Eurographics Italian Chapter Conference (2007)
Raffaele De Amicis and Giuseppe Conti (Editors)

Realtime Wavelet Video Encoding with Generic Graphics Processing Unit

C. Stocklöw[1] and S. Noll[2]

[1]TU Darmstadt, Germany
[2]GraphiTech, Italy

Abstract

Wavelet video encoding with multi-resolution analysis is the base for a layered coding scheme. From one video source streams of different resolutions can be generated in one coding process. To reduce computing time the graphics processing unit (GPU) of a PC system is used for tasks of video compression. Color space conversion and wavelet transformation with just one single rendering pass for horizontal and for vertical decomposition were done be the GPU. Together with the CPU the GPU increases the computed frames per second and possible resolutions for live videos.

Live broadcasting and interactive systems with streaming video like video conferencing are applications which take advantage from the proposed concept. This is especially interesting for live video encoding up to the high definition TV (HDTV) formats 720p and better.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture-Parallel processing I.4.2 [Image Processing and Computer Vision]: Compression (Coding)

1. Introduction

Transmission of multimedia content over internet became increasingly important in the last few years. Applications like videoconferencing and internet broadcast and the need for high quality video make high demands on computational performance.

On the other hand we have a huge amount of data that has to be transmitted over the network of a heterogeneous environment where every participant has a different bandwidth. Instead of coding a video multiple times with different resolutions a layered coding scheme may be used to provide every participant with the optimal video resolution according to his network and computational capabilities.

The multi-resolution analysis of wavelet decomposition provides a solution for this use case. The video input frame is transformed into different subbands, dividing the frame in a coarse approximation part and a part containing finer details. If applied multiple times to the approximation we achieve different levels of details continuously getting a better resolution.

Currently a blocked discrete cosinus-transformation (DCT) is mostly used to encode pictures and video for example in JPEG or MPEG. But the wavelet compression becomes increasingly significant in this area and was already used as the successor of JPEG in JPEG2000 using the well-known biorthogonal Cohen-Daubechies-Feauveau CDF 9/7-tap wavelet [CDF92] for lossy compression.

As for videoconferencing the timing is of particular importance. The standard ITU-T G.114 defines a maximum delay of 300 ms from capturing at one participant until playback at the other participant. Considering that additional time is needed to process these data and sending them over the network encoding and decoding has to be very fast.

With the need for high quality coding different architectures and extensions have been evolved like Intel's Multimedia Extensions (MME) and Streaming SIMD Extension (SSE). While these extensions can improve coding efficiency significantly the CPU may still be overloaded if other tasks like collaboration tools have to be performed simultaneously.

On the other hand modern consumer PCs are typically equipped with a high level graphics processing unit (GPU)

which is optimized for maximum data throughput and processing of huge amounts of data. Furthermore, computational power and efficiency of GPU grows much faster than the well-known Moore's Law for CPU [SpGL*05]. With the appearance of programmable GPUs the way was smoothed for an efficient way to use the GPU for other purposes than its original intention. This is called general purpose graphics processing unit (GPGPU).

This paper is organized as follows: section 2 describes the motivation for this work. Section 3 reviews related work. Section 4 gives a brief overview of the graphics processing unit and optimization techniques. In section 5 we describe our Solution for wavelet based decomposition on GPU. Section 6 shows the results and section 7 gives a conclusion and future work.

2. Motivation

Multi-Party video conference systems could be implemented with different architectures. For larger conferences a multimedia transfer unit (MTU) or advanced network protocols like multicast or application layer multicast / data forwarding might be useful. For conferences with small groups one client could send its streaming data (voice, video, data) directly to all other participants. This technique has the disadvantage of a higher network load. But if the used networks have enough bandwidth, multiple direct streams have also advantages: minimum delay times between two clients and be able to send to one other client a specific quality and format.

The video conference system communitrust is using the described full mesh method for small group multi-party conferences with up to 6 participants.

The wavelet encoder will be integrated in the communitrust system to reduce the network load and to reduce the used computer resources in multi-party conferences.

In a scenario with 5 participants every conference client has to sent out 4 video streams to the 4 other sites. Also every client receives 4 streams of videos from the other participants. Typically in a meeting every member would in a specific moment just look at one other person or observe two other persons (which i.e. are talking to each other). Therefore the most reasonable user interface for one client would show one or two high quality and high resolution videos and would show the other 2-3 videos in lower quality and in smaller windows. If we take the situation with 2 observed participants in high quality and 2 other participants viewed with lower quality it leads to the following configuration:

In our example this client receives and decodes:

- Two high quality video streams with 700 kbps
- Two low quality video streams with 100 kbps

The incoming streamed data are reduced from 4*700 = 2.8 Mbps to 2*700+2*100 = 1.6 Mbps = 57 % of 2.8 Mbps. Additionally each client encodes one high quality video via the wavelet encoder which generates a low and a high quality output stream, which are send out 1-3 times over the network.

3. Related work

Much research has been done in the area of GPGPU.

[HE99] described 3D convolution for volume rendering using a special Extension of OpenGL which is not supported by all GPUs. In this case the convolution is emulated in software. [MA03] reported a method to perform the fast Fourier transformation on GPU.

[SpGL*05] performed a profiling of DCT-based video decoding. They found that the two parts with the most computational complexity was color space conversion and motion compensation. To make use of GPU and to achieve a simple load balancing between GPU and CPU they moved these two transformations to GPU. [FSLC05] implemented and compared multiple techniques to perform DCT and inverse DCT on GPU.

[GS05] used tileboarding for GPU-based 3D wavelet reconstruction. [DCH05] presented a technique to store multidimensional datasets in graphics memory bypassing the restriction of most graphic cards to handle only textures with 1, 2 or 3 dimensions by using an adapted version of the Wavelet Coefficient Tree [LF97].

In [HE00] an implementation for hardware-based decomposition and reconstruction of wavelets was presented using special OpenGL features for convolution.

[WLHW] realized wavelet transformation on GPU. Their implementation is close to ours but was intended to improve coding efficiency for single pictures instead of video. It was integrated in the reference software for JPEG2000, JasPer [AK00].

Previous work has shown that wavelet transformation can indeed be performed on GPU and can outperform a software-based solution. While DCT was studied in the context of video encoding, less work has been done to perform a realtime GPU-based wavelet transformation. Another important aspect is the use of specialized functionalities used by these authors as these functionalities are available only in a few graphic cards.

4. Graphics processing unit

The need for accelerating 3D rendering - for example in the computer games industry - has led to high-performance parallel graphics hardware. Most consumer-level computers are equipped with such a device. With the introduction of user-programmable parts in the pixel pipeline, the GPU can be used for more than 3D rendering.

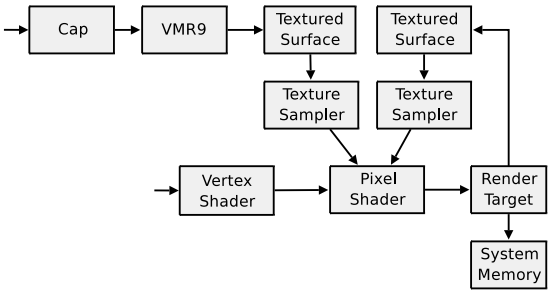


Figure 1: *Simplified architecture of GPU and the integration of DirectShow filter graph.*

Figure 1 shows a simplified version of the architecture of the GPU and the integration of Microsoft's DirectShow-interface. The first step in the pixel pipeline is geometry processing handling geometric data of rendering primitives for 3D transformations. As we only have two-dimensional data this part is essentially bypassed by using identity-transformations. The primitives are then projected to 2D and rasterized to a rendering target. At this point the pixel shader is executed for every pixel independently. It gets some parameters like the current position and a small number of textures and calculates the color for this particular pixel. The textured surface is basically a memory holding image data to be used as a texture while the texture sampler allows access to this surface.

A significant feature is the possibility to use the rendered data as input of another rendering pass. In this case the rendering target - which is typically a surface - can be used as a texture for the next pass.

Note that in this work the word pixel can also stand for preprocessed data or additional parameters, for example the wavelet coefficients.

Along with the complexity of the GPU the APIs have evolved to support new features of graphics hardware. The two main APIs are DirectX and OpenGL and their high-level shader languages HLSL and GLSL respectively. For our implementation we decided to use DirectX as this allows the easy integration of DirectShow to get video data from a file or from some sort of capture device. The video mixing renderer 9 (VMR9) directly streams this video into a surface which can be processed by the pixel shader.

Although the advantages of GPU calculations are obvious there are also some constraints:

- The number of instructions is limited. As shaders were originally designed for 3D data there was no need for complex calculations. Pixel shader version 1.1 only supports 8 arithmetic instruction slots. This constraint has decreasing importance as this number is growing rapidly. The actual pixel shader version 3.0 supports a minimum of 512 instruction slots.
- Limited memory bandwidth between CPU and GPU. Transfer of huge amounts of data between main memory and graphic card memory should be avoided. Due to the asymmetric nature especially read-backs to main memory can significantly slow down computation [THO02].
- Format of surfaces. While modern graphic cards support a variety of different formats for surfaces, older hardware may not implement them.

For our implementation we only use standard functionalities of DirectX 9 and pixel shader 2.0. All shaders are implemented using the high-level shader language HLSL.

4.1. GPU optimization

Some techniques may be applied to improve performance and to overcome some constraints of the GPU like the limited number of instructions of the pixel shader.

1. Number of rendering passes. All Pixels are processed independently by the GPU allowing for highly parallel computation. For the pixel shader this means that results from other pixels can't be accessed directly. To get access to these results, one must set the output of one rendering pass as the input of another rendering pass what can take some time.
2. Pixel packing. Textures and rendering targets support a variety of different formats. To decrease texture reads some values can be packed into different channels of a pixel.
3. Lookup texture. The pixel shader allows for multiple textures as input what can be used to provide the shader with precalculated values. Newer shader versions even allow random access to texture contents via dependant texture read. This is especially useful for video as some values can be preprocessed and doesn't have to be computed for every pixel of a frame resulting in a higher performance and a smaller number of the restricted number of instructions. On the other hand, additional texture lookups have to be made to retrieve these data.

5. GPU-based wavelet decomposition

The wavelet decomposition transforms a signal in a low-frequency (L values) and a high-frequency (H values) part. For two-dimensional data the wavelet transform has to be performed two times, in horizontal and in vertical direction because 2D-wavelet transformation is separable (see figure 2). The original data is then decomposed into 4 subbands.

The overall reason for this is the decorrelation of neighboring values. In pictures the neighboring pixel are often similar. With the decorrelation we get a part with average values and a part with details containing the differences. These details, after quantization, often are zero or close to zero and thus are a good candidate for further processing. In

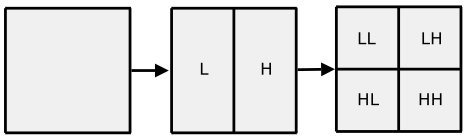


Figure 2: *Wavelet decomposition into 4 subbands. First a horizontal and then a vertical transformation is applied.*

image coding often a run-length encoding and entropy encoding is applied to reduce the amount of data significantly.

This process can be repeated for the L values resulting in different level of details making this transformation especially interesting for layered coding.

5.1. Color space conversion

The first step of video compression is mostly color space conversion. As the human visual system is more sensitive to brightness than to color, the initial data can be reduced in size by coding the color part with less resolution. We used 4:2:0 chroma sub sampling to map data from RGB to YCrCb. This scheme is also used in the MPEG standard. In the following steps the different channels are processed separately.

5.2. Convolution vs. lifting scheme

[Swe95] proposed a method to increase wavelet transformation by applying a lifting scheme thus having a computational cost of only 14 compared to 23 of the standard convolution resulting in a speedup of 64%. This scheme works in 4 steps using the intermediate results of the previous step. To apply this scheme every step has to be realized by a separate rendering pass to get access to the intermediate values. As multiple rendering passes require additional time we decided to use the standard convolution.

5.3. Preprocessing

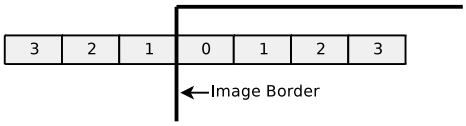
Typical video has a frame rate of about 25 frames per second. To achieve realtime encoding of high quality video as much computation as possible should be done only once by making a preprocessing step.

A technique for higher GPU efficiency is the use of a precalculated lookup table with dependent texture read which is shown in figure 4.

5.4. Boundary extension

Close to the boundaries the signal has to be extended because it is of finite length. A few methods have been proposed to face this problem like zeroing, mirroring and replication [Dau92] in order to reduce artifacts produced by boundary handling. This problem increases with the number of levels of decomposition.

As proposed in [WLHW] we used a texture as a lookup table to save the positions of coefficients to use. We also used mirroring to symmetrically mirror pixels across the boundaries not repeating the border sample:



Note that some boundary extension schemes like [KZT02] can not be applied since we only save positions of coefficients.

5.5. Wavelet transformation

The general principle of wavelet transformation is depicted in figure 3 for the horizontal direction and has to be performed for every pixel. The vertical direction works accordingly.

This method uses only one rendering pass for horizontal and one for vertical decomposition.

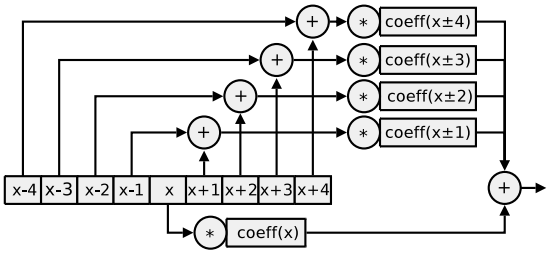


Figure 3: *Principle of 1D-wavelet decomposition: for every pixel at position x, we add the samples at xśy, y=1..4 and multiply it with the wavelet coefficient. Then the processed sample at position x is added.*

For every input pixel there are four pixel in the lookup table containing all the data needed for the pixel shader to perform the wavelet convolution. The format of the lookup table is given in figure 4. The first 8 values give the positions of 8 samples to be used for CDF97 wavelet filter kernel, followed by the wavelet coefficients. Due to the symmetric nature of wavelet coefficients the samples are first added and then multiplied by the coefficients (see figure 3). The last two values contain position and coefficient for the last sample.

For the high frequency part with only 7 values, position and wavelet coefficient for $\pm$ 4 is set to zero allowing the same method to be used for high and low frequency subband.

As the picture is decomposed into a low and a high frequency subband (see figure 2) the two halves of the picture

	1	2	3	4
R	Pos(x-4)	Pos(x+4)	Coeff(x±4)	Pos(x)
G	Pos(x-3)	Pos(x+3)	Coeff(x±3)	Coeff(x)
B	Pos(x-2)	Pos(x+2)	Coeff(x±2)	
A	Pos(x-1)	Pos(x+1)	Coeff(x±1)	

Figure 4: *Format of the lookup table for a pixel at position x with the positions of samples and the wavelet filter kernel.*

use different filter kernels. [WLHW] proposed a method to save an index in the lookup table where 0 stands for the use of low frequency and 1 for the use of high frequency coefficients. As this results in an additional texture for the wavelet coefficients and an additional texture lookup for every coefficient we decided to directly save them into the lookup table. Thus, for every pixel we have only 4 texture lookups in the lookup table and 9 texture lookups for the image data resulting in a total of 13 texture lookups.

6. Results

We have implemented our solution and performed performance measurements on an AMD Athlon 64 Dual 3800+ with 1 GB RAM and nVidia GeForce 6600 GT.

The measured times are given in Table 1. As expected all values grow accordingly with increasing video resolution.

	CSC	DWT	Read-Back	**Total**
256x256	0.08	1.9	0.6	**2.6**
512x512	0.3	6.7	3.6	**10.5**
768x576	0.4	12	6	**18.9**
1280x720	1.0	25.3	12.5	**39.7**
1920x1080	2.4	54.2	28.3	**86.6**

Table 1: *Time needed for color space conversion (CSC), discrete wavelet transformation (DWT) and read-back to system memory with different resolutions in milliseconds.*

Note that the time for writing the incoming video data from system memory to GPU-memory is not contained in these results because the video mixing renderer automatically takes care of this. Due to the asymmetric nature [THO02] for transferring data between system and GPU memory this value may not be as great as the read-back. Also, for high precision calculations we used float surfaces as rendering targets. While the captured imput data consists of only one byte per pixel and per channel, the data for read-back consists of four bytes per pixel and per channel.

For a video of 25 frames per second we only have 40 ms for encoding until the next frame arrives. The resolution of 1280x720 is chosen because it is the smallest possibility for HD video (720p) and the highest resolution that can be processed within the given time-frame. It is important to note that this calculation is done on GPU only while CPU is not involved giving us the possibility for a simple load balancing between GPU and CPU. After a frame is wavelet transformed by the GPU and read-back to system memory the CPU can further process this data in a pipelining manner and perform entropy encoding while the GPU is processing the next frame.

7. Conclusion & Future Work

We have shown a method to successfully perform two important steps of wavelet video encoding - color space conversion and wavelet transformation - on consumer-level GPU in realtime. Although there is still the possibility of optimization the results are very encouraging as we achieved to transform video in high-definition resolution of 720p. GPU and CPU can then work in a pipelining manor, realizing a simple load balancing, for other transformations like entropy encoding.

Future work includes Motion Compensation, comparison of different wavelet transformation methods and a better load balancing between CPU and GPU. This codec will then be integrated into the multiparty videoconferencing system communitrust.

References

[AK00] ADAMS M. D., KOSSENTINI F.: Jasper: A software-based jpeg-2000 codec implementation., 2000.

[CDF92] COHEN A., DAUBECHIES I., FEAUVEAU J.-C.: Biorthogonal bases of compactly supported wavelets. *Comm. Pure Appl. Math. 45*, 5 (1992), 485–560.

[Dau92] DAUBECHIES I.: *Ten lectures on wavelets*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[DCH05] DIVERDI S., CANDUSSI N., HÖLLERER T.: *Real-time Rendering with Wavelet-Compressed Multi-Dimensional Datasets on the GPU.* Tech. Rep. UCSB-CSD-05-05, University of California at Santa Barbara, 2005.

[FSLC05] FANG B., SHEN G., LI S., CHEN H.: Techniques for efficient dct/idct implementation on generic gpu. In *ISCAS (2)* (2005), pp. 1126–1129.

[GS05] GARCIA A., SHEN H.-W.: Gpu-based 3d wavelet reconstruction with tileboarding. *The Visual Computer 21*, 8-10 (2005), 755–763.

[HE99] HOPF M., ERTL T.: Accelerating 3D convolution using graphics hardware. In *IEEE Visualization '99* (San Francisco, 1999), Ebert D., Gross M., Hamann B., (Eds.), pp. 471–474.

[HE00] HOPF M., ERTL T.: Hardware accelerated wavelet transformations, 2000.

[KZT02] KHARITONENKO I., ZHANG X., TWELVES S.: A wavelet transform with point-symmetric extension at

tile boundaries. *IEEE Transactions on Image Processing 11*, 12 (2002), 1357–1364.

[LF97] LALONDE P., FOURNIER A.: A wavelet representation of reflectance functions. *IEEE Transactions on Visualization and Computer Graphics 3*, 4 (/1997), 329–336.

[MA03] MORELAND K., ANGEL E.: The fft on a gpu. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 112–119.

[SpGL*05] SHEN G., PING GAO G., LI S., SHUM H.-Y., ZHANG Y.-Q.: Accelerate video decoding with generic gpu. *IEEE Trans. Circuits Syst. Video Techn. 15*, 5 (2005), 685–693.

[Swe95] SWELDENS W.: The lifting scheme: A new philosophy in biorthogonal wavelet constructions. In *Wavelet Applications in Signal and Image Processing III* (1995), Laine A. F., Unser M., (Eds.), Proc. SPIE 2569, pp. 68–79.

[THO02] THOMPSON C. J., HAHN S., OSKIN M.: Using modern graphics architectures for general-purpose computing: a framework and analysis. In *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture* (Los Alamitos, CA, USA, 2002), IEEE Computer Society Press, pp. 306–317.

[WLHW] WONG T. T., LEUNG C. S., HENG P. A., WANG J.: Discrete wavelet transform on consumer-level graphics hardware. In *IEEE Transactions on Multimedia*. to appear.