Real Time 3D Rendering of Optical Coherence Tomography Volumetric Data

Joachim Probst[a], Peter Koch[b], Gereon Hüttmann[a]
[a]Institut für Biomedizinische Optik, Universität zu Lübeck, 23562 Lübeck, Germany
[b]Thorlabs HL AG, 23562 Lübeck, Germany

ABSTRACT

Modern measurement equipment delivers more detailed data and faster data with each generation. These data can be used for different applications, one of them is doing real time display. Instead of saving all data during the measurements and analyze it afterwards, the data is displayed in real time and only especially selected parts of the data are saved for further work. Moving the screening part of the analysis to the human brain and pattern recognition avoids the saving of vast amounts of data and massive calculation power on computers afterwards and it dramatically improves the level of interaction with the measurement systems.

This work starts first with a short look to the question what OCT is and how data is acquired. The different possibilities of volume rendering are presented in their basic ideas. Graphics hardware and algorithms are presented and discussed. Last the results of measurements taken by the system will be presented and discussed.

Keywords: optical coherence tomography, volume rendering, real time

1. INTRODUCTION

This feasibility study covers the realization of creating a high speed optical coherence tomography (OCT) system with on-line rendering of the volume data set measured at near real-time speed, based on standard components.

Optical coherence tomography was introduced more than 15 years ago for imaging retinal structures [1]. At the beginning mainly used for imaging retinal structures in ophthalmology, but with the years also used in dermatology and other imaging fields. Since then imaging depth did only increase marginally, resolution was improved by less than a factor of ten, but imaging speed was boosted by more than three orders of magnitude, from less than 100 to more than 300,000 A-scans per second [2]. This enables not only the scan of larger tissue surfaces like esophagus [3], colon [4][5] or vessel [6], but also opens new application beyond diagnosis. A non-contact volumetric imaging with less than 15μm resolution, which is not possible by ultrasound or any other medical imaging modality, can guide microsurgery at the eye [7][8], in ENT [9] and in other medical disciplines [10]. First attempts for an intra-surgical use of OCT failed mainly because of low imaging speed [7][8]. Full use of OCT during surgical procedures can only be made with rapid 3D imaging, since otherwise it is difficult to bring the image field in coincidence with the relevant but dynamically changing anatomical structures. With the increased speed the amount of data taken by OCT measurements increased, too. With modern systems one can measure hundreds of megabyte of data within a few seconds. This arises the problem of handling this huge amount of data.

With low speed OCT systems the data was either viewed directly or could easily being saved and processed off-line after the measurements had been finished. Due to the low scanning speeds in the beginning measurements where not suited for real-time purpose, as the measurement of slices only already took several seconds and the measurement of volumes could even take minutes. The faster OCT systems than established the possibility to do real-time measurements, cross sectional images could be taken with much less problems due to motion artifacts, nowadays even volumes, especially when examining in vivo samples.

In addition to the optical part of such fast OCT systems, the post processing part got more complex, too. A lot of systems therefore only display some sort of cross sectional image to the user in real-time. Further processing or volume rendering is left for off-line handling. Therefore the complete raw data sets are saved. This way of data handling has two major drawbacks. First there is no possibility to get a processed real-time display of the data, may it be 3D rendering of a

volume to make structures visible, display of segmentations, structural tracking, etc. Such direct response would allow the use of optical coherence tomography as an on-line imaging technology during surgery or other medical treatments and open a new field of use for this technology. The second drawback of the off-line processing of the measured data, is the need of saving the data. With the fast volume scans possible this data can grow rapidly in the areas of hundreds of megabytes or even gigabytes. The possibility of on-line display of such data, would target both drawbacks. The user has direct response of his work and the amount of data to be saved would be limited to achieving purpose of interesting or important parts only. But even for the real-time display of cross sectional images the raw data taken from the sensor has to be post-processed in the same speed. As such modern and fast OCT systems are often spectral domain systems (see 2.1 for more details on OCT systems) this includes fast Fourier transformations, clipping of the data and more. Standard consumer computer hardware is near the limit of full load on just doing these tasks in real-time. This does not leave much room for further, especially further complex, tasks.

Therefore this work will examine the use of other means of doing such further processing of the measured data in real-time, the use of new generation of video cards allowing general-purpose computations on the graphics processing unit, also known as GPGPU. The concrete application is the 3D visualization of OCT scans in near real-time.

2. THEORY

2.1 Volume rendering

Volume Rendering describes the process of generating a 2D image of a 3D volumetric data set. Two major principles are commonly known for this purpose, surface extraction and direct volume rendering.

Isosurface rendering is the most common method of surface extraction. Within the volumetric data volumetric pixels (voxel) with equal properties are located. These voxels are than combined to form surfaces as polygonal meshes in the three dimensional space of the data volume. These meshes, build on vertices (points in the three dimensional space), edges, polygons and surfaces, can than be processed very efficient with modern video cards, as they just have to be handed to the hardware rendering pipeline. For the surface extraction there are various algorithms known for many years, for example the Marching Cubes algorithm. As this method creates surfaces it is very well suited for visualize sharp property borders within a volume data set. For CT and MRI this method is therefore working very well - e.g. for visualizing the bone structure or the boundaries of internal organs - and even the rendering with non-state-of-the-art video cards is possible.

For direct volume rendering a completely different approach is used. Each single voxel is mapped via some transfer function to color and transparency. Such transfer functions can range from simple ramp or windowing functions to somehow mathematically described functions or even arbitrary mappings using tables. After this mapping of the raw data the volume cube of color values has to be projected on to pixels in a 2D image presented to the user. For this step different rendering techniques are available. The two most commonly known are volume ray casting and texture mapping.

In volume ray casting the 2D image is constructed from the observers point of view. For each pixel in the image plane to be generated, one ray is cast from the observers position through the volume to be rendered in the 3D scene. This ray can than miss the objects or hit an object within the scene. The voxel hit by the ray determines the further proceeding. In the most easiest version it could just deliver back the color values to the pixel of the image, the ray was cast for, and thus determine the color of the pixel. A more complex version would be to accumulate voxel data along the ray and use this accumulated data to determine the color of the original pixel. This is especially needed, when not only solid colors are used, but partial transparency is used. In this case, the color could not only be determined by the very first voxel the ray has contact with, but voxels laying behind the first one will have an impact to the final color, too. Going this direction further on, a ray hitting a voxel could also being reflected or partly reflected by a voxel, enabling mirror effects. At this point this method crosses the line to ray tracing for generation of photo realistic images e.g. in CAD systems. This method delivers normally the most accurate and detailed images, but on the cost of relative high need of computational power. As modern video cards do not naturally implement this working principle, the standard rendering pipelines on the video cards can not be used and custom programming on CPU or GPU (as hardware accelerated volume ray casting) is needed.

In this work the third rendering technology is used, texture mapping. Texture mapping tries to overcome the drawback of the ray casting technology described above, the missing hardware support on modern video cards. Modern video cards

use vertexes, connecting them to triangles or quads forming surfaces. These surfaces can than be applied with patterns or being defined to have special material properties. The most common way of applying patterns to such surfaces is the use of textures, as some sort of extended color definition of surfaces. Texturing a surface could be compared with painting an image of a wall to a canvas and placing it into a window frame and simulating to be a real wall. As all modern video cards use hardware rendering pipelines on the basis of vertices and primitives (triangles, edges and points), this rendering technology tries to break down the process of volume rendering to these scenery components known to modern video cards.

All realizations of this rendering technique share the use of plane stacks. The planes in the stack are the surfaces to be textured and are placed within the volume to be visualized. To avoid having the user looking through in between the planes the stacks alignment is changed according to the viewpoint of the user.

Prior to the modern video cards only 2D textures could be handled by the graphic hardware. Thus for each plane an appropriate image had to be calculated from the volume data set based on their position within. This images have to be newly calculated each time the planes change their position within the volume data set, e.g. when the user changes his viewing position. In the time where the limitation of 2D textures was valid to graphic hardware the computational power of CPUs and memory was more limited as today, too. Therefore the alignment of the plane stack was limited to the objects x-, y- and z-axis. This way the images for texturing could be pre-calculated for each alignment and a given number of planes. This has two major drawbacks. First the pre-calculation was quite time consuming and second an more important: all the pre-calculated images had to be stored and could easily sum up to more image data, than the original volumetric data. The big advantage of this rendering technique is, that when once all textures and planes are stored within the video card, nearly no further interaction from the CPU is needed for rotation or zooming operations.

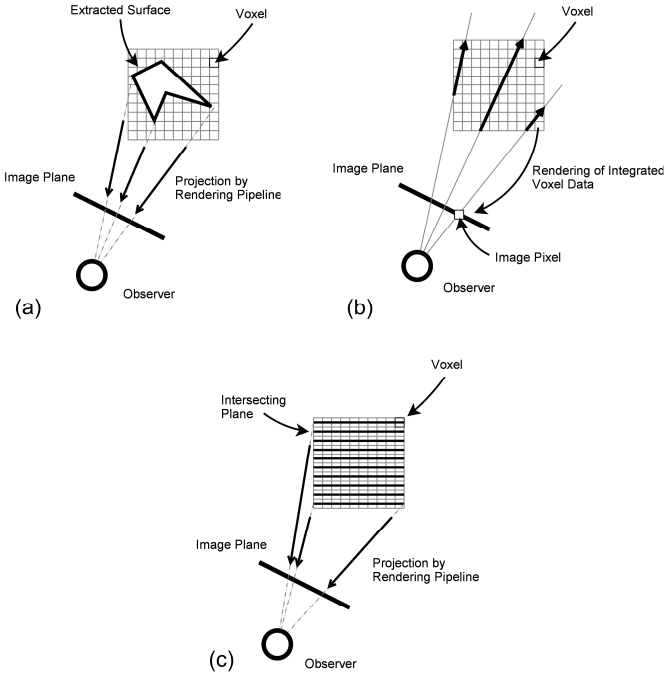

Figure 1: Two dimensional illustrations of the rendering techniques

The next step in the development of texture mapping for direct volume rendering was the support for 3D textures in the graphic hardware. This made it possible to transform the volumetric data set to one 3D texture. The calculation of the appropriate images for each plane in the plane stack could now be done on-the-fly directly in the video card hardware. This gives a massive reduction of memory needed for visualization, especially when using a high number of planes for a smooth looking result.

The use of 3D textures also made another step possible. When the images for texturing the planes surfaces could be calculated on-the-fly in the video card, than it is not necessary to limit one self to the object axis for alignment, especially when taking into account that the computational power of CPUs had also increased. The plane stack could now be aligned perpendicular to the viewing direction instead of the objects axises. As less rendering artifacts are produced, this rendering technique produces better results with less planes needed, when working on very fine detailed volumetric data. This enhancement in quality comes with the cost of more complex planes within the plane stack as the planes are now not limited to rectangular areas any more, but could take any form resulting from plane to cube intersection.

Figure 1 shows a two dimensional sketch of the different volume rendering techniques. Subfigure (a) refers to the isosurface extraction, (b) to ray casting and (c) to texture mapping.

2.2 General-purpose computation on graphics hardware

Extending the computational power of computers is long time known in computer history. In former times this was done by extension cards or accelerators. These extension cards are focused for special purposes. One example from near history are PhysX extension cards of the Ageia company from 2006. This technology was targeted at the calculation of physical effects as they became more and more used in games. The additional extension card should facilitate the CPU from the calculations of physical effects.

In the last years video cards got more and more powerful, mainly due to the increasing computational power needed for complex 3D worlds in computer games, simulations and CAD systems. With the high peak performance video cards achieve got more and more discovered to be used not only for displaying purpose of graphics, but for general purpose operations. Like the PC itself, video cards have (most times) own memory and processing units and are connected via high bandwidth bus systems with the rest of the PC. Figure 2 shows the progression of the peak performance of modern NVIDIA 3D video cards in comparison to the performance of CPUs [11].

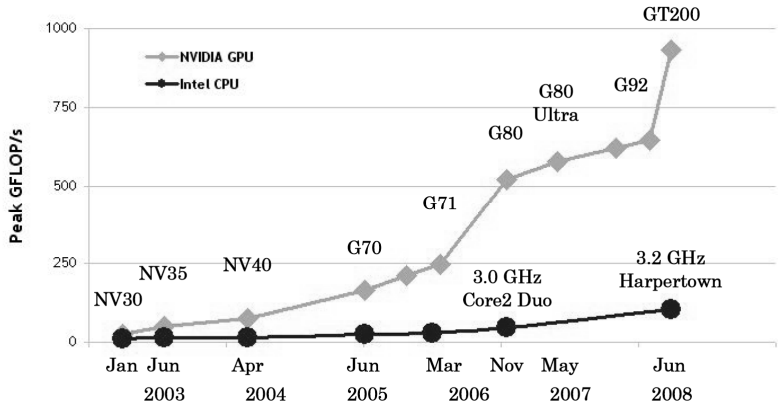

Figure 2: Comparison on the development of the computational power of video cards and CPUs [11]

These enormous computational power comes along with one major drawbacks. The devices are designed for a special purpose. Working in the intended area they are highly efficient, but not as flexible as normal CPUs are, which are - in contrast - designed to be multi-purpose devices. The second symptom of this is that the the graphic processing units (GPUs) are not easy to program, as they are special developed and do not follow the well known Intel or RISC architectures.

Around 2001 NVIDIA introduces the GeForce3 graphics processor into the market as a first programmable graphics processor. With the years Cg as a first possibility of programming the graphics processor shader devices introduced the discipline of General-Purpose Computation on Graphics Hardware (GPGPU). GPGPU introduced extensions to the DirectX and OpenGL graphical APIs (Application Interfaces) and toolkits to address the programming of the programmable shader units on modern video cards. In 2007 NVIDIA CUDA became available on the market and opened

the possibilities of GPGPU wider than ever, with making the computational power of video cards available through an extension of the C language. At this point General-Purpose Computation on Graphics Hardware started to get really into the focus for scientific programming.

CPU and GPU design is very different. While CPU design focuses on general purpose device for office work, video, music, games, etc. GPUs have a very distinct main field of work, the processing of graphical data. This work is characterized by doing one and the same instruction over and over again on a huge amount of data. GPUs could be therefore being categorized as SIMD architecture (single instruction, multiple data), where as CPUs are MIMD architectures (multiple instructions, multiple data). On multi core CPUs each core can execute different instructions on different data, optimized for a sequential flow of instructions to be executed as fast as possible. GPUs instead often feature complete sets of cores that have to execute the same instruction at the same time, but on different data. This need is very common in processing of graphic data. But this limitation is also an advantage, as it makes the control of program flow for each single core much easier, leaving space and execution time for more cores and more processing units. In Figure 3 illustrates the different amount of space dedicated to the different tasks (memory interface, cache, program flow control and arithmetic units) in CPU and GPU design.

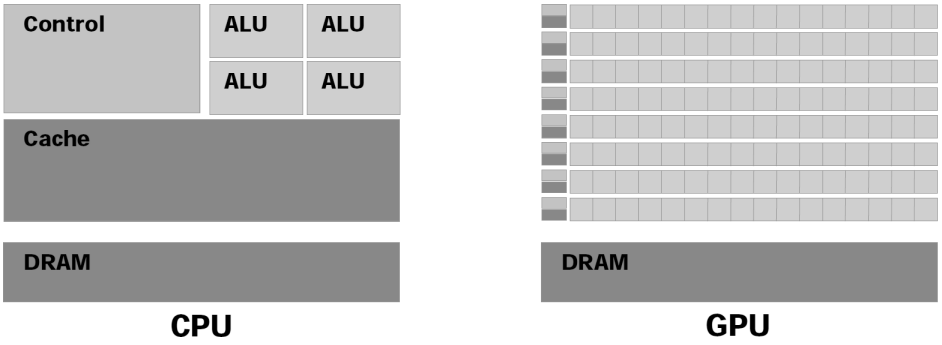

Figure 3: Illustration of space dedicated to different tasks in GPUs and CPUs [11]

3. HARDWARE ASPECTS

3.1 OCT Hardware

The OCT hardware used in this project is a Hyperion Spectral Radar OCT from Thorlabs HL Company, Lübeck.

The Basler sprint series camera, used as a detector in the Thorlabs HL Hyperion Spectral Radar OCT offers the possibility to limit the number of the sensor elements to be read out and gain higher read out speeds by this trick. By limiting the sensor to read out only 1024 sensor elements the read out speed could be accelerated to 215kHz A-scan rate at 512 data points per A-scan in change to the information given on the product sheet by Thorlabs HL.

To take the measurements an OCT-OPM adapter build by Eva Lankenau [9] was used.

3.2 PC Hardware

For the computer hardware a DELL Precission T5400 with a National Instruments NI-1429 frame grabber adapter and a NVIDIA GTX 280 video card was used. The T5400 is powered by an quad core Xeon processor at 2.6GHz and equipped with 2GB of main memory.

The PCI Express Bus was the main problem of finding a PC working with the needed expansion cards from National Instruments and NVIDIA. Both video and frame grabber card had to be connected via PCI Express (PCIe) slots on the motherboard.

Most motherboard today, especially the motherboard for consumer PCs, are equipped with one or more PCIe x16 slots and a few additional PCIe x1 slots. The PCIe x16 slots are used most times for the connection of one or multiple video cards and therefore often implemented as PEG slots. The problem at this point is that many computer and motherboard manufactures do not clearly distinguish between the different slot types or realizations of the slots defined in the PCI

Express standard. This can lead to specified PCIe x16 slots only being the to video cards limited PEG slots or being PCIe x16 slots by their physical connector but realized as PCIe x1 slots electrically. In system designed to host multiple video cards even switching the used connection speed can be changing, depending on the number of physical slots used.

The DELL PC used was found as the result of a lengthy digging for specifications and informations after experiencing the first problems getting both extension cards working together.

3.3 Video Card Hardware

For the video card there had been two different solutions possible. Either using the NVIDIA video chip set or the ATI video chip set. Video cards of both companies offer the possibility of using the hardware for general-purpose programming on the GPU, but use different tools for this. NVIDIA is using the CUDA SDK, ATI is using the ATI Stream SDK. Both are - up to now - not compatible one to each other, even so this may change in future with OpenCL starting to find its place as general purpose, platform in depended software development kit for parallel programming of heterogeneous systems on PC CPUs, GPUs, signal processors and much more architectures. But even if both companies are using different tools, both systems offer the same principle possibilities.

Due to a bigger and more active community, better documentation and a lower barrier to start programming as CUDA is just an extension to the standard C language, the choice was made to take the NVIDIA product . With the choice on NVIDIA and CUDA, the video chip set was determined to be the GT200, the actual state of the art chip by NVIDIA.

Video cards are usually produced in two lines, both using the same GPU chip, one consumer line and one professional line. Both lines are targeting to different uses. Where as the consumer line is aiming for the normal PC at home or office, doing office, Internet and PC games, the professional line aims at the use in design and construction software. Where the consumer line is optimized for the use of many and large textures as they are often used in games, the professional line is optimized for fine detailed 3D models, build of thousands and millions of vertexes and primitives.

In this work the rendering method of 3D texture mapping was chosen, with a very simple 3D model, but with possibly very big textures. Thus the consumer line with the GTX 280 model and 1 GB of video memory was chosen, fitting better into the needs of the project.

4. SOFTWARE ASPECTS

4.1 Color Coding of the Raw Data

To visualize volumetric data, represented by one dimensional data, the data has to be transformed into color and may be even transparency information for each data point. This is called color coding.

In this project the color coding, the transfer of OCT data into a texture for displaying purpose, is realized by a simple ramp or windowing function. Formula 1 gives the mathematical representation of this windowing function. All OCT data values within a defined window, a value range, are mapped linearly to color and transparency values. All OCT data values outside of this window are either rendered completely transparent or solid white.

$$opacity = \begin{cases} 0 & \forall f : f < lower \\ \dfrac{f - lower}{upper - lower} 255 & \forall f : lower \le f \le upper \\ 255 & \forall f : f > upper \end{cases} \quad (1)$$

The window position and size can be defined by the user, setting a window offset (the lower border of the range) and the window size (the size of the range). As an adjustment of the window position and size requires a recalculation of the color coding, a change will in this work only have an influence to the next processed volume and therefore require an active running scanning.

Instead of this very simple color coding transformation a more complex algorithm can be used. The linear transform can be extended to an in sections linear algorithm, arbitrary transforms e.g. of higher level order or the use of splines. Even the use of lookup tables can be realized and used to apply freely defined transformations not expressible by a mathematical representation.

To smoothen the effect of speckels in the data the algorithm used was averaging the OCT data in the actual processed B-scan with it's two neighboring B-scans. The weights are arbitrary defined as 0.25, 0.5 and 0.25.

4.2 Triple-Buffering of Texture Data

Textures in DirectX need locking in the different steps of working to avoid trying to render broken textures or erasing textures while they are used. This behavior needs to be taken into account when thinking about the handling of textures in the software. Assuming only the OCT data set and the texture for the rendering process would be used. This would lead to two problematic situations, as the rendering process should happen as often as possible, up to some reasonable maximal rate to ensure an really interactive touch and feel of the rendering, allowing to zoom and rotate the rendered result independently of the speed of new data acquisition.

One common solution to this problem is to introduce a so called back buffer. This back buffer decouples the creation of a texture, from the usage of the texture or buffer. The time consuming process of texture creation is directed in a temporally valid memory section. When the texture creation is finished, this memory is copied to the real location of the texture. As the copying process is working much fast as the creation process itself, the chance of a locking problem and the duration of this conflict is decreased. But even here there are still situations where such waiting problems do occur.

The next step of improvement is not to copy the data but to use two textures and just swap their usage for being target of the creation of a new texture or being the texture being used in the rendering process. Like with the coping back buffers, the creation and the rendering process are now completely decoupled, except at the moment where both textures should be swapped in their usage. Here a moment in time is needed, where both textures are not used anyhow. This means, that starting the creation of a new texture, has to wait for the textures to be swapped, as otherwise the just new created texture would be overwritten and the already older one in the rendering process would stay the one being rendered.

The principle of triple buffering or back buffering with two back buffers, is shown in figure 4. Here the decoupling is quite good, even in the moment of swapping textures, rendering and texture creation are decoupled. When one texture is ready for displaying purpose it is placed in between the texture for actual rendering and the creation of new textures. At the same time the rendering process is signaled that a new texture is ready for displaying purpose and the creation of the next texture is started immediately. When now the rendering process has finished one cycle, the texture actual rendered texture can be swapped with the new one and the rendering process can continue immediately. This way neither the rendering process nor the creation process of new textures has to wait for the other one to execute the swapping of textures.

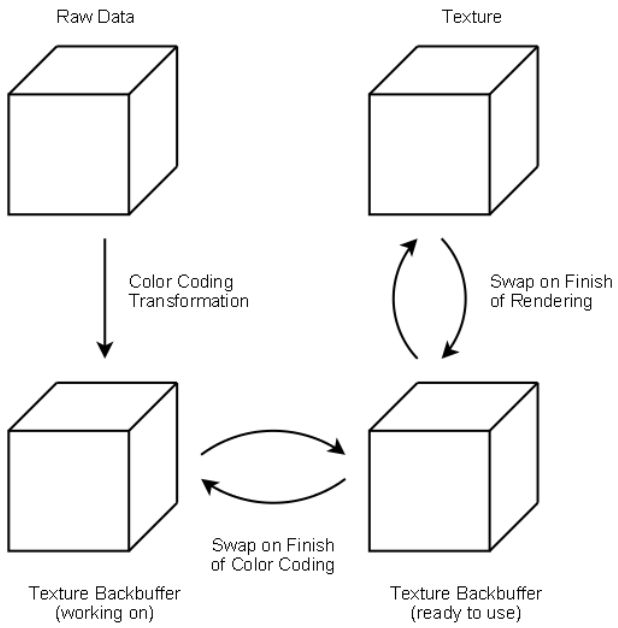

Figure 4: Illustration of triple buffering working principle.

4.3 Data Processing on the GPU

One of the biggest problems in parallel programming is to decide on the proper splitting of the problem to divide it to the different threads or tasks and gain the biggest advantage of parallel processing. The different aspects to keep in mind for this decision, are located in the algorithm, dependencies of the processing steps one to each other, thread or task control, memory management and other hardware limitations.

The algorithm used in this work is very good-natured for parallel processing. Each single voxel in the volume has to be threated the same way and independently of other voxels, especially it is not necessary to have already the result from one voxel for the processing of an other. This means that it is of no interest for the algorithm in which order the different voxels are processed.

The most important aspect in this project is the memory management. CUDA tries to fetch all data for one complete thread block in one go, assigning the memory linear to the single threads in the thread block according to the thread id. This means, that a thread with id *i* and an thread with the id *i+1* should work on data point *d* and *d+1* to be most efficient. If they don't , the “natural” memory management of CUDA would not fit to the need of the single threads and the data for each thread would have to be fetched separately instead of one single burst. This means, that the work assigned to the threads has to follow the structure of the OCT data to be processed.

An additional aspect is the limitation of the hardware for parallel processing. The NVIDIA GTX 280 video card used in this work can handle up to 512 threads per thread block. Thread blocks can be formed as an up to three-dimensional array of threads, the maximal size in each dimension is limited to 512 x 512x 64. Thread blocks are again arranged in a grid. The number of blocks within one grid is not limited, only the grid size is limited to 65535 x 65535 blocks per grid.

Figure 5 shows the resulting distribution of the work on multiple threads, as it was used in this project.

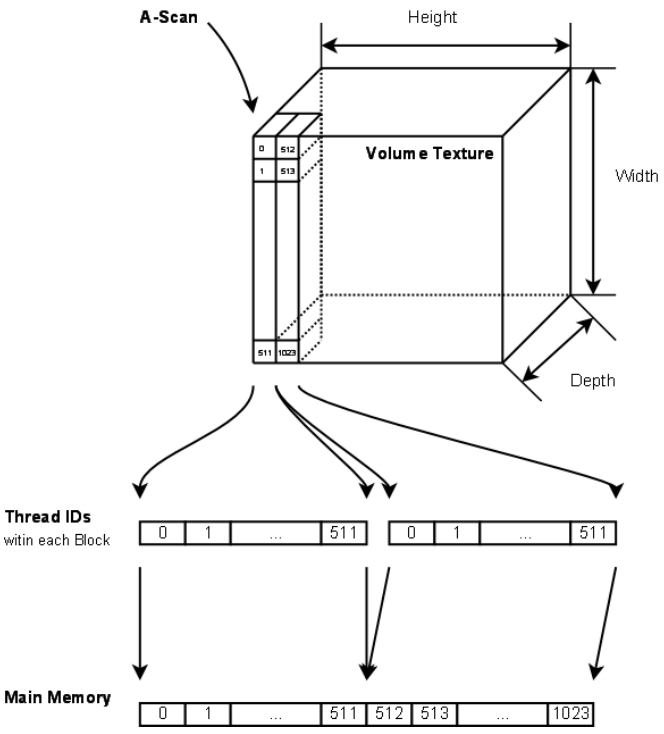

Figure 5: Distribution of data processing to parallel threads on the GPU

The OCT data is located in the memory as a series of A-scans, first A-scan of the first B-scan, second A-scan of the first B-scan, until the first B-scan is completed, than the next B-scan is following the same pattern until the volume is completed. As in this work the the detector was operated in an 1024 detector elements mode, each A-scan included 512

data points, fitting directly to the maximal allowed number of threads per thread block, each A-scan is processed by one thread block of the size 512 x 1 x 1 with 512 threads, one for each data point.

The thread blocks are arranged in an two-dimensional grid with <number of B-scans> x <number of A-scans per B-scan> in size.

4.4 3D Scenery Model

In figure 6 the 3D model used is depicted in detail, together with the adjustable rendering parameters.

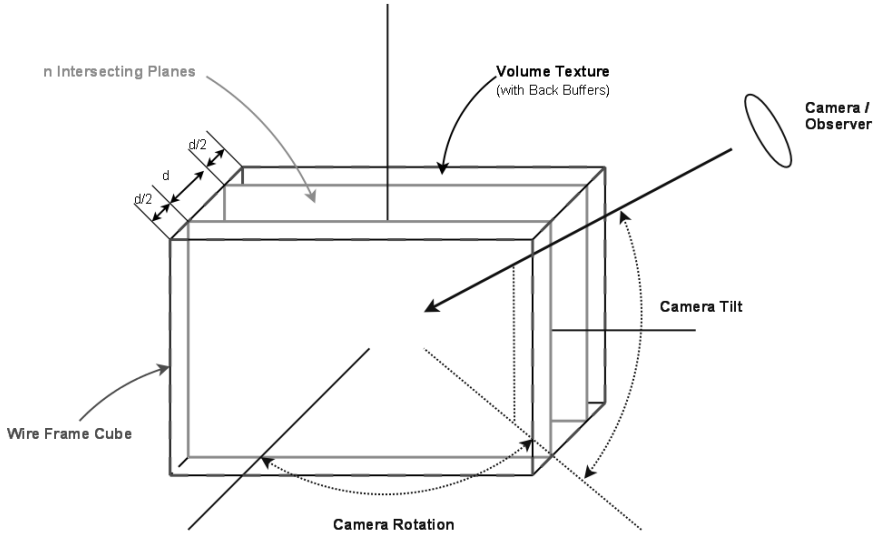

Figure 6: 3D model and rendering parameters

The cube is defining the volume covered by the volumetric texture and the two back buffering textures used. This cube - invisible in itself - can be visualized by a wire frame model of the cube, if requested by the user. The volumetric texture is intersected by the planes forming a plane stack of n planes (light gray). In figure 6 two intersecting planes are shown. The planes are located in regular distance *d* one to each other and with a distance of *d/2* to the border of the volumetric texture. On the right side the camera or observers position is depicted. The position is defined by the camera rotation and the camera tilt in addition to the distance. The observer is always facing the center of the volumetric cube.

5. RESULTS

Figure 7 shows the timing diagram of the projects result. The OCT raw data was acquired periodically from the Basler Sprint camera in 141ms or 215k A-scan rate including the fly-back-cycles of the galvos in an separate software thread.

A second thread was processing and preparing the raw data for display. Processing the detectors raw data to get OCT A-scans took 116ms for a complete volume. Due to the fly-back-cycles, which are gathered by the detector, but not converted into A-scans, this steps work at an A-scan rate of 207kHz. After the OCT data is computed from the detectors raw data it is transported onto the video adapter (11ms or 2182k A-scans/sec) and color coded on the GPU (5ms or 4.8M A-scans/sec). Including the clean-up work on the GPU, the complete process of displaying the data (data transport onto the video adapter, processing on the GPU and clean-up) took 23ms resulting in an A-scan rate of 1043kHz. The complete processing from conversion of raw data into OCT data and displaying the OCT data took 139ms giving an A-scan rate of 172kHz.

With a volume size of 300x80 this gives a measurement and online display of 7.3 volumes per second.

Figure 8 shows some screen shots of the rendered images. On the left side a nail fold is shown from different view points. On the left side a plastic cap is shown as rendered OCT volume and as an ambient light image taken through the OP microscope used for the measurements.

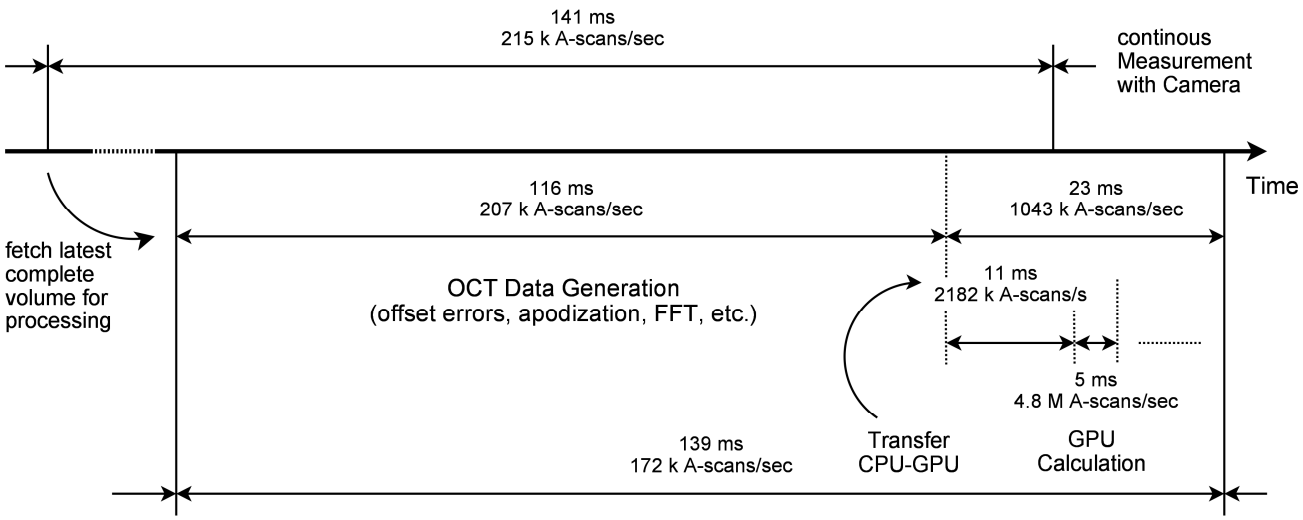

Figure 7: Timing diagram

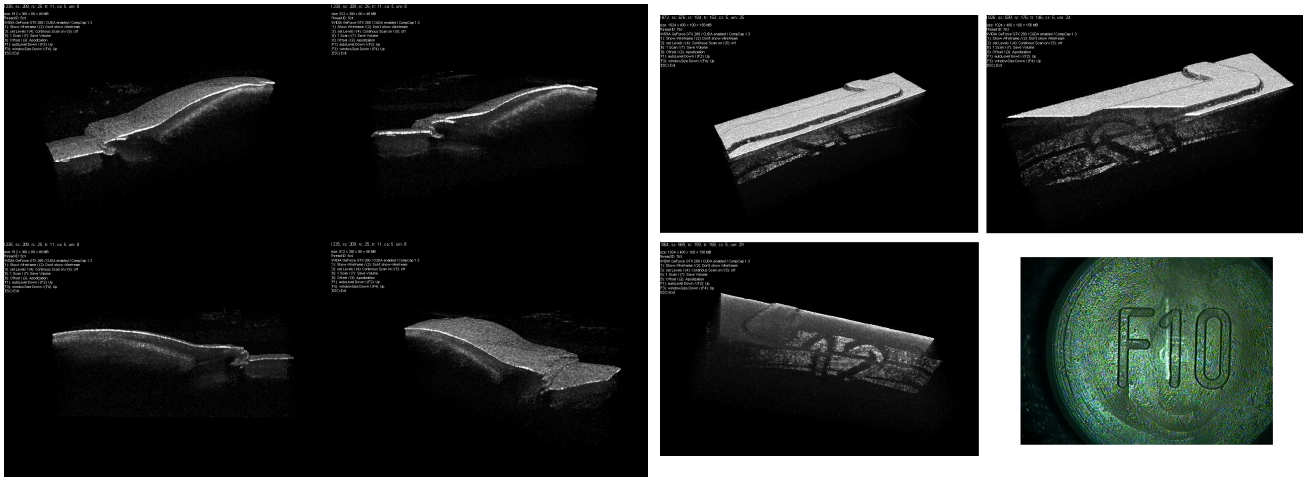

Figure 8: Rendering of a nail fold (left), rendering and ambient light image of a plastic cap (right)

The work shows that online rendering of volumetric OCT data is possible on consumer type computer hardware. It shows, too, that the process is by the used principle not limited by the computational power needed for the data display, but by the acquisition speed of the OCT system, especially the fly-back-cycles of the galvos result in a 21% loss of acquisition speed with 215kHz A-scan rate including fly-back-cycles versus 170kHz for the rendered data without fly-back-cycles.

REFERENCES

[1] D. Huang, C. Lin, J. S. Schuman, W. G. Stinson, W. Chang, M. R. Hee, T. Flotte, K. Gregory C. A. P. E. Swanson, S. E. Swanson, J.G. Fujimoto: *Optical coherence tomography*. In: Science 254 (1991), S. 1178–1181

[2] R. Huber, D. C. A. ; Fujimoto, J. G.: *Buffered Fourier domain mode locking: unidirectional swept laser sources for optical coherence tomography imaging at 370,000 lines/s*. In: Optic Letters 31 (2006), S. 2975–2977

[3] B. J. Vakoc, S. H. Yun W. Y. Oh M. J. Suter A. E. Desjardins J. A. Evans N. S. Nishioka G. J. T. M. Shishko S. M. Shishko ; Bouma, B. E.: *Comprehensive esophageal microscopy by using optical frequency-domain imaging (with video)*. In: Gastrointestinal endoscopy 65 (2007), S. 898–905

[4] X. Qi, Z. Hu W. Kang J. E. Willis K. Olowe M. V. S. Y. Pan P. Y. Pan ; Rollins, A. M.: *Automated quantification of colonic crypt morphology using integrated microscopy and optical coherence tomography*. In: Journal of biomedical optics 13 (2008), S. 054055

[5] D. C. Adler, T. H. Tsai J. Schmitt Q. Huang H. M. C. Zhou Z. C. Zhou ; Fujimoto, J. G.: *Three-dimensional endomicroscopy of the human colon using optical coherence tomography*. In: Optics Express 17 (2009), S. 784–796

[6] Barlis, P. ; Schmitt, J. M.: *Current and future developments in intracoronary optical coherence tomography imaging*. In: EuroIntervention 4 (2009), S. 529–533

[7] R. Heermann, P. R. I. C. Hauger H. C. Hauger ; Lenarz, T.: *Application of Optical Coherence Tomography (OCT) in middle ear surgery*. In: Laryngorhinootologie 81 (2002), S. 400–405

[8] G. Geerling, C. Winter H. Hoerauf S. Oelckers H. L. M. Muller M. M. Muller ; Birngruber, R.: *Intraoperative 2-dimensional optical coherence tomography as a new tool for anterior segment surgery*. In: Archives of Ophthalmology 123 (2005), S. 253–257

[9] T. Just, G. H. E. Lankenau L. E. Lankenau ; Pau, H. W.: *Intra-operative application of optical coherence tomography with an operating microscope*. In: Journal of Laryngology and Otology (2009), S. 1–4

[10] H. J. Böhringer, J. Leppert U. Knopp E. Lankenau E. Reusche G. H. D. Boller B. D. Boller ; Giese, A.: *Time-domain and spectral-domain optical coherence tomography in the analysis of brain tumor tissue*. In: Lasers in surgery and medicine 38 (2006), S. 588–597

[11] Corporation, NVIDIA: *NVIDIA CUDA - Programming Guide v2.1*. NVIDIA Corporation, 2008