying a station that only does color correction, for example, if it does it well. Then they could do the final color corrections in-house instead of doing a fine scan (when using 16mm or 35mm). Another program would

need to be compatible with their current stations. This means that it must read and write their common formats, like Digibeta.

If they are modifying digital recordings, that are interlaced, then they want to de-interlace the videos first. Since the amount of color information is more limited than scanned film, it would be nice to see how much of the image information, that is the amount of different colors, that is retained when doing transforms and color manipulations.

The conclusions from this discussion are that even the expensive systems available now had flaws from the user's perspective. Things were not always rational and easy to do. For the future, they wished that they would be able to do the corrections faster, without losing the possibility of control. They wanted to be able to store color correction settings, for example. When working they wanted to see the differences of their actions, by being provided with a before-after view. They also wanted a masking functionality, so that they could control colors differently on different parts of the picture. The masks need not necessarily be of arbitrary shapes, but could be polygons or even rectangles.

The tasks that are performed today would be enhanced if the response time were faster. This means that if the graphics cards give better performance when only viewing the samples (without the need to write to a hard drive), then this would be a sufficient benefit. Thus this will be part of the investigations in this work.

6. Existing technology

The way color correction is performed today is to scan the film (rough scan), and then when the movie is edited, to scan it again (fine scan), to ensure that the colors are not lost in the conversion. If you do not have the raw material on film, then you will probably do the corrections on the available systems for editing and compositing.

Dedicated hardware

It is common to use dedicated hardware that can handle high-resolution data. These are often limited in dynamic range instead, and will not be upgradeable. There will probably continue to exist dedicated hardware accelerated for certain purposes even if the graphics cards take some or most of the market share, since the graphics card is becoming increasingly complex. Today it is similar to the processor with respect to the amount of transistors. Dedicated hardware can do a limited set of operations at extreme speeds, but they are hard to upgrade to new standards.

OpenGL's Imaging subset

OpenGL Imaging subset is an optional extension to OpenGL³⁰. It provides a collection of routines that can do 2D-computations - per pixel modifications - that are similar to the functionality of the pixel shaders. The Imaging subset consists of a pipeline of the following fixed stages:

1. Color lookup table
2. Convolution scale and bias
3. Post-convolution color lookup table
4. Color matrix scale and bias

³⁰ Actually OpenGL's Imaging Subset is not a subset, even though the name indicates it, since it does not have to be supported. Support of the subset can be determined by checking the result of `glGetString(GL_EXTENSIONS)` for the substring 'ARB_imaging'

5. Post-color matrix color lookup table
6. Histogram
7. Min- and max-algorithms

The possibilities of OpenGL's Imaging subset is limited by the given routines that enable the user to change colors (table lookups or space conversions), to filter images and to combine images (computing the sum, difference or min/max-values from each image).

In many cases, this subset is sufficient. But since it cannot be programmed (for example with conditional blocks) more complex operations cannot be performed. Not even all color conversions can be performed. Some color space conversions require more complex operations like logarithms.

The extension can only be used with OpenGL and features a limited set of operations. All vendors have not supported it either. This makes it unlikely that this will be used for doing effects on video.

DirectX 9

Microsoft's DirectX 9, and its SDK, were released after the ATI Radeon 9700 which was used in the investigation in the thesis. Until it was released only 8 pixel shader instructions could be made per pixel.

Direct3D is Microsoft's equivalence to OpenGL. It is a 3D framework that is common in games. All modern 3D cards support both OpenGL and Direct3D.

DirectShow is a part of DirectX 9. It is a framework for handling video, audio and still images. It is similar in functionality to Apple's Quicktime (described below). When using DirectShow to display a video, a "filter graph" is built using a predefined set of "filters". Depending on the format, a proper decoder filter for that format is used to read the format into an uncompressed format that is passed on to the next filter. The same thing happens to the audio. The next filter can then be a transformation filter, modifying the video, or a render filter, that displays or stores the video.

Since DirectShow is modular, it is easy to reuse filters. This means that a transformation filter created to change the contrast of the video will work almost³¹ regardless of the format being used as input. DirectShow is bundled with a lot of filters that can read many different formats like AVI, MPEG-1, MPEG-2, MPEG-4, DV and some others. It also has filters for writing video in various formats.

VMR-9 is a special kind of filter. Its name is an abbreviation for Video Mixing Renderer (for DirectX 9). This suggests that it can only be used to mix different simultaneous streams of video. That is not the case, however. Since DirectX 9, it is using the graphics card to render video. This means that for display purposes, VMR-9 can make use of pixel shaders to modify a video stream.

³¹ The support for different formats needs to be considered when creating the filter, though.

Quicktime

Quicktime, developed by Apple, is probably one of the most used frameworks for working with all kinds of video. Like DirectShow, it is a component based architecture. It supports video, audio and still images. Quicktime also supports other features, like interactivity (through scripting), Quicktime VR and streaming of content. It supports many video, image and sound formats. The Quicktime framework is available on many platforms.

When the technique of using the graphics card for displaying and manipulating video matures, Quicktime will certainly incorporate such components within its framework.

ATI's technologies

VIDEOSHADER and FULLSTREAM. use pixel shader functionality for video purposes.

- VIDEOSHADER – removes noise and de-interlaces video. It is aimed at consumer video cameras. It enhances the picture de-interlaces it, converts it from YUV to RGB and performs additional effects before storing/showing/streaming it.
- FULLSTREAM – removes the blocky artifacts which are seen in for example mpeg-video. The company Real Networks is already supporting FULLSTREAM, and others will follow. DivX and other codecs are on their way to support FULLSTREAM. The technique requires modifications to both the encoder and the decoder.

The technologies are aimed at the consumer market, giving extra features when watching downloaded video or easing the process of enhancing or encoding home videos for different purposes. But the techniques could be used professionally with some enhancements in functionality. Then the programs used for professional purposes must extend to use these features.

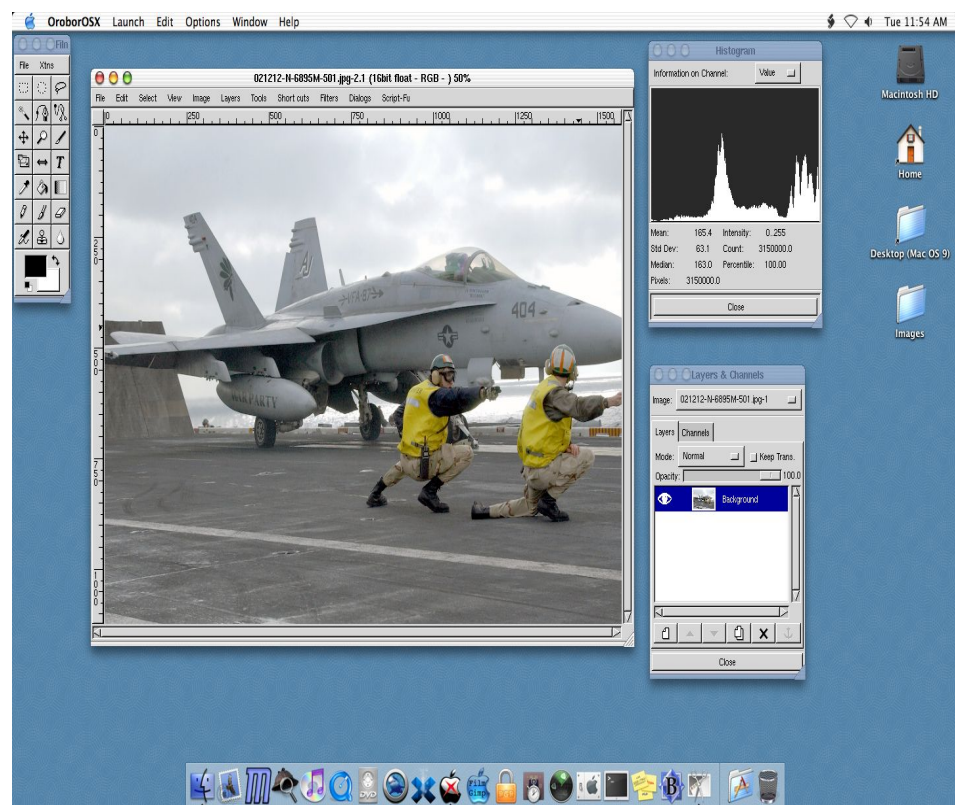
Nvidia's technologies

Nvidia has a technology named VideoFX. It is a technique for using the graphics card for various 3D-effects in real-time on video. This is similar to ATI's VIDEOSHADER. Nvidia created a demo program that shows what it can do. The program is fed with a video stream through firewire or USB2, and it is then uploaded to the graphics card. There, the video stream can be interacted with by pointing and clicking with the mouse. The program has some effects, like shattering the image or stretching and warping the image. These effects are more traditional 3D effects, which do not need pixel shaders.

CinePaint

CinePaint (formerly known as Filmgimp) is an open-source program, that is developed with high dynamic range images in mind. It has support for many high dynamic range formats, like Cineon and OpenEXR (see Appendix A). It is developed mainly by developers from the movie and

VFX industry.



A screenshot from Cinepaint on MacOSX

7. Building media tools

Why create a demo program?

A program of my own could tell how to write a program that utilizes the graphics card for enhanced performance. It could also show how the DirectShow architecture could be used to create a filter that can be used by any program. Finally, it could show how a program that changes the colors of a movie could be like.

The filter will show how the usage of the graphics card through Direct3D is implemented. Since the same filter can be implemented in several different ways, it can give a good estimate of the strengths and weaknesses of different approaches. The variables that can be changed are:

- using Direct3D or using the processor
- altering the number of effects applied
- rendering to the screen or back to the main memory

Because it gives a coherent environment, changing the parameters answer questions about how good their relative performance is. Writing a test program will also show how it compares visually with results from the processor.

Together with some test programs (described in the next section), valuable conclusions can be drawn concerning the potential of the graphics cards. The parameters that result in bottlenecks will be identified.

Building media tools using DirectShow and graphics cards

As mentioned earlier, DirectShow is a component-based architecture. It enables playback, transformation and capture of many formats. It makes it easy for example to build an image viewer or video player, without having to write support for every format.

“DirectShow divides the processing of multimedia tasks such as video playback into a set of steps known as *filters*. Filters have a number of input and output *pins* which connect them together. The generic design of the connection mechanism means that filters can be connected in many different ways to achieve different tasks, and developers can add their own effects or other filters at any stage in the graph. DirectShow *filter graphs* are widely used in video playback (in which the filters will provide steps such as file parsing, video and audio de-multiplexing, decompressing and rendering) as well as being used for video and audio recording and editing. Interactive tasks such as DVD navigation are also successfully based on DirectShow”³²

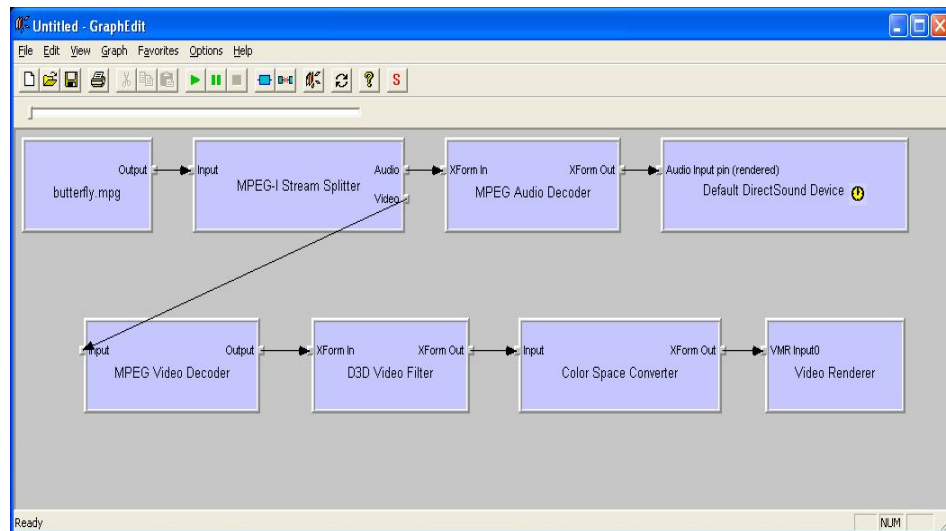
The filters mentioned in the citation are the components that makes the resulting media tool. All filters are derived from DirectShow’s IBaseFilter superclass. IBaseFilter inherits in turn from the IMediaFilter and COM’s IUnknown interface. Through the IUnknown interface, other supported interfaces can be acquired.

The filter graph is what holds all connected filters together. Through it, you can connect new filters and disconnect old. Since DirectShow is based on COM, it makes it possible to load all supported filter on the computer. The supported filters are acquired by using their name or GUID, an identification number that defines COM components. This usage of COM is about to be replaced by managed code, which means that the code is run in a virtual machine, similar to the programming language Java. Managed DirectX was released as a part of DirectX 9, but unfortunately it does not support DirectShow in this version. Managed code is usually written in either C# or Visual Basic.

The filter graph is controlled through different interfaces. Filters are created, added to the filter graph and controlled through interfaces. The filter graph is managed through the Filter Graph Manager, which implements interface IGraphBuilder. From the filter graph an interface, IMediaControl, can be acquired to control the movies passing through the filters. Using this interface, you can play, pause and stop movies for example. Another interface is the IVideoWindow interface. It that can be used to set properties on the video window. The video window can be both a window or an area in a window (in Windows programming even buttons are viewed as a sort of window).

Microsoft has created an utility called GraphEdit, that facilitates testing of compatibility between filters. Through its interface it is possible to add filters, move them around and connect them to each other. If all filters in the created filter chain are compatible, then a media file can be loaded and displayed (or stored) using the filters in the graph. If they are not compatible, the program will give an error message. GraphEdit can also assemble filters automatically, iterating through all filters and trying to build a working graph. Actually, building graphs automatically is a part of DirectShow that makes it easy to assemble graphs programmatically as well. Filters can be added, and when used to render a media file, a graph will be assembled using the added filters and adding new ones if they are needed to create a working graph.

³² [23]



An overview of a filter graph in GraphEdit

Accessing the graphics card

A program that utilizes the graphics card has to implement that functionality in some way. A filter, like VMR-9, can do this. VMR-9 is a Render filter, ending the filter chain by displaying video from one or more connections. If VMR-9 does not contain all the desired functionality, the filter has to be created from scratch. One way of accessing the graphics card and use its programmable pipeline is to use Direct3D. Since both Direct3D and DirectShow are parts of DirectX, using Direct3D together with DirectShow provide more support than OpenGL.

There are at least two things that must be done to use the graphics card. The first is that the media has to be transferred to the graphics card. The second is that the card must to use the media for the desired purpose, for example displaying a modified version of the media. Transferring the media to the card is done by applying the image to a texture and applying that texture to a polygon that covers the display area.

If there is a need for a filter that uses the graphics card, but also passes the modified media on to the next filter in the chain, it is preferably done by deriving DirectShow's Transform filter. Microsoft's VMR-9 cannot pass the data on to the next filter in a chain. It can only display it as the final filter in the filter graph.

When the graphics card is used for modifying the card, algorithms using pixel shaders must be developed using pixel shaders. This can be done in assembly language, Cg or HLSL. If the resulting media are supposed to go back to memory, the image should be rendered to a texture instead of rendering it to the frame buffer, because fetching from the frame buffer gives lower performance.

Transform filters

Transform filters are useful in a wide area of applications. They “can be used to compress and decompress data, to split audio and visual data, or

to apply effects, such as contrast or warbling, to media data³³. Transform filters modify input data and deliver them to the next filter in the filter graph. The next filter can be another transform filter or a render filter that renders the media to the screen or stores it in a file.

When developing a Transform filter, there are some things that have to be considered. First, choosing a base class. There are some different types of transform filters that support different features³⁴. To do this it must be decided if the filter should copy the data received from the previous filter, or if it can work with them directly. If the graphics card is used, the data must be copied. Therefore, the preferred filter is CTransformFilter. It must then be decided what media types the filter should support. Filters in a filter chain are always negotiating about the media type before sending the data. Finally the transform method has to be implemented. This is the method for modification of the data. Here the input data must be transferred to the graphics card as a texture. Then the pixel shaders have to be applied, modifying the frame. The rendered frame should be fetched back to memory and passed on to the next filter in the chain.

Connecting it with a graphical user interface

When designing a GUI (graphical user interface) for using DirectShow filters, a display area or window is needed (unless it is an audio-only application). A good GUI gives instant feedback about the user's behavior. Therefore, there are obvious benefits in using the graphics card for generating visual feedback on the changes performed. For this purpose the VMR-9 is very good. It eliminates the need for developing a filter from scratch. The final rendering can be done using the processor. This gives both the programmers of the application and the users control of what they want to achieve.

From the user's perspective, it is good to have one window for each clip that has been modified. This allows easy comparisons between clips. It also makes it easier to handle several clips simultaneously. One way of achieving such a user interface is to use MFC³⁵. It has one user interface model called MDI³⁶, which allows several documents to be open at the same time. Using this model, each media file will be represented by a document. It is possible to have several different views of the same document. This could be useful by having several different previews of the same image, thus giving a better overview of different changes.

DirectShow-filter

The DirectShow-filter³⁷ that was developed as part of this thesis derived from a Transform filter as described above. It opened a hidden window that rendered video frames on it using Direct3D. During the first phase of the development the example programs bundled with DirectX SDK were

³³ [22]

³⁴ Like dropping frames if the next filter in the filter graph falls behind.

³⁵ Microsoft's foundation classes

³⁶ Multiple Documents Interface

³⁷ The filter was developed together with Karl Jonsson, who used the filter to detect 3D features using stereo cameras [8].

examined. One of the example programs, Contrast, was easy to understand, and was quite similar to what was wanted. This, and another program, Texture3D, was good to have as a reference when developing the filter.

Using this filter, it is possible to change what the pixel and vertex shader do, in order to use the video card for altering the images. The filter is generally written in such a way that it can be used for all kinds of purposes. A separate document describing this general filter is available in the appendix.

The filter creates a hidden window, which each frame is drawn upon. When drawing to this window, the filter utilizes custom pixel- and vertex shaders. These shaders can be programmed to perform all kinds of operations. The primary effects tested were color-correcting effects, since they are very common, and give a good instant feedback about modifications.

When rendering the frame in Direct3D, the method that is used is `Idirect3DSurface9->EndScene(D3DX_FILTER_NONE)`. This method proved to be the bottleneck. On a forum³⁸ I was told that there existed faster ways, but there seems to be no documentation of such a way.

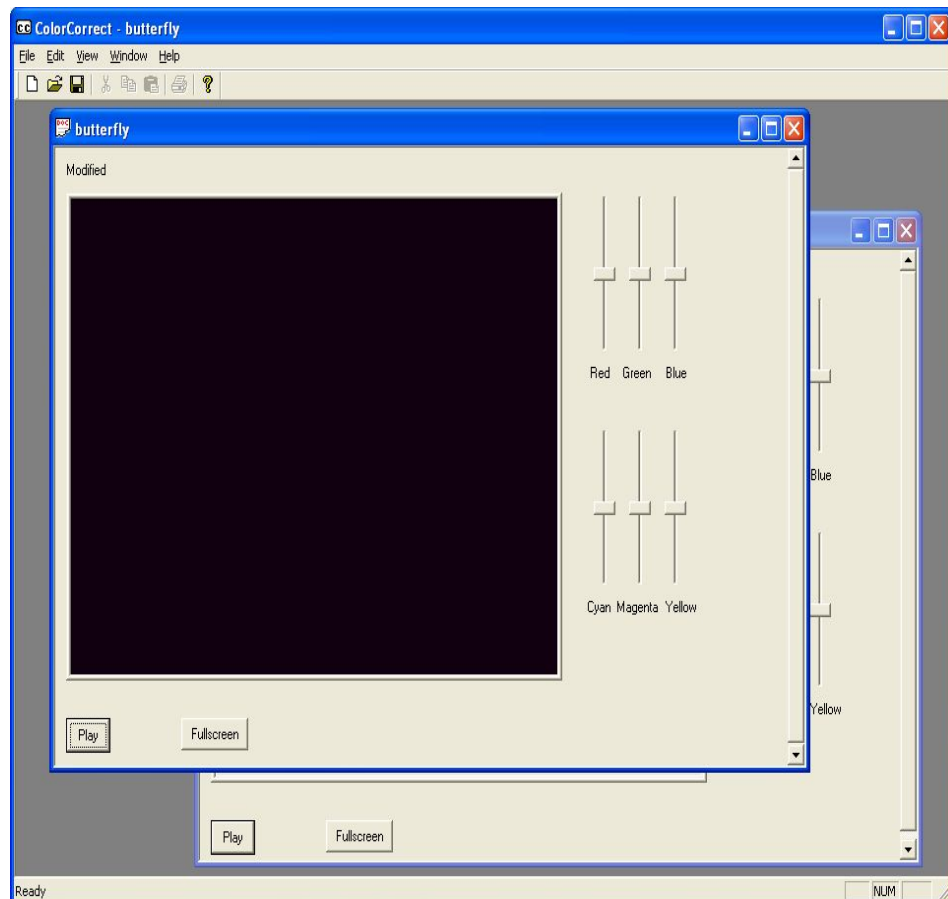
When testing the filter, the timer included in the `IBaseFilter` superclass, `IReferenceClock`, gave strange results. It showed sometimes that a subroutine took longer time than the containing method. To avoid this a switch was made to use `DXUtil_Timer`. When using this, but mixing the usage of `TIMER_GETABSOLUTETIME` and `TIMER_GETELAPSEDTIME`, the same strange behavior persisted. When `TIMER_GETABSOLUTETIME` was used in all calls to the method, the problem disappeared.

The MFC GUI

When the DirectShow filter was running, a GUI for the filter (developed using MFC) was created. At this point a switch was made to Microsoft's Visual Studio .NET. Full-screen mode was created, but a way to return from full-screen mode using the ESC-key was not found. It is possible to press the Windows key on the keyboard, but that is not a very good solution. I kept the full-screen button anyway.

The program uses the MFC multiple documents model, because a real program would be designed to be able to handle several simultaneous streams. An alternative could be to implement everything directly in Direct3D. That would have made it difficult to make it compatible with DirectShow, which was required to read all the computer's supported video formats.

³⁸ microsoft.public.directx.graphics



The graphical user interface of the program i developed

There are six sliders that one can move to change the color of the image. The six colors are: red, green, blue, cyan, magenta and yellow. These represent two different color spaces. The idea behind this is that it shows how several potentially very different color spaces could be modified from the same interface.

Testing the program

The MFC program that was developed to use custom DirectShow filter was tested in various conditions to see how well it performed. First, the time it took to transform one frame was measured. Since this time proved to be quite high (around 45 ms/frame), some tests were done to spot the most time consuming tasks made in that method. Some measures revealed that it was the Direct3D method EndScene that took almost 80% of the time.

The EndScene method alone required an average of 35 ms per frame. This time is added to the time it takes for each frame to be loaded from the hard drive, and submitted to the filter through the filter chain. In the filter, the frame is then copied to a texture surface, the scene is rendered and finally the surface is copied again and delivered to the next filter in the chain. All these steps together took around 9 ms. The time from the end of the transformation filter's transform method to the beginning of the next call to the same method was approximately 4 ms. This means that there are some optimization issues in connection with the current usage of the EndScene method. This was strange, considering that Direct3D is

very fast when rendering 3D-graphics in games. The delay could be due to the fact that the EndScene was called on a IRenderToSurface object.

When using the same filter, but changing the transform method to use only the processor, the tested color transform used a simple matrix multiplication. There are a lot more advanced color transforms, so this can be considered the simplest case (if not a lookup-table has been generated in advance). This matrix multiplication took approximately 15 ms to do on each frame. This means that it is three times as fast as the filter using the graphics card. This shows again that the Direct3D implementation was not fast enough.

8. Performance analysis

The high resolution sample that was used to test the different programs was a rendered movie at 1920x1080 and 8 bit/color. The sample had fast-moving parts in order to make it easy to spot dropped frames visually.

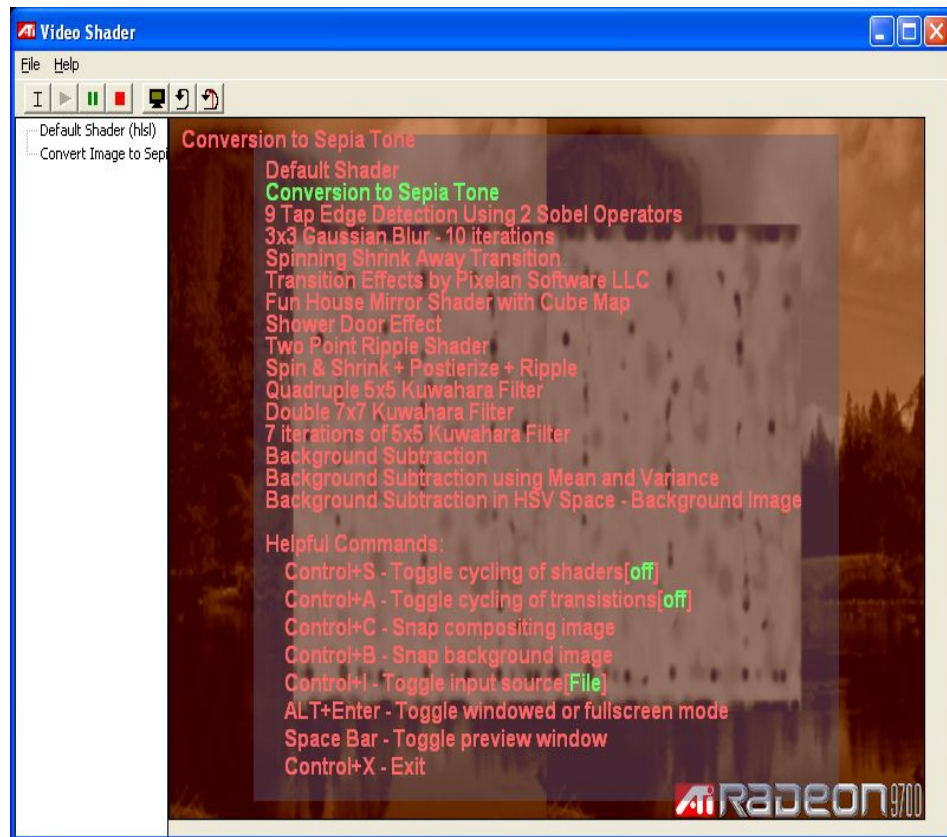
ATI's demo program

ATI has created a demo program of its own, called “9700 Video Shader 1.1”³⁹. It can open up any type of video, and render it using many predefined effects. Several effects can be run simultaneously.

What that program does not do, however, is giving the user information of its performance. It is not possible to know how long time the rendering process takes. Therefore it cannot be used to compare the performance when using the graphics card with the same algorithm run on the processor.

The first test runs, without taking the time, confirmed that ATI's program could run the sample without any visually noticeable frames dropped and with several filters applied at the same time. Actually it was even better at running the sample than Microsoft's Windows

³⁹ A free download copy of this program can be obtained at <http://www.ati.com/developers>



ATI's Video Shader 1.1 running a sepia tone color correction effect in real-time

Media Player. ATI's video shader program could handle all the filters bundled without any lag, and could have several filters in many iterations without noticing any dropped frames.

It seemed that the program had no problem at all with the high resolution required. It was so good that I suspect that it doesn't render all the pixels on the sample, since it is a lower resolution copy that is displayed in the window. But since the code is not available, and since the program does not tell if frames are dropped or delayed, it is impossible to tell.

Since ATI did not release the code for its demo program, it is difficult to tell if the results they obtained is because they only render the displayed image (at a much lower resolution than the 1080p HDTV input sample), or if it is because it is so good. Their program does not try to store the rendered video to disc again, and therefore avoids the problems of rendering to memory mentioned above.

OpenEXR's EXRDisplay

EXRDisplay⁴⁰ is a program that displays OpenEXR images (see Appendix A), and allows the user to change exposure, gamma and defogging in real time using the Nvidia's Geforce FX. The reason why only Nvidia's cards are supported is that OpenEXR's 16 bit float format is similar to Nvidia's "half" data type (see Appendix B).

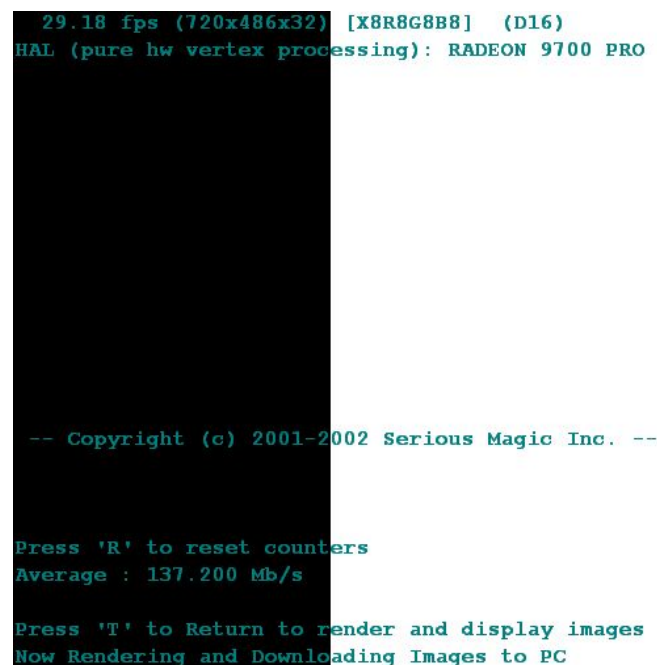
⁴⁰ This program is bundled with the OpenEXR source, available at <http://www.openexr.org>.

Render to memory

As described above, ATI has already produced a program that demonstrates the possibilities of using the graphics card for manipulating high-resolution video. But, the company has not shown that it is advantages for manipulating the content and then storing it. ATI uses Microsoft's VMR-9, which is very fast when rendering the image to the video memory. If the data are supposed to go back through the AGP port, stored in the main memory and then on the disc, some new problems arise. The main memory will need to handle the video stream twice, which could slow down things.

To figure out how the performance is affected when trying to deliver the modified frames back to main memory for storage, some testing has to be done.

Serious Magic's Texture download benchmark

A screenshot of a terminal window showing the output of a benchmark program. The text is green on a black background. It displays performance metrics for a Radeon 9700 Pro card, including FPS, resolution, and texture format. It also shows the average download speed and instructions for resetting counters and returning to the render loop.

```
29.18 fps (720x486x32) [X8R8G8B8] (D16)
HAL (pure hw vertex processing): RADEON 9700 PRO

-- Copyright (c) 2001-2002 Serious Magic Inc. --

Press 'R' to reset counters
Average : 137.200 Mb/s

Press 'T' to Return to render and display images
Now Rendering and Downloading Images to PC
```

Screenshot from Serious Magic's Video Download benchmark

Serious Magic developed a program⁴¹ to test the download speed, because the company also realized that there are many applications for retrieving the data from a graphics card. The program renders a simple texture which it tries to download to the main memory, and it shows how fast it can be done

Matt Craighead's test program

Matt Craighead is a developer at Nvidia. He developed a program to test an extension that Nvidia has made to OpenGL⁴². The extension speeds up reading and writing to the graphics card by specifying one area for reading and one area for writing in the main memory:

⁴¹ This program is given to anyone on request from Serious Magic.

⁴² The extension is named NV_PIXEL_DATA_RANGE

“This extension defines an API where an application can specify two pixel data ranges, which are analogous to vertex array ranges, except that one is for operations where the application is reading data (e.g. `glReadPixels`) and one is for operations where the application is writing data (e.g. `glDrawPixels`, `glTexSubImage2D`, etc.). Each pixel data range has a pointer to its start and a length in bytes.” [17]

The extension is only supported by Nvidia's graphics cards. Craighead's program uses the method `glReadPixels` to read the pixels from the framebuffer. The results show that it is very fast compared to DirectX's similar operations. The program can also be used to write pixels to the graphics card and copying pixels on the card. These tests can be used to compare the read and write speeds with each other.

When using the program, the parameters is used to control the tests. By passing “-read”, the program tests how fast it can fetch data from the graphics card and by passing “-draw”, it test how fast it can write data to the card. To use the extension described above, the parameter “-readpdr” is added when reading and “-writepdr” when writing. The command line used when testing was: “`pixperf -size 128 -format bgra -type ubyte -read`” (adding “-readpdr” when using the extension).

Craighead's program can test the performance in OpenGL, and can therefore reveal implementation differences between it and Direct3D. It will also show if it is an advantage to use different memory areas when reading and writing to the graphics card. Finally, it will aid to reveal fluctuations due to parameters, like the driver versions for the graphics card.

This program was used to test OpenGL's performance when downloading images from the graphics card to the main memory. It seems that it performs much better than Serious Magic's. The reason is that drivers have been updated recently, the tests showed.

Test results

As described above, ATI's demo program seemed to be very fast, although many effects were turned on at the same time. It could not be used to measure any performance, but by looking at it, the performance could be estimated. The good part of ATI's program was that it showed that it could run several effects on top of each other without penalty hits. Visually it was very good, without any visible artifacts or variance on the resulting image.

Matt Craighead's test program was run using the following command line: “`pixperf -size 128 -format bgra -type ubyte -read`”, and if Nvidia's extension was used, the command line changed to: “`pixperf -size 128 -format bgra -type ubyte -read -readpdr`” (-readpdr was added).

With 1GHz Athlon with Geforec2 GTS and 4x AGP the following results were obtained:

Reading without extension: 42.027222 Mpixels/sec

Reading with extension: 49.903904 Mpixels/sec

With 2GHz Athlon with R300 and 8x AGP the following results were obtained:

Reading without extension: 23.448397 Mpixels/sec

Reading with extension: - (unavailable)

Each pixel takes 4 bytes, therefore the transfer speeds obtained were:

Test	Radeon 9700, AGP 8x	Nvida Geforce2
128x128x32 (without extension)	93.793588 Mb/s	168,108888 Mb/s
128x128x32 (with extension)	N/A	199,615616 Mb/s

This seemed strange. How could an older card like Geforece2 be twice as fast as the new DirectX9 compatible R300? It could be because Geforce2's drivers had been updated to the latest drivers, so the extension could be tested. I did complementary tests using other Geforce2's with older drivers. Another Geforce2 equipped computer gave 30.355715 Mpixels/sec (122 Mb/s) without Nvidias extension. But even stranger were the results received with Serious Magic's program with this configuration. It showed only a rate of 3.41 Mb/s. Another machine with the same configuration resulted in a performance of more than 100 Mb/s [machine 2]. Since the drivers differed between these two machines, an update of the first machine's drivers was made. This gamve a very large performance improvement compared to the older drivers. Now it could also use Nvidia's extension to get even greater performance under OpenGL.

Card	Serious Magic's program	Matt Craighead's program
Radeon 9700 (ver 6.14.1.6292), AGP 8x	36 Mb/s	93.8 Mb/s
Radeon 9700 (ver 6.14.1.6307), AGP 8x	138 Mb/s	97.3 Mb/s
Geforce2 (integrated), Nforce [machine 1] (version 2.8.3.2)	3.41 Mb/s	122 Mb/s
Geforce2 (integrated), Nforce [machine 1] (upgraded to version 4.1.1.3)	96.2 Mb/s	127 Mb/s (143 Mb/s with ext)
Geforce2 (integrated), Nforce [machine 2] (version 4.1.1.3)	102 Mb/s	186 Mb/s (219 Mb/s with ext)
Geforce2, AGP 4x	N/A	168 Mb/s (200 Mb/s with ext.)
Geforce3 Ti 200, PII 200 Mhz (very slow proc)	101 Mb/s	0.564 Mb/s
Geforce2 laptop		146.4 Mb/s

The Radeon 9700 configuration in all tests have been the same, making it a good reference platform for evaluating the different implementations. Serious Magic's test program performed much worse than Matt Craighead's program, not even using the extension that should enhance the performance...

Some complementary tests were done to see how the fetching of data from the graphics card compares to writing data to the card. On the upgraded Radeon 9700, it could write an average of 166 Mb/s to the graphics card (using ““pixperf -size 128 -format bgra -type ubyte -draw”). When run on the updated Geforce2 (Nforce), 341 Mb/s was achieved. The low results with the 9700 was surprising, but the fact that the results

were much higher when drawing to the graphics card compared to reading from it was hardly surprising.

9. Conclusions

The potential of using the graphics card

ATI's program shows the potential of modifying video on the graphics card for display purposes. It is no coincidence that new graphics formats like OpenEXR are compatible with those specified by Cg and HLSL (see Appendix A for more details). The industry is growing more aware of the potentials, and has already begun adapting to the changes. Today, there is enough potential using the graphics cards, if the purpose is only to visualize the results of some applied effects. Several effects can be viewed simultaneously on a high-resolution video stream in real-time, giving editors and compositors the freedom of viewing their changes on the original samples while changing the parameters. This will change the need in the industry for using low-resolution proxies or areas of interest. These solutions will be much cheaper than using special hardware for certain tasks, allowing more people to do complex effects on high resolution video streams and lowering the costs for existing movie production studios.

It is only a matter of time before the graphics cards can be used for applying the effects as well (and not only visualizing them). Until then, the transfer rate from the graphics card to the main memory has to increase. The tests using OpenGL and Nvidia's extension show that this could occur in the very near future, since the graphics cards manufacturers are becoming aware of new market potentials. The transfer rate varies a lot depending on the implementation, which suggests that this is a software issue. It seems like OpenGL is faster than Direct3D in moving data from the graphics card to the processor.

In general, the speed seems to have increased a lot in a couple of months, thanks to driver updates of the graphics cards. There is no hardware bottleneck since the AGP-bus should be able to handle 1Gb in each direction (using 8x AGP). There is plenty of room to send data to the graphics card and back again. The increase in performance when updating the drivers (software) confirms that this has been a software

issue.

There are already some advantages in letting compositing programs use the graphics card. Since compositing programs use layers, they can be seen as 3D-programs. The fact that the graphics cards handle an alpha channel seamlessly together with the other color channels (using the same amount of bits), make them ideal for compositing. They will be especially suitable for high dynamic range images. There are already compositing programs on the market that use the graphics card, like Apple's Shake. But these programs are currently not using the programmable pipeline for effects.

As the demo program developed for this thesis shows, it is useful to take an existing API for video (like DirectShow), and to implement transform filters or render filters that use the graphics cards. Microsoft's VMR-9 is a step forward, simplifying for developers to use the graphics card with good performance. Tools like VMR-9 are needed to make the technology available to developers.

In addition to giving faster effects on the video stream, it is also possible to create effects quite easily that would have required a lot of programming without the tools available, when using the graphics card. Examples of such effects are bump mapping, scaling, shatter transitions and flip transitions. Masking is another feature that can be done very fast with high quality, because of the presence of a high precision alpha channel.

Some effects are even easier to implement in a 3D environment than doing it on a processor. For example, it would be very efficient to scale video with high-quality, due to features like bi-linear filtering. Programming such a filter is only a matter of moving the vertexes in the polygon making up the video. There is no need to code all the 3D features that lie behind this, since they are supported by the 3D framework. Other filters that are easier to create for a graphics card are bump-mapping, outlining, 3D-filters, etc. The standard of always using an alpha channel in 3D can prove useful when compositing.

A graphics card is more flexible than dedicated optimized hardware solutions. This means that it will be possible to write new software if new features are wanted.

The future

Hopefully, new drivers will make the transfer rate from the graphics cards to the main memory speed up even more. Soon the speed will be sufficient to use them in real-time movie editing studios.

The transfer rates will continue to increase in hardware and storage, making it possible to handle the enormous amount of data required.

More programs will use programmable graphics cards, adding new 3D-features to the list of available effects. 3D programs will probably integrate more support for video, making compositing programs and 3D programs even more similar.

In the future higher level scripting language will come, which could make

it even easier to create potent video manipulation filters.

The graphics cards are cheap compared to the costs of storing the media.
This means that there still will be some cost issues in the future.

Appendix A – Formats

There is no standard for 16 bit float formats. Therefore there are several implementations of the format, all with their particular advantages and disadvantages. The only 16 bit float format I will cover here is OpenEXR's. RnH (Rythm and Hues) have a 16 bit format of their own, which in short only takes the 8 bit exponent and truncates the mantissa of the IEEE 32 bit float specification. OpenEXR's format sacrifices some range for precision.

Open-EXR

The Open-EXR format was developed by ILM(Industrial Light & Magic). In January 2003, the company made the Open-EXR an open source project, thus releasing the code to the public. Due to its flexible support for both 16 bit or 32 bit floats for each color, it features high dynamic range together with high resolution. The format can have any number of image channels, and has support for custom compression algorithms. It is supported on many platforms. Since floating point numbers have an exponent, they can be viewed as non-logarithmic, thus supporting a larger dynamic range than integer formats. Open-EXR has been used in several movies where ILM has been involved.

DPX/Cineon

The Cineon file format was developed by Kodak because the company wanted a digital format that functioned more like the negatives of real film, which have a logarithmic sensitivity to light. This means that very bright areas still differ slightly, instead of being completely white. This can be seen in many video recordings, and the only thing that can be done to correct this is to put a large amount of time in lighting the scene.

By using the logarithmic format, Cineon can reproduce approximately the same amount of color information with 10 bit resolution as it would take using 14 bits in linear color space. Cineon is a very common format; probably the most common format when scanning film.

TIFF

TIFF is probably the most commonly used loss-less format in the world (for still images). It is a standard that is mostly used as an 8-bit format, but it has 16 bit support. Nevertheless, it isn't very popular in the movie industry. Both the 8-bit and 16-bit format are integer formats, and they support some lossless compression algorithms.

DV

DV-support is probably the most wide-spread format, because all digital consumer cameras use this format. Even many cameras for the professional market uses this format. The DV format uses approximately 3.5 Mb/s, and is compressed to about 1:5. Most of the compression is on the color information, since humans are more sensitive to changes in luminance than in color. The DV compressions are usually denoted by either 4:2:0 or 4:1:1, which tell where the colors are sampled. It is easy to support the DV format since there already exists plenty of support for it in DirectShow and Quicktime.

Digi-beta

Digi-beta is the format which is most widely used as the broadcasting format in the television industry. It is very similar to DV, except that it is less compressed and has twice the color information (4:2:2). The Digi-beta format is not public, which makes it difficult to support.

HDTV

HDTV is not a single resolution. It specifies several different resolutions and properties of the film in which the largest resolution is 1920x1080. When you want to specify which resolution, you just specify the vertical resolution (the number of lines), like 1080. After the resolution you append a letter which describes if the frames are interlaced (i) or progressive (p), e.g. 1080p. The different modes are 1080p, 1080i (1920x1080), 720p, 720i (1280x720) and 480p, 480i (704x480). The HDTV support color depth up to 10 bits/color.

Appendix B – Graphics cards

Older cards

Before the current generation of cards (that support DirectX 9), ATI held the lead in shader support. ATI's 8500 series supported pixel shader 1.4. Nvidias Geforce (3 and later) cards supported only pixel shader 1.1 until GeforceFX was released. Pixel shader 1.x was restricted to a maximum of 8 color instructions.

Nvidia's Geforce FX

Nvidia's graphics card supports up to 32 bit floating point numbers. Nvidia also has a half datatype that is very similar to OpenEXR's half (see Appendix A). This means that OpenEXR images encoded with its half value will easily be transferred to the graphics card and modified using pixel shaders.

ATI's R300

ATI thought it would be sufficient with 16 bit floating point numbers. Their 16 bit implementation differs a little bit from Nvidia's, resulting in some compability problems. The R300 has eight pixel rendering pipelines that make it possible to do 64 to 160 instructions per pixel in one single pass.

Bibliography

Books:

- [1] Ron Brinkman, “The art and science of digital compositing”, 1999
- [2] Jeremy Birn, “Digital Lighting & Rendering”, 2000
- [3] James D. Murray and William vanRyper, “Graphics File Formats”, 2nd edition, 1996
- [4] Dale Rogerson, “Inside COM”, 1997
- [5] Mason Woo, Jackieneider, Tom Davis and Dave Shreiner, “OpenGL Programming Guide”, 2000

Papers:

- [6] Marwan Y. Ansari, “Video Image Processing Using Shaders”, ATI Research, 2003
- [7] “SMARTSHADER™ TECHNOLOGY WHITE PAPER”, ATI, 2001
- [8] Karl Jonsson, “[NAMN]”, Thesis Report, Department of Computer Science, University of Lund, Forthcoming 2003

Internet:

- [9] <http://www.ati.com/developer/index.html>
- [10] <http://developer.nvidia.com/>
- [11] <http://www.openexr.com>
- [12] <http://msdn.microsoft.com/>
- [13] Color Conversion Algorithms
http://www.cs.rit.edu/~ncs/color/t_convert.html
- [14] A problem with cinematic rendering on a VPU

<http://www.tech-report.com/etc/2002q3/agp-download/index.x>

[15] ATI's Radeon 9700 Pro graphics card, September 19th, 2002

<http://deltaweb73.blogspot.com/>

[16] OEEPE Workshop on Electronic image sensors vs film

http://phot.epfl.ch/workshop/wks99/2_1.htm

[17] NV_PIXEL_DATA_RANGE

http://oss.sgi.com/projects/ogl-sample/registry/NV/pixel_data_range.txt

[14] HD and Quantel - A White paper

<http://www.broadcastpapers.com/hdtv/Hdwhitep.doc>

[19] A Guided tour of different color formats

http://www.inforamp.net/~poynton/PDFs/Guided_tour.pdf

[20] A Color FAQ (general questions)

<http://www.inforamp.net/~poynton/ColorFAQ.html>.

<http://www.ece.purdue.edu/~bouman/info/ColorFAQ.pdf>

[21] HDTV Television - An Introduction, EE 498, Professor Keln J. Kuhn

<http://www.ee.washington.edu/conselec/CE/kuhn/hdtv/95x5.htm>

[22] Creating a Transform filter

http://www.microsoft.com/Developer/PRODINFO/directx/dxm/help/ds/filtdev/Creating_Transform_Filter.htm

[23] <http://www.gdcl.co.uk/dshow.htm>