Concurrent CT Reconstruction and Visual Analysis using Hybrid Multi-Resolution Raycasting in a Cluster Environment

Steffen Frey, Christoph Müller, Magnus Strengert and Thomas Ertl

Visualisierungsinstitut der Universität Stuttgart

Abstract. GPU clusters nowadays combine enormous computational resources of GPUs and multi-core CPUs. This paper describes a distributed program architecture that leverages all resources of such a cluster to incrementally reconstruct, segment and render 3D cone beam computer tomography (CT) data with the objective to provide the user with results as quickly as possible at an early stage of the overall computation. As the reconstruction of high-resolution data sets requires a significant amount of time, our system first creates a low-resolution preview volume on the head node of the cluster, which is then incrementally supplemented by high-resolution blocks from the other cluster nodes using our multi-resolution renderer. It is further used for graphically choosing reconstruction priority and render modes of sub-volume blocks. The cluster nodes use their GPUs to reconstruct and render sub-volume blocks, while their multi-core CPUs are used to segment already available blocks.

1 Introduction

CT scanners using modern flat-panel X-ray detectors are popular in industrial applications. They are capable of acquiring a set of high-resolution 2D X-ray images from a huge number of different angles at rapid pace. However, the reconstruction of a volumetric data set on a Cartesian grid from these images is very time consuming as the commonly used reconstruction method by Feldkamp et al. [1] has a runtime complexity of $O(N^4)$. Subsequently, oftentimes a computationally expensive segmentation algorithm is run to support analysis which in total results in a long delay until the examination can be started.

In this work, we focus on industrial applications, where engineers require a high-resolution reconstruction and segmentation, while it is often critical to have the results of e. g. a non-destructive quality test at hand as early as possible. We therefore propose to distribute the reconstruction and segmentation processes on a cluster equipped with CUDA-enabled GPUs and multi-core CPUs. Employing our hybrid mult-resolution renderer, finished full-resolution parts are successively displayed in the context of a low-resolution volume. The low-resolution data set can be created quickly by a single GPU within several seconds on the front-end node. It is also used for prioritizing blocks for reconstruction and render-mode selection.The full-resolution volume is progressively created by back-end nodes, which also segment and render their respective blocks.

2 Related Work

In this work, we use the reconstruction algorithm for 3D cone beam computer tomography that was developed by Feldkamp et al. [1]. Turbell [2] gives extensive and detailed overview over varitions of this method, as well as fundamentally different approaches for CT reconstruction. The use of graphics hardware for computer tomography was first investigated by Cabral et al. [3] on non-programmable, fixed function SGI workstations. Xu et al. [4] introduced a framework that implements the Feldkamp algorithm using programmable shaders. Scherl et al. [5] presented a comparison between their Cell and a CUDA implementation.

Distributed volume rendering has been investigated for a long period of time and a magnitude of publications can be found on this issue. Most of the existing systems fit either into the *sort-first* or *sort-last* category according to Molnar et al.'s classification [6]. Recent systems use GPU-based raycasting [7] with a single rendering pass [8] since GPUs support dynamic flow control in fragment programs. Dynamic load balancing issues in such systems have been addressed by Wang et al. [9] using a hierarchical space-filling curve as well as by Marchesin et al. [10] and Müller et al. [11], who both use a kd-tree in order to dynamically reorganise the data distribution in a cluster.

Multiresolution rendering is a LOD approach enabling the visualisation of large data on a GPU interactively. Different data representations are used adaptively depending on various parameters (e. g. the view point or estimated screen-space error) resulting in a quality/performance trade-off. Most frequently, trees are used as underlying data structure in combination with raycasting [12]. Ljung et al. [13] address the interpolation between different blocks with different LOD in detail. Guthe et al. [14] employ texture-based volume rendering using a compressed hierarchical wavelet representation which is decompressed on-the-fly during rendering.

The analysis of industrial workpieces using the segmentation of CT data was discussed by Heinzl [15], and the integration of segmentation information in a raycaster was discussed by Bullitt and Aylward [16] in a medical context.

3 Architecture Overview

Our reconstruction and visualisation system consists of two classes of nodes: a single front-end node and one or more back-end nodes. The front-end node exposes the user interface and displays the volume rendering of the data set that is being reconstructed. Furthermore, it allows the user to influence the order of reconstruction and the rendering mode of high-resolution sub-volume blocks on the back-end nodes. The back-end nodes in turn perform the actual reconstruction and segmentation of the high-resolution sub-volume blocks and additionally render images of these blocks, which are used by the front-end node to generate the final rendering.

Figure 1 illustrates how the distributed program is controlled by the user. The first step is to provide the input, most importantly the X-ray images. Our

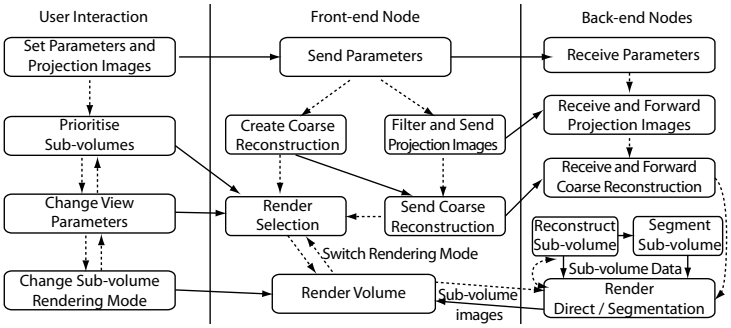

Fig. 1. Control (dashed) and data flow between the threads and processes of the system and the user. The back-end process may run simultaneously on multiple nodes.

approach is not limited to reading the projection images from disk, but could also handle images being streamed directly from the scanner while performing the incremental reconstruction. However, the time for image acquisition is in the order of minutes while high-resolution reconstruction can take hours hours in our field of application. Besides, industrial CT scanners often perform modifications (e. g. center displacement correction) after the actual acquisition.

While distributing the parameter set is completed quickly using a single synchronous broadcast operation, scattering the projection images can take a long time due to disk I/O and network bandwidth limitations. We address this problem firstly by distributing the images asynchronously while the front-end node completes a low-resolution preview reconstruction, and secondly by daisy-chaining the back-end nodes for the distribution process. This approach avoids unnecessary disk load on the front-end and the back-end nodes, which is the bottleneck in case of a high-speed interconnect like InfiniBand. Additionally, projection images are kept in main memory on the back-end nodes as long as possible to avoid I/O slowing down the reconstruction process. They are only swapped to disk if the system comes under memory pressure.

As the reconstruction of the coarse preview volume usually completes by far earlier than the distribution of the projection images, the user can start investigating the data set at an early state of the overall procedure. The preview rendering can be superimposed by a grid showing the sub-volume blocks that are reconstructed on the back-end nodes, which allows the user to specify the order of reconstruction that is then communicated from the front-end node to the back-end nodes. The visualisation of the reconstructed data set is compiled and presented on the front-end node. At the beginning, the low-resolution preview reconstruction volume data is exclusively used for local rendering. As a detailed sub-volume block has been reconstructed on a back-end node, it is rendered on the same node and included in the final rendering on the front-end node as a kind of pre-integrated sub-volume block using our hybrid raycasting and compositing approach. Additionally, the sub-volume is queued for segmentation on the multicore CPU of the respective node.

4 Implementation Details

Our system consists of three basic modules. The CT reconstruction module (Section 4.1) uses a CUDA implementation of the Feldkamp algorithm.The segmentation module (Section 4.2) is a fully automatic 3D flood-fill variant designed for distributed operation. The volume rendering module (Section 4.3) is used for image generation on the front-end and back-end nodes. Additionally, there is a sub-volume selection and picking rendering mode. It displays the coarse volume and allows the precise selection of sub-volume blocks by mouse click for moving these in the reconstruction priority queue or for choosing the visualisation mode.

4.1 Reconstruction of Sub-volume Blocks

The Feldkamp cone beam reconstruction algorithm works for industrial CT scanners which move the source in a circular trajectory shooting rays diverging as a cone through the object of interest on a detector. The algorithm can be subdivided into two phases: the preparation of the projection images and their subsequent depth-weighted backprojection into the volume. The preparation consists of weighting and filtering each image with a filter kernel derived from a ramp filter. The computationally most expensive part of the reconstruction is the backprojection, on which we will concentrate in the following. It is commonly implemented by determining for each volume element which projection image value it corresponds to by projecting it along the X-ray from the source to the detector. The depth-weighted sum of the respective pixels from all projection images yields the reconstructed voxel value.

In order to completely reconstruct a group of voxels in parallel by backprojection in a single CUDA kernel pass, all required projection image data must reside in graphics memory. This can be accomplished – even for large data sets as we focus on – by only considering one sub-volume block for reconstruction at a time such that just subsets of the projection images are needed. The dimensions of the sub-volumes are determined in a preprocessing step to cover the volume with a minimal amount of blocks considering the graphics memory available.

All projection images are cropped and stored in a single container texture, similar to the storage of renderings for the front-end raycaster (Section 4.3). The coordinates to access this texture for each voxel and every projection image are computed by projecting the eight corners of each sub-volume along the X-rays to the detector plane. These coordinates are subsequently linearly interpolated on the CPU to get the coordinate values $\boldsymbol{p}_{xyw}$ for a considered slice. Afterwards, the window position $\boldsymbol{w}_{xy}$ of the sub-image with respect to the whole image and its coordinates $\boldsymbol{i}_{xy}$ in the container texture need to be applied to the projection coordinates $\boldsymbol{p}_{xy}$. Further, $\boldsymbol{p}_{xy}$ needs to be weighted with the projective component $\boldsymbol{p}_w$ to yield $\boldsymbol{q}_{\boldsymbol{xyw}}$ that is uploaded to a texture: $\boldsymbol{q}_{xy} = \boldsymbol{p}_{xy} + \boldsymbol{p}_w(\boldsymbol{i} - \boldsymbol{w}); \boldsymbol{q}_w = \boldsymbol{p}_w$. Weighting with $\boldsymbol{p}_w$ is required to counter the effect of the projective division $\boldsymbol{c} = (\frac{\boldsymbol{p}_x}{\boldsymbol{p}_w}, \frac{\boldsymbol{p}_y}{\boldsymbol{p}_w})^T$ that takes place after the bilinear interpolation on the GPU. This finally yields the coordinates $\boldsymbol{c}$ for accessing the value of one projection image in the container texture that is backprojected on the considered voxel.

4.2 Volume Segmentation

For segmentation, we use a fully automatic 3D flood fill variant that leverages a multi-core CPU. While the GPUs are reconstructing sub-volume blocks, multiple CPU threads grow regions around a user-defined number of randomly distributed seed points in the already completed blocks. The decision on whether to add a voxel to a region is based on a user-defined threshold (e. g. the maximum range of values allowed in a region) and a gradient-based criterion.

Volume segmentation also requires some communication – between different threads and nodes – due to the fact that the algorithm must merge segments that have been created on different CPU cores or cluster machines. For merging two sub-volumes' regions that belong to different cluster nodes, the region IDs at the face they meet always has to be transmitted. Additional data needs to be transferred depending on the region criteria chosen, e. g., for a gradient-based method using central differences, it suffices to transmit a sub-volume face while value-based methods require extra region information to combine the regions.

4.3 Hybrid Multiresolution Volume Rendering Incorporating Sub-volume Blocks

Each back-end node renders images for the front-end node of the high-resolution sub-volume blocks it has reconstructed and segmented. The renderer only raycasts the pixels that lie within the image-space footprint of the current subvolume with respect to the camera parameters transmitted by the head node. As the reconstruction of a subvolume can be interrupted by rendering requests, the renderer must be able to handle sub-volumes for which high-resolution images are only available up to a certain part. The renderer therefore substitutes the missing high-resolution slices with coarse volume data that has been reconstructed by the front-end node during the initialisation phase. In order to avoid dynamic branching on the GPU and to achieve more efficient texture fetching, the coarse volume data of resolution $\boldsymbol{l}$ is appended to the texture of high resolution $\boldsymbol{h}$. Thus, the sampling coordinates $\boldsymbol{s}$ on the ray must be scaled to yield the texture coordinates $\boldsymbol{c}$ for the coarse volume past the boundary b in z-direction to low-resolution data: $\boldsymbol{c} = (\boldsymbol{s}_x \cdot \boldsymbol{\tau}_x, \boldsymbol{s}_y \cdot \boldsymbol{\tau}_y, (\boldsymbol{s}_z - b) \cdot \boldsymbol{\tau}_z + b)^T$ with $\boldsymbol{\tau} = (\frac{\boldsymbol{l}_x}{\boldsymbol{h}_x}, \frac{\boldsymbol{l}_y}{\boldsymbol{h}_y}, \frac{\boldsymbol{l}_z}{\boldsymbol{h}_z})^T$ for $\boldsymbol{s}_z > b$ and $\boldsymbol{\tau} = (1, 1, 1)^T$ otherwise (note that $b = \boldsymbol{h}_z$ for complete block).

The renderer on the front-end node combines high-resolution imagery from the back-end nodes with the coarse volume data that are available on this node into a volume rendering by integrating compositing in the raycasting loop of the front-end node. High-resolution images are raycasted by the front-end node as a kind of pre-integrated voxels. All pre-rendered images are stored in one colour texture on the graphics card, similar to the container texture of the reconstruction algorithm. Images are placed next to each other until the end of the texture is reached and then a new row of images is started at the base level of the tallest image of the previous row (Figure 2).

The information on whether a high-resolution rendering for a sub-volume exists, respectively the coordinates to access it are uploaded to the graphics

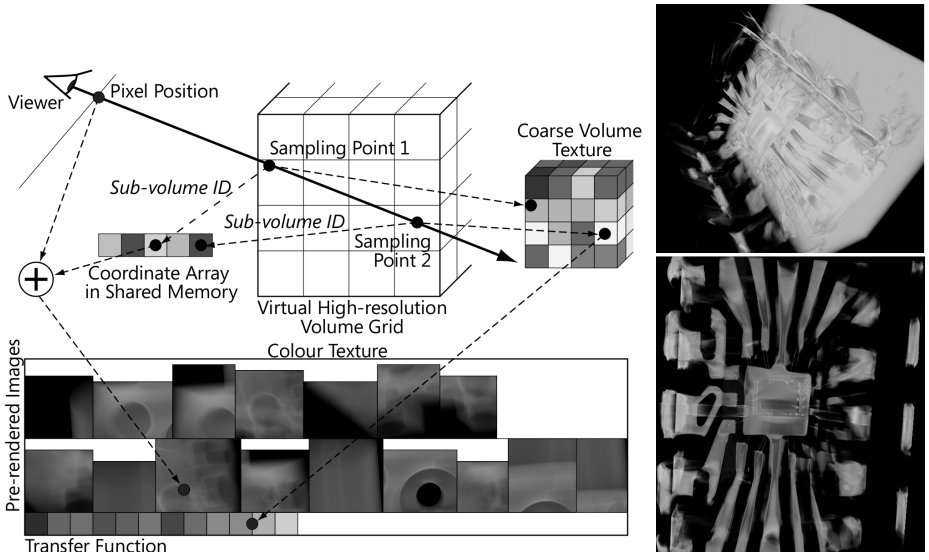

Fig. 2. Left: When rendering the final image, it is determined for each sampling point whether there is pre-rendered high-resolution data available (sampling point 1) or not (sampling point 2). As the case may be, different coordinates are used to access the colour map. Right: Renderings of the IC datset resulting from this technique.

card's shared memory for efficient access. Two 16 bit integers per sub-volume are utilised to determine the texture coordinates of a pre-rendered pixel for the sample point of a ray. These coordinates have already been pre-modified such that the pixel position of a ray only has to be added to fetch a pre-rendered pixel. Due to the use of shared memory and the overloading of the colour map access, no actual branching is required and the amount of texture memory accesses in the sampling loop is the same as of a standard single pass raycaster: one volume texture fetch requesting a scalar density value and one colour transfer texture lookup retrieving a 4D vector. Yet here the colour texture lookup is also used for accessing a rendered high-resolution image, depending on the sub-volume the respective sample is in.

4.4 Communication and Data Exchange

The communication patterns of our application differ in two different phases: At the beginning, the input parameters and projection images are distributed synchronously, the latter in a daisy-chain from one back-end node to the other. This alleviates the disk I/O load on the front-end node, which initially stores the input data, and introduces only a small latency in the distribution pipeline.

After initialisation, both node classes enter a message loop for asynchronous communication. For the front-end node, this is equivalent to the message loop of the window system, which starts as soon as the preview volume has been recon-

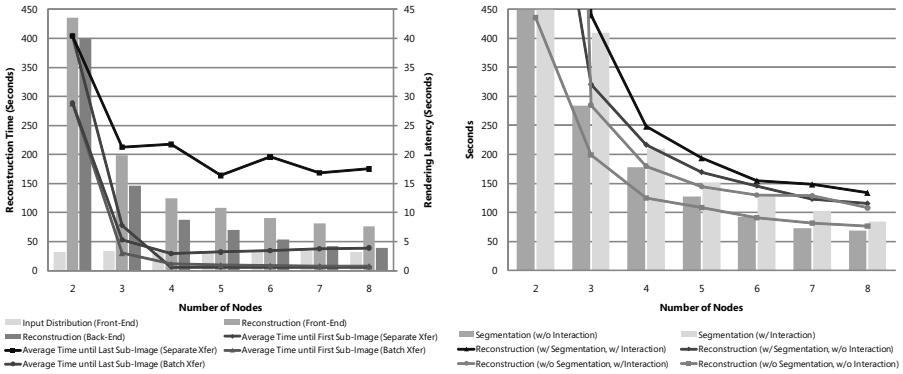

Fig. 3. Left: Timing results for several cluster configurations. The bar chart shows initialisation and reconstruction times on the front-end and back-end nodes (units on the left), while the lines show the average time from a remote rendering request until the reception of the first/last sub-volume image on the front-end node (units on the right). Right: Mutual influence of reconstruction, segmentation and user interaction during the segmentation.The results for only one back-end node are clamped for clarity.

structed. The back-end nodes have a second message loop for the communication of the segmentation subsystem in order to decouple this process completely from the reconstruction and rendering tasks. The message handling of communication with the front-end can interrupt the reconstruction process, which is running whenever no more important requests must be served, e. g. for requesting high-resoluting renderings. It may therefore take a significant amount of time from issuing a rendering request until all sub-images are available. Hence, images are received asynchronously and replace parts of the local preview rendering as they come in.

The assignment of reconstruction tasks to back-end nodes is carried out by the front-end node as it must be possible to re-prioritise blocks on user request. Load-balancing is implicitly achieved by the back-end nodes polling for new tasks once they have completed a sub-volume block.

5 Results

We tested our system on an eight node GPU cluster with an InfiniBand interconnect. Each node was equipped with an Intel Core2 Quad CPU, 4 GB of RAM, an NVIDIA GeForce 8800GTX and a commodity SATA hard-disk. One node acted as front-end creating a 256^3 voxel preview volume, while the remaining ones reconstructed a 1024^3 volume from 720 32-bit X-ray images with a resolution of 1024^2 pixels (Figure 4 shows the volumes). The calculated sub-volume size was 352^3 resulting in a total of 27 sub-volumes. The time from program start until the preview volume is reconstructed on the frontend-node and rendered is around 29 s, of which the most part is required for the I/O caused by projec-

tion image downsampling that runs in parallel with the data distribution, while the actual reconstruction on the GPU takes only 1.3 s. The determination of the sub-volume dimension that takes a few seconds only runs concurrently. Figure 3 (left) shows the data distribution and reconstruction times measured on the front-end and the back-end nodes. Note that in contrast to the separate transfer, the batch transfer may interrupt the reconstruction of a sub-volume. The times for the front-end node also include communication overhead and show the span between program start and the availability of the complete high-resolution volume. In contrast, the numbers for the back-end nodes comprise only the longest computation. So although the average reconstruction time on the back-end nodes quickly decreases with an increasing number of nodes, the observed time on the front-end node declines more slowly, because this timing includes the input distribution and other communication. Input distribution takes slightly longer the more nodes are involved, because the measurement on the front-end node includes the time from reading the file from disk until the last node received the data. Thus the last node in the daisy chain must wait longer for its data.

The rendering times depicted in Figure 3 (left) indicate the time span between the moment the front-end node requests new sub-volume images and the moment the first respectively the last remotely generated image is used in the visualisation. Images from the back-end nodes can either be sent in batches or as separate messages. In our measurements, we let the batch requests – in contrast to the separate requests – interrupt the reconstruction not only after a sub-volume block but already after a sub-volume slice has been completed. For the reconstruction, this means that after rendering projection images have to be re-uploaded to the graphics card, resulting in a slightly worse rendering performance than when interrupts are prohibited. For the separate transfer of sub-images there is only little latency between the request and display of an image on the front-end node which gives the impression of a more fluid interaction, while in our test setup the time until the last image is received on the head node is much longer. This is partly due to the decrease in network throughput and the potential interruption of the image transfer by higher priority messages. But more significantly, a node potentially has to wait the reconstruction time of up to 14 s for a sub-volume block to be completed by the reconstruction until rendering can be started. This happens more often the more nodes are involved and hinders scaling with the cluster size of the average latency until the last image has been received. Please note, however, that the system always remains responsive as it can use the coarse volume data that is available from the beginning.

Figure 3 (right) points out the mutual influence of the reconstruction, the segmentation and user interaction during the reconstruction phase. Frequently interrupting the reconstruction by manipulating the scene increases overall reconstruction time and subsequently also segmentation time. The latter is caused by the fact that all sub-volumes have to be read for rendering from disk resulting in reduced I/O performance of the segmentation threads. The same holds true for reconstruction performance, but in this case both computations additionally conflict in the usage of the GPU.

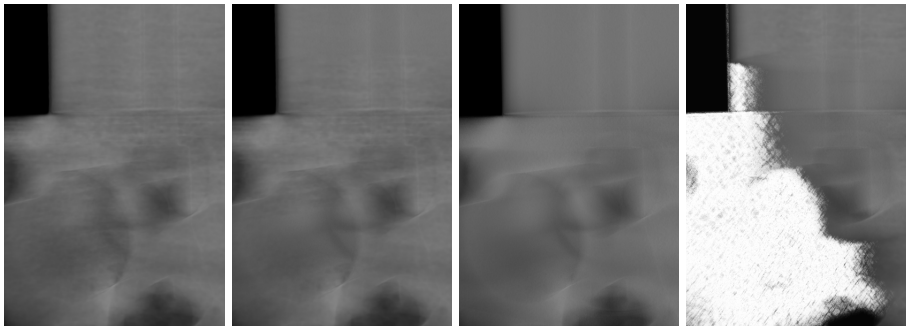

Fig. 4. From left to right: The coarse 256^3 volume, a partially reconstructed volume, the fully reconstructed high-resolution 1024^3 volume and an intermediate state when the left part of the volume has already been segmented.

When reconstructing a 2048^3 volume from 1440×2048^2 projection images on eight nodes, data distribution takes ~ 13.5 min. It is limited by disk I/O as nodes need to read and write data simultaneously. The reconstruction needs $\sim$103 min, which is about 84 times longer than reconstructing the 1024^3 volume. The scaling behaviour is severly hindered by I/O performance in this case due to the excessive need for data swapping, which affects primarily the storage/access of the input data (23 GB versus 4 GB of RAM). Sub-volume sizes are further limited to 256^3 due to GPU memory restrictions, which requires a total of 512 sub-volumes to cover the complete volume – resulting in 19 times more accesses to projection images and sub-volumes. This significantly adds to the permanent memory pressure as the ratio of projection images and sub-volumes that can be stored in memory is already eight times worse in comparison to the 1024^3 case.

6 Conclusion and Future Work

We introduced a distributed software architecture for the 3D reconstruction of computer tomography data sets while continuously visualising and segmenting. Exploiting the computational power of GPUs, our system provides a fast preliminary reconstruction and visualisation, which allows for prioritising interesting sub-volume blocks. While the reconstruction is in progress, the visualisation is continuously kept up-to-date by integrating all available high-resolution blocks immediately into the rendering. For that, we use a hybrid CUDA-based volume raycaster that can replace low resolution blocks with pre-rendered images. We also leverage multi-core CPUs for parallel volume segmentation, which is often utilized for advanced processing steps as well as the support of visual analysis. As this is executed in parallel with the reconstruction on the GPU, the user can switch the render mode to segmentation on a per block base display shortly after the sub-volume has been reconstructed. In our tests, we could see a 256^3 preview of a 1024^3 data set after 30 seconds and the full resolution volume after less than three minutes. A large 2048^3 volume could be reconstructed in 103

minutes, but the process was hampered by the limited I/O performance and GPU and main memory. Testing our system with a bigger cluster and a faster distributed filesystem therefore remains for future work. Our architecture could be further extended by (semi-) automatically assisting the user during the subvolume selection by pointing out regions that could be of high interest, e. g. by identifying areas with large gradients in the coarse volume.

References

1. Feldkamp, L.A., Davis, L.C., Kress, J.W.: Practical cone-beam algorithm. J. Opt. Soc. Am. **1** (1984) 612–619
2. Turbell, H.: Cone-Beam Reconstruction Using Filtered Backprojection. PhD thesis, Linköping University, Sweden (2001) Dissertation No. 672.
3. Cabral, B., Cam, N., Foran, J.: Accelerated Volume Rendering and Tomographic Reconstruction using Texture Mapping Hardware. In: Proceedings of the Symposium on Volume Visualization. (1994) 91–98
4. Xu, F., Mueller, K.: Accelerating popular tomographic reconstruction algorithms on commodity pc graphics hardware. IEEE Transactions on Nuclear Science (2005) 654–663
5. Scherl, H., Keck, B., Kowarschik, M., Hornegger, J.: Fast gpu-based ct reconstruction using the common unified device architecture (cuda). SIAM Journal of Applied Mathematics (2007)
6. Molnar, S., Cox, M., Ellsworth, D., Fuchs, H.: A sorting classification of parallel rendering. IEEE Computer Graphics and Applications **14** (1994) 23–32
7. Krüger, J., Westermann, R.: Acceleration Techniques for GPU-based Volume Rendering. In: Proceedings of IEEE Visualization '03. (2003) 287–292
8. Stegmaier, S., Strengert, M., Klein, T., Ertl, T.: A Simple and Flexible Volume Rendering Framework for Graphics-Hardware–based Raycasting. In: Proceedings of the International Workshop on Volume Graphics '05. (2005) 187–195
9. Wang, C., Gao, J., Shen, H.W.: Parallel Multiresolution Volume Rendering of Large Data Sets with Error-Guided Load Balancing. In: Eurographics Symposium on Parallel Graphics and Visualization. (2004) 23–30
10. Marchesin, S., Mongenet, C., Dischler, J.M.: Dynamic Load Balancing for Parallel Volume Rendering. In: Eurographics Symposium on Parallel Graphics and Visualization, Eurographics Association (2006) 51–58
11. Müller, C., Strengert, M., Ertl, T.: Optimized Volume Raycasting for Graphics-Hardware-based Cluster Systems. In: Eurographics Symposium on Parallel Graphics and Visualization, Eurographics Association (2006) 59–66
12. Crassin, C., Neyret, F., Lefebvre, S., Eisemann, E.: Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering. In: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D), Boston, MA, Etats-Unis, ACM, ACM Press (2009) to appear.
13. Ljung, P., Lundström, C., Ynnerman, A.: Multiresolution interblock interpolation in direct volume rendering. In Santos, B.S., Ertl, T., Joy, K.I., eds.: EuroVis, Eurographics Association (2006) 259–266
14. Guthe, S., Strasser, W.: Advanced techniques for high quality multiresolution volume rendering. In: In Computers and Graphics, Elsevier Science (2004) 51–58
15. Heinzl, C.: Analysis and visualization of industrial ct data (2009)
16. Bullitt, E., Aylward, S.R.: Volume rendering of segmented image objects. IEEE Transactions on Medical Imaging **21** (2002) 200–2