Evaluating the Performance of Processing Medical Volume Data on Graphics Hardware

Matthias Raspe, Guido Lorenz, Stefan Müller

Institute for Computational Visualistics,
University of Koblenz-Landau (Campus Koblenz)
mraspe@uni-koblenz.de

Abstract. With the broad availability and increasing performance of commodity graphics processors (GPU), non-graphical applications have become an active field of research. However, leveraging the performance for advanced applications combining hardware and software implementations is more than just efficient shader programming: the data transfer is often the main limiting factor. Therefore, we will investigate in the applicability of commodity graphics hardware for medical data processing, and propose a GPU-based framework for representing computations on volume data. Also, we will show the clear performance gain of different operations compared to CPU algorithms and discuss their context. Not only the much higher performance of hardware implementations is attractive, but also the fact that the computation results can be visualized directly, i.e., without introducing an overhead and thus allowing for literally interactive applications.

1 Introduction

With the advances in computer graphics hardware during the last few years, the computational performance of such commodity hardware is able to clearly outperform modern multi-core processors. As shown recently in [1], many areas are able to benefit from the computing performance of programmable graphics processors (GPU). In medical application, however, visualization tasks are still the primary use for graphics hardware.

While many general purpose applications have been successfully addressed, image processing remains a particularly interesting field due to the architecture of graphics hardware. Being optimized especially for two-dimensional textures, processing such image data is obvious and achieves optimal performance. However, lots of image data, especially the medical field is acquired and processed volumetrically, i.e., as three-dimensional data.

This imposes several issues for applications targeting at graphics hardware. First, the available on-board memory becomes critical for large volume data produced by modern imaging systems. Also, 3D textures are not very flexible as they usually do not allow for direct write access, thus preventing their use

as render targets. However, the main problem is still the data transfer to and especially from the graphics memory, with volume data intensifying the issue due to the additional dimension. For certain applications, however, this is not as problematic as it might seem. As Langs et al. [2] have shown by using the graphics hardware for filtering large volumes representing video frames and thereby achieving a performance several orders of magnitudes faster than commercial CPU implementations. Köhn et al. [3] have implemented image registration algorithms on the GPU and also report a clear performance gain being only flawed by some graphics driver limitation.

In order to evaluate the discussion for processing (medical) volume data, we will compare different types of operations with respect to their runtime performance as CPU and GPU implementations, respectively. Therefore, our GPU-based framework "Cascada" that is already used in the context of medical segmentation (see [4]) regards computation processes at a more abstract level and handles processing and visualization tasks uniformly. The cross-platform system also provides basic functionality for handling (medical) volume data and integrates different visualization techniques and input devices.

2 Material and Methods

2.1 Overview of the GPU-based Framework

The system for (general purpose) GPU programming focuses on processing volume data, mainly from tomographic imaging systems such as CT or MRI. The algorithms are represented in a hierarchical structure, consisting of:

- *Sequences* encapsulate procedures ranging from simple thresholding operations to iterative algorithms or visualization components
- Sequences consist of multiple *passes* that resemble shader programs being executed by rendering geometry
- *Shader programs* combine the different types of shaders (currently: vertex and fragment programs) with an additional infrastructure for handling parameters, concatenation, etc.

For efficient data handling, we use basically two complementing approaches. First, the data is represented as volumes packed into RGBA-tuples, thus allowing for direct rendering into the volume and exploiting the SIMD (Single instruction, multiple data: processing four data items simultaneously) architecture of GPUs. Therefore, four successive slices of the scalar volume (or four DICOM slices) are combined into one RGBA slice. For both backward compatibility and better performance, the system additionally uses a two-dimensional representation of the volumetric data as introduced by Harris et al. [5]. As shown in [2], the time for accessing the flat-3D texture by converting the 3D texture coordinates into the 2D address is even less than direct 3D access for basic interpolation modes. All representations of the volume data are converted and loaded to/from the GPU only on demand, i.e., expensive operations are kept at a minimum.

At the level of render passes in our hierarchical representation, the system does not distinguish between processing and visualization steps. The only difference is the target of the pass/sequence (i.e., offscreen/onscreen buffer) which can be changed during runtime, if needed. Therefore, the system implicitly allows for displaying the results of algorithms while they are computed, with a negligible performance overhead on modern graphics hardware, as shown in [4]. Currently, our system provides several volume visualization modes, with standard MPR (multi-planar reformation) and direct volume rendering by means of GPU raycasting.

2.2 Performance Comparison

In order to provide some measure for the efficiency of our framework, we have compared its processing performance with MeVisLab [6], a widely used software system for efficient medical data processing and visualization. Although "Cascada" also provides software implementations of the algorithms, it is not optimized in this regard. Let alone the fact that comparing a system with itself is not meaningful and transferable at all. Most of MeVisLab's core algorithms are based on an image processing library that, in combination with the powerful frontend, allows for flexible and rapid development of applications. However, the system does not utilize programmable graphics hardware, except for visualization purposes and simple modifications of primarily visual results via shader programs.

As mentioned before, an important factor for leveraging the performance of GPUs is the amount of data transfer relative to the processing. Basically, a much better performance can be achieved by loading the data once to the graphics card's memory and compute as much as possible there. This is also preferable for the simultaneous visualization of the results during computation.

We have set up two different scenarios for comparing the systems. The first one is intended to show the performance at operations with different levels of computational complexity, ranging from simple thresholding to gradient computation in a 3D neighborhood. The second performance test (figure 1) aims at mimicking a reasonable procedure for volume processing, so the focus is on combining different operations while limiting data transfer to the extent needed.

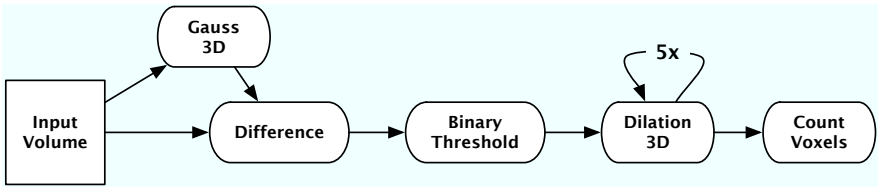

Fig. 1. Test setup for evaluating the performance of some pipeline consisting of different types of operations

Table 1. Computation times in seconds for the two data sets (CT scan/MRI scan). The GPU version excludes data transfer from/to the video memory and is averaged over multiple runs

Operation	MeVisLab	Cascada (all)	Cascada (GPU)
Binary	0.73 / 0.21	1.9 / 0.55	0.038 / 0.011
Gradient 2D	10.1 / 2.9	2.6 / 0.7	0.06 / 0.017
Gradient 3D	14.9 / 3.9	2.6 / 0.72	0.059 / 0.018
Gauss 2D	1.36 / 0.37	2.48 / 0.73	0.061 / 0.017
Gauss 3D	3.65 / 1.01	2.59 / 0.78	0.09 / 0.026

Table 2. Single computation times in seconds for the two data sets (CT scan/MRI scan). The GPU version includes the shader and all setup steps between the stages; the initial upload is omitted as it is addressed in table 1. Total time does not include loading the data for both platforms

Operation	MeVisLab	Cascada (GPU)	∅ speedup
Gauss+Difference	3.6 / 1.48	0.74 / 0.23	5.5
Binary Threshold	0.7 / 0.23	0.125 / 0.049	5.2
Dilation 3D (5x)	27.12 / 7.6	1.14 / 0.343	23
Count Voxels	1.1 / 0.7	0.33 / 0.078	6
Total	32.9 / 10.8	2.41 / 0.76	14

All the experiments have been performed on an Intel Core2Duo (2.4 GHz) with 2 GB RAM and an Nvidia Geforce 8800 GTS with 640 MB of VRAM. For MeVisLab we have used the latest SDK version (1.5.1) available from their website with default settings. We have applied the algorithms in both applications on the same datasets: a $512 \times 512 \times 223$ CT scan (DICOM files), and an MRI scan $256 \times 256 \times 256$ (raw volume), both with 16 bit values.

3 Results

In the following tables, we have gathered all the timings from the aforementioned experiments. The first table shows some details on the single operations with increasing complexity (no neighborhood, 4/6-neighborhood, and 8/26-neighborhood, respectively). In addition to their behaviour in the different dimensions, the transfer overhead is measured in detail.

The second table gives an overview of some reasonable concatenation of varying operations. No additional time for data transfer and/or conversion is measured here, as both systems work in their appropriate format for subsequent execution. We have also integrated special caching modules in MeVisLab to decouple the different steps and allow for comparable measuring.

4 Discussion

As can be seen in the preceding tables, the GPU is able to clearly outperform CPU implementations for all operations with speedups of up to 250, even for

non-trivial algorithms. In contrast, including the additional time for the data transfer to and from the GPU reveals the severe impact on the performance: depending on the type and dimensionality of the operation, and the size of the volume the software implementation can surpass the GPU's performance in this case. Aside from several optimizations that are left as future work for the GPU system to tackle this issue, reloading the whole volume in every stage is not very likely to be needed. Therefore, the second scenario has been set up to provide a more "real life" workflow, with the GPU being more than one order of magnitude ahead of CPU implementations and thus being well suited for the computationally demanding field of medical image processing.

Of course, there is still room for improvement and investigation, aside from just technical optimization. Software performance is not only measured in raw computing performance, but also in programming effort where a research system as Cascada cannot cope with established platforms. Also, graphics hardware vendors have already started to simplify the use of their hardware by regarding GPUs more as devices than purely graphics oriented hardware. However, the key advantage of using graphics hardware also for computation, namely the visualization "for free" will be harder to use then. Regarding the fast development cycles of graphics hardware, the performance advantage will most likely be increasing as can be easily seen from speedups with past hardware generations.

References

1. Owens JD, Luebke D, Govindaraju N, et al. A survey of general-purpose computation on graphics hardware. Comput Graph Forum. 2007;26(1):80–113.
2. Langs A, Biedermann M. Filtering video volumes using the graphics hardware. Lect Note Comp Sci. 2007;4522:878–87.
3. Köhn A, Drexl J, Ritter F, et al. GPU accelerated image registration in two and three dimensions. Proc BVM. 2006; p. 261–5.
4. Raspe M, Wickenhöfer R, Schmitt F. Visualisierungsgestützte 3D-Segmentierung und Quantifizierung von Bauchaortenaneurysmen. Proc CURAC. 2007; p. 197–200.
5. Harris MJ, Baxter WV, Scheuermann T, et al. Simulation of cloud dynamics on graphics hardware. Proc ACM SIGGRAPH/EUROGRAPHICS Conf Graph Hardware. 2003; p. 92–101.
6. MeVis. MeVisLab: A Development Environment for Medical Image Processing and Visualization; 2007. http://www.mevislab.de.