Volume Graphics (2006)
T. Möller, R. Machiraju, T. Ertl, M. Chen (Editors)

GPU-Accelerated D^{2}VR

Fang Xu Klaus Mueller

Center for Visual Computing, Computer Science, Stony Brook University

Abstract

Traditional volume rendering approaches rely on obtaining values of sampled points in volumetric space, typically on a cartesian grid. Often, this cartesian grid is not the original source of the data. For example, in tomographic imaging applications, such as used in diagnostic medical or industrial CT, the primary source of the data is the set of X-ray projections taken around the object. To enable visualization with established volume rendering methods, the volume must first be reconstructed from these projections. Since sampling is involved, this process introduces errors, adversely impacting image quality. Recently a new rendering technique was proposed, named D^{2}VR, which skips the intermediate reconstruction step entirely and samples the projections directly. It was shown that doing so can improve image quality significantly. But despite its great promise, a shortcoming of the method was its comparatively slow rendering speed. Interactive or at least near-interactive speed, however, is critical for clinical deployment of a visualization framework. To address this shortcoming, our paper proposes a GPU-accelerated D^{2}VR, with facilities for occlusion culling and empty space skipping to achieve further speedups.

Categories and Subject Descriptors (according to ACM CSS): I.3.3 [Computer Graphics]: Display Algorithms.

1. Introduction

Volumetric datasets can have many origins. They can be the output of numerical simulations, such as CFD, finite elements, finite difference, and other calculations of this nature. They may also be generated by inverse Fourier transform, which is the case for MRI imaging. And they can be the product of the voxelization of analytical functions or polygonal objects. Finally, they can result from tomographic reconstruction (CT), which is a process invoked whenever an object is scanned with a transmissive radiation, such as X-ray, ultrasound, or infrared light. In CT, an object is irradiated with a transmissive source on one side and a projection is acquired on the other side. This process is repeated at a sufficient range of viewing angles, and the resulting projection set is processed in the CT reconstruction procedure. Major applications are in medical imaging and also in industrial CT and security. The well-known engine dataset, for example, was obtained via industrial CT. Medical imaging with CT is ubiquitous. It is a relatively inexpensive scanning technology, when compared to MRI, and has many diagnostic applications in medicine. Almost all parts of the human body can be imaged and diagnosed with CT imaging. This paper is dedicated to the large body of datasets obtained with CT reconstruction methods.

There are a number of different scanning geometries: parallel beam, fan-beam, cone-beam, and spiral (helical) CT. Reconstruction algorithms for the former two geometries are referred to as slice-based CT, while the latter are referred to as volumetric CT. In fact, spiral CT has become a multi-slice acquisition method, that is, a stacked (multi-slice) array of detector arrays rotates about the patient, with a cone-beam source irradiating the patient. Nowadays, spiral CT scanners with up to 64-slices are employed in clinical practice. For fan and spiral beam geometries, there exists the ability to rebin the projection data into parallel-beam data. If a sufficient amount of projection (ray) data have been collected, then one may sort these rays into equivalent bins of parallel beam rays, which can then be used in conjunction with conventional parallel-beam CT reconstruction methods. However, there are also reconstruction methods for fan-beam, cone-beam, and spiral CT which do not use rebinning. This is the domain of the exact reconstruction methods, as described in [KS88, Kat04, KND00, TSS98, TD00, Gra91] and others. Finally, there are also approximate methods that work well in practice and are very popular, under certain conditions. For example, Feldkamp's algorithm [FDK84] is often used in cone-beam reconstruction and produces good results for sufficiently small cone-angles < 20°.

The original volume rendering framework of D^{2}VR [RCG*06] assumes that the CT reconstruction resulting in the volume dataset is (or better, would have been) obtained with parallel beam data and algorithms. However, direct fan

©The Eurographics Association 2006.

and cone-beam reconstruction geometries would also be feasible with D^{2}VR. Finally, the more complicated reconstruction techniques used in advanced spiral-CT algorithms could be considered as well, but these would produce a larger amount of overhead.

CT reconstruction is a data conversion process, and due to associated sampling with imperfect filters, it is a lossy data conversion process. CT reconstruction is needed to convert the data into the format used by traditional volume renderers. D^{2}VR, on the other hand, is a non-traditional volume renderer, which does not require the data conversion and instead produces volume renderings directly from the raw projection data, eliminating the errors incurred in the conversion process.

D^{2}VR traverses the volume space as usual, but instead of performing the sampling there, it maps the sample positions into each projection image and interpolates the data in those. A sample value is then composed by adding all contributions so obtained. By finding the derivatives in each projection image, one can reconstruct the sample gradient in a similar fashion. Once the sample and gradient values have been obtained, the usual transfer function lookup, shading, and compositing can take place. In essence, the 3D array of sample values that ensue from this process are those that would be generated with a parallel-beam CT reconstruction, had the volume grid been placed at this orientation. Therefore, one may say that the result using an axis-aligned volume rendering with traditional techniques and D^{2}VR are identical (assuming the gradients were calculated in volume space in both cases).

One should note, however, that the CT-based data conversion process does not only produce data in a format that is more convenient to render, it also reduces the overall data complexity. Assuming fan-beam geometry and rendering at no magnification, alias-free reconstruction requires $\pi/2 \cdot N$ projections to reconstruct a slice with N^2 voxels [KS88]. Thus, with D^{2}VR, each sample requires $\pi/2 \cdot N$ bilinear interpolations, which is substantially more effort than the K^3 neighbors needed for an interpolation in volume space (with K being the 1D extent of the interpolation filter). Even with acceleration techniques, such as empty space skipping and early ray-termination, which reduce the number of samples to be computed (at complexity $\pi/2 \cdot N$), GPU-assistance in this task still seems necessary to overcome this great computational burden. Our paper describes such a GPU-acceleration approach.

The paper's structure is as follows. First, in Section 2, we briefly discuss related work on GPU-accelerated volume rendering and CT. Then, in Section 3, we describe our basic GPU CT reconstruction framework, which is followed by Section 4 where we describe the GPU accelerated D^{2}VR. Finally, Sections 5 and 6 present results and conclusions.

2. Related Work

The combination of volume rendering and CT on graphics hardware is not new. Already in 1995, Cabral et al. [CCF94] utilized the inverse relationship of these two procedures to derive a common mathematical theory for both plus a common framework that would accelerate them on SGI texture mapping hardware. The capabilities of this hardware were quite limited, and in this early work the hardware was mainly used to accelerate the rasterization effort. Much of the work, such as accumulation and filtering in CT, had to be performed on the CPU. There was also no direct connection of CT and volume rendering, such as the one that was formulated for D^{2}VR. The same hardware was later used by Mueller and Yagel [MY00] to accelerate iterative CT algorithms, such as SART (Simultaneous Algebraic Reconstruction Technique). This approach showed how two color channels with 8-bit precision could be combined to reach higher precision for integer-arithmetic. With the evolution of GPUs more sophisticated CT implementations were possible. Xu and Mueller [XM05] described a general framework for GPU-accelerated CT, spanning iterative and analytical algorithms for transmission (CT) as well as emission tomography (PET, SPECT), fully accelerated on the GPU. They achieved speedups of 1-2 orders of magnitude, compared with CPU implementations of the same accuracy. Chidlow and Möller [CM03] described a GPU-accelerated implementation for SPECT imaging, but the accumulation stage was exported to the CPU.

The GPU has been the source of many acceleration efforts for volume rendering as well. As mentioned before, the common ancestor for both domains is Cabral et al. [CCF94], but the work that followed on the volume rendering track was much more prolific. While the implementations using the SGI rendering hardware were constrained by the limited set of operations, the revolution of the more recent PC-based graphics hardware took away most of these restrictions. In the following, we shall just name a few of the most prominent advancements, for regular grids. First, there is the work by Rezk-Salama et al. [RSE*00], which used multi-texturing to enable fully-hardware based volume rendering, and there is the work by Engel et al. [EKE01], which introduced pre-integrated volume rendering to eliminate the stair-stepping artifacts caused by the common slice-based rendering paradigm. A more recent work is that by Krüger and Westermann [KW03] who describe a ray-casting implementation, fully GPU-accelerated. Their implementation also includes mechanisms for early ray-termination and empty-space skipping, the latter by using a low-resolution occupancy octree. Neophytou and Mueller [NM05] showed how the z-buffer's early fragment-kill capabilities can be exploited to skip over empty space and voxels that would project into already opaque image regions. The latest development is the system proposed by Stegmeier et al. [SSE05], which completely eliminates the use of slice rasterization

©The Eurographics Association 2006.

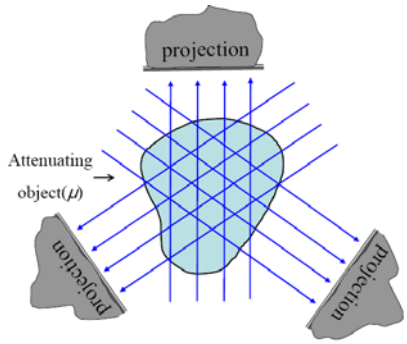

Figure 1: Principle of computed tomography

and runs the entire ray advancement in a single fragment shader loop. This bears some advantages for the implementation of non-linear ray effects, such as refractions.

The approach presented here builds on our GPU-accelerated CT framework for filtered backprojection [MX06], which we call RapidCT (http://www.rapidCT.com) but generalizes it substantially. First, it allows the reconstruction of arbitrary oriented volumes, which is not needed for CT, but is necessary for D^2VR since we must generate a matrix of samples exactly aligned with the image plane, whose orientation is completely arbitrary. Second, we incorporate various acceleration techniques to limit the reconstruction effort to only the visible and non-occluded (the relevant) matrix samples. Our system enables framerates of 2 frames/s and more for realistic dataset sizes.

3. GPU Accelerated Filtered Backprojection

In this section, we will first briefly discuss the theory of CT reconstruction with filtered backprojection, generalized to the cone-beam projection geometry. Then we will outline two different methods that are well suited for its acceleration on the GPU. These methods make ample use of the fast built-in circuitry on GPUs.

3.1. Theory and Algorithm

Filtered backprojection is the most popular algorithm for computed tomography. Feldkamp's (FDK) algorithm [FDK84] is a popular method used for cone-beam CT, but when executed in orthographic mode it is essentially a 3D extension of a series of 2D filtered backprojections, one for each row of the 2D projection. By assuming a circular trajectory with the object under reconstruction placed in the center, we can model and discretize the equation of transmissive tomography of a volumetric model on a grid size of N^3 as (see also Fig. 1):

$$p_i = \int_0^L \mu(t)dt = \sum_{j=0}^{N^3-1} \mu_j w_{ij} \qquad (1)$$

Here, t is the parametric variable defined along the ray, and L is the distance between the source and the detector bin. The μ are the attenuation factors of voxels while the p are the pixel values recorded on the detector. The w_{ij} are the weights with which the values of the μ_j contribute to the pixels p_i. The backprojector is the inverse of (1). It is written as:

$$v_j = \sum_{p_i \in P_\varphi} p_i w_{ij} \qquad (2)$$

where the v_j can be any quantity to be reconstructed (here, μ) and the w_{ij} are as before.

The FDK algorithm first filters the acquired images along their columns, using a ramp filter such as Shepp-Logan, Ram-Lak, etc. [KS88], before backprojecting them into the volume. The filtering is efficiently done in the frequency domain, since the filter is usually large in the spatial domain. The frequency transform can be performed using FFTW (http://www.fftw.org) or on the GPU directly [JRH*04].

For a cone-beam geometry, during backprojection, the w_{ij} must be multiplied by a depth correction factor:

$$w_{ij(d)} = w_{ij} \frac{a^2}{\left(a + \sqrt{Y(v_j) + Z(v_j)}\cos(\varphi - \varphi_r)\right)^2} \qquad (3)$$

Here, Y and Z return a voxel's y and z coordinate, a is the source-axis distance, φ_r is the principal orientation angle of the r-th projection ($\varphi = arc\tan((Y(v_j))/(Z(v_j)))$), and $w_{ij(d)}$ is the depth-weighted w_{ij} in (2).

3.2. GPU Implementation

A straightforward method to implement backprojection on the GPU is to simulate the procedure of projecting a volume slice onto the image plane, which is done by using *projective textures* [SKv*92]. It computes the transformation matrix between the projection and the volume slice to be backprojected, and then uses it as the texture matrix to guide the "reverse" texture mapping. An FDK implementation using projective textures requires two volume slice stacks, one for each major direction, X and Z (Y is the axis about which the projection rotates, see Fig. 2). Projections are divided into two sets accordingly, depending on which axis (X or Z) they are more perpendicular to. Each volume slice then receives backprojection from all projection images that belong to the corresponding set. With this implementation, two copies of the volume slices need to remain in the GPU memory at any time, occupying precious video memory. In addition, two stacks of volume slice must be merged at the end of the reconstruction to produce the final result.

Alternatively, we can sample the volume into a horizontal stack whose axis is aligned with the rotation axis. Then under orthographic settings, each row of the projection is bound as a

©The Eurographics Association 2006.

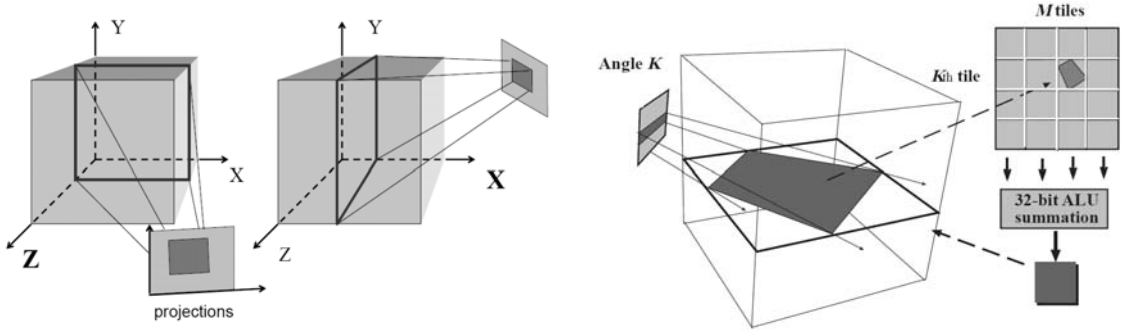

Figure 2: Two alternative backprojection methods: (left) projective texture, (right) texture spreading

1D texture and spread across the buffer to form its contribution to that volume slice. This method was also used, for parallel-beam projection by [CM03], and then extended to cone-beam by us [MX06]. Here, we group the spreading operations from all projection angles with respect to a certain volume slice, accumulate all spreading results, and then move forward to the next slice. During the accumulation, we create a large texture sheet containing sub-textures of individual contribution from each projection angle. Then a multi-pass accumulation step is performed in the fragment shader to sum the tiles up, which completes the reconstruction of a volume slice. An alternative way to implement this is to compute the transformation matrices for each voxel, which designates its sampling locations on the projections. We iterate through a loop for all projection angles in the fragment shader to fetch and sum all sample values to every voxel on a slice. With the spreading setup, only one slice stack is needed to represent the volume and therefore no merge operation needs to be executed in the end. But the drawback for this algorithm is that all projections need to remain in memory until the whole volume is reconstructed.

3.3. Acceleration by Precision Tuning

Computed tomography applications involve the processing of large amounts of data which can easily introduce memory traffic congestions due to pipeline stalls on limited bandwidth. So it is natural to reduce the size of the input dataset by using integer representations. This strategy can partially alleviate the bandwidth problem, since now more projections can be streamed through the GPU within the same amount of time, which yields real speedup. Since projection datasets acquired from commercial CT scanners are usually in 8-bit or 16-bit precision format, it is justifiable to conduct the backprojection operations in 8-bit through the fixed-function pipeline (although sometimes the pre-filtering operation may produce higher precision depth). Note, however, that the accumulation still has to be done in floating point precision to avoid overflow.

To test these theories we have conducted various experiments using the 3D Shepp-Logan brain phantom, the industry standard for CT. It has features of extremely low contrast (less than 0.5%), which is about one intensity level in the typical range used in volume rendering, [0, 255]. There, we have seen that narrowing down the dynamic range from 32 to 8 bits during the backprojection procedure will result a loss of detail, which subsequently leads to streak artifacts. These artifacts are much reduced on datasets whose contrast constraint is slightly weakened (1%). Thus, while this will not be quite sufficient for high-precision volume rendering, it may serve as a previewing mode. As a compromise, a "pseudo 16-bit" scheme was proposed to enhance the reconstruction quality of the 8-bit rasterization pipeline without losing its faster processing ability. For this, we first compress the dynamic range of the 32-bit data into 16-bit integer words. We then separate each 16-bit word into higher and lower 8-bit bytes. Two backprojection passes are performed individually to generate two accumulation sheets from both bytes. During the accumulation stage, the sheet computed from the higher 8-bit byte is shifted to the left and added to the other one. Note this method loses the information of the lower 8-bit of the higher 8-bit texture during the interpolation, but in practice, for the Shepp-Logan phantom, we were able to achieve excellent reconstruction results [MX06]. The two-pass strategy takes twice as long as the one-pass 8-bit reconstruction, but it is still 5 times faster than the full floating point reconstruction.

4. GPU D^{2}VR Implementation

In this section, we first describe the basic GPU D^{2}VR framework and then the various techniques which can be used to accelerate it further.

4.1 Basic GPU D^{2}VR Framework

Let us assume an arrangement in which all projections are distributed around a circular orbit. In order to achieve a maximal volume resolution N^3, which we would like to reconstruct without aliasing, we need to have $M=\pi/2 \cdot N$ projections distributed around a half circle, assuming parallel beam data. It is our goal to be able to reconstruct image-aligned volume slices in front-to-back order, since this will enable us to perform occlusion culling. Let us assume that

©The Eurographics Association 2006.

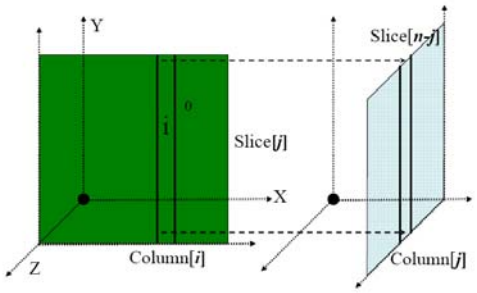

Figure 3: Shuffling operation to swap texture stacks

the projections are perpendicular to the x-z plane. Then we can use the projective texture method to reconstruct volume slices given that the viewing vector is tilted within a range of ±45° perpendicular to the x-z plane. If the vector exceeds this range, we must either provide another orthogonal set of projection images arranged in a circular orbit around the x-y plane, or we switch to the texture spreading method. In the latter case, we will be able to make do with just one set of projections. But if we only want to use one type of method for all viewpoints, then we need to provide two orthogonal half-orbits of projections, equally spaced in angle. This, in fact, is in line with Tuy's condition [Tuy83] for exact 3D reconstruction from projection data.

When the projective texture method is used, we first rotate the volume proxy polygons so that one slice set is perpendicular and the other is orthogonal to the viewing vector. We define these slices as viewing (V) and support (S) stacks, respectively. Accordingly, we divide the projections into one V-set and one S-set, based on which stack they are more perpendicular to. The S-stack is first reconstructed from projections from the S-set. Then we shuffle the partially reconstructed volume into the V-stack, which is used for rendering. The data transfer can be implemented on the GPU by repeatedly rendering different columns of a source texture from one stack to the same column on all target textures from the other stack (see Fig. 3). This operation is quite fast. We then use the V-stack to fully accumulate/reconstruct the volume, at which time we can shade and composite.

4.2 Gradient Estimation

In D^{2}VR, we have two choices. We can either reconstruct only the densities and then compute the gradients in volume space directly by central-differencing adjacent samples in all three orthogonal directions, or we can estimate the gradients in projection space and reconstruct the gradients as well. Note that when computed in projection space, the two gradient components more parallel to the viewing direction should be scaled with cosθ and sinθ, where θ is the projection angle with respect to the image. We call this approach *gradient-from-reconstruction (GFR)*. The projection space gradient is stored into the RGB color channels with the alpha channel carrying the filtered density values and both properties are reconstructed at the same time. The other approach, called *gradient-from-samples (GFS)*, reconstructs only the density values and performs the gradient estimation from these samples in volume space. This reduces the memory bandwidth in the accumulator. It benefits the reconstruction stage but slows down the rendering procedure, but since memory bandwidth is usually the GPU-bottleneck, this tradeoff is advantageous. The computation of gradients in this approach will require 6 neighborhood samples when the central-difference operator is used, and each such sample should be a direct neighbor.

The GFS also tends to produce images of higher perceived definition (or visual sharpness and acuity) than the GFR. As a justification, consider the following. Let us first compare the D^{2}VR with regular DVR in terms of their filter pipelines:

$$\begin{aligned} \boldsymbol{CT+DVR} &\rightarrow R\{p \otimes h_p\}^N \otimes h_v \\ \boldsymbol{D^2VR} &\rightarrow R\{p \otimes h_p\}^N \end{aligned} \tag{4}$$

In this equation, p are the projection data, h_p and h_v are the interpolation filters in projection and volume space, respectively, and R is the reconstruction operator with N being the number of projections. This is a formal way to show that D^{2}VR's filter pipeline only involves one interpolation filter, and thus produces lesser artifacts.

Most gradient filters, in particular the central difference filter, fall off towards the highest frequencies and mostly accentuate the midrange frequencies (see e.g. [BBT96]). This is a good feature in some respect since it reduces the effect of noise, which typically resides in the higher bands. But on the other hand, the sharpness of the gradients also suffers, since the desirable (signal) portions of the higher bands are now missing. The difference of GFP and GFS are most prominent when upsampling during the D^{2}VR, that is, when the viewport has a higher resolution than the reconstructed volume slices. We shall explain this now. Consider the following pair of filter pipelines for gradient computation, assuming D^{2}VR is used for both:

$$\begin{aligned} \boldsymbol{GFP} &\rightarrow R\{(p \otimes g_p) \otimes h_p\}^N \\ \boldsymbol{GFS} &\rightarrow R\{p \otimes h_p\}^N \otimes g_v \end{aligned} \tag{5}$$

We observe that with the GFP method the gradient filter g_p is applied first, in projection space (followed by filtering with the ramp-filter), and then, within the reconstruction R, the upsampling of the above-mentioned band-passed frequency spectrum is performed, using interpolation filter h_p. On the other hand, for GFS the unlimited (ramp-filtered) frequency spectrum is interpolated first, using h_p in reconstruction R, which then undergoes the band-passing of the gradient filter, in volume space. Here, the upsampling plays an important role. It stretches the most-active (mid-range) frequency window of the gradient filter into the higher frequency bands of the reconstructed slices (that is, it better approximates the

©The Eurographics Association 2006.

ideal ramp filter), and thus accentuates these higher frequencies more than GFP. This leads to stronger gradients, and therefore sharper object features, but also possibly to enhanced noise.

For D^{2}VR there is a special advantage that comes with the texture-spreading method. If projections are spread onto the slice stack orthogonal to the view point, we can immediately start shading and compositing the reconstructed slice as soon as the accumulation process from all projections is finished. This assumes that the gradients are reconstructed. Just like in the projective texture method, when only the density values are reconstructed, we need to wait until the slice behind it is generated, in order to compute the gradients using the central difference operator.

4.3 Viewport vs. Volume Resolution

When determining the resolution of the volume slices to be reconstructed from the projection data to obtain an image at a certain viewport resolution, one should realize that CT projection data of a certain resolution, say N^2, will not be able to yield a volume of higher resolution than N^3 (assuming there are a sufficient number of them, theoretically $\pi/2{\cdot}N$, as mentioned before). This is due to the frequency spectrum as derived from the Fourier Projection Slice Theorem. Therefore, reconstructing density volume slices at the resolution of the projection data, with subsequent upsampling of these to the viewport resolution will produce similar results, provided a decent interpolation filter is used to sufficiently suppress aliasing. An obvious consequence is faster rendering speed, since in-slice density upsampling is less expensive than density reconstruction on a finer grid. Classification and shading is performed on the high-resolution grid in both such pipelines.

4.4 GPU D^{2}VR Acceleration Methods

According to the theory of computed tomography, reconstructed voxel values are valid only for those voxels that fall into the "effective" reconstruction area, which is a (truncated) circle for a 2D (rectangle) square volume slice, or a (truncated) cylinder for a 3D cubic (rectangular solid) volume, assuming parallel beam reconstruction. Hence we can calculate the initial bounding volume and slice it according to the viewing direction. We then use these bounding volume slices as our depth buffer to guide the reconstruction, which essentially restricts the computation within the effective area throughout the whole reconstruction pipeline. A more sophisticated scheme would, as a preprocess, reconstruct a volume mask that would label all voxels that are in the shadow of all non-zero projection data. This occupancy mask could then be sliced, for each visualization frame, with the present slice configuration to drive the reconstruction more accurately than the bounding sphere. This is equivalent to empty space skipping, with the former method being a good approximation for many cases.

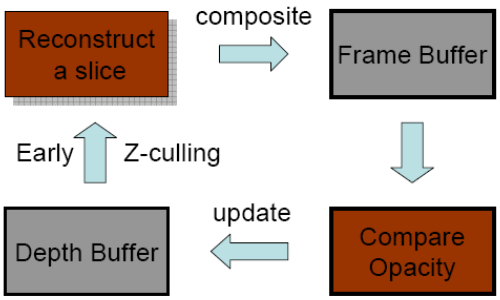

Figure 4: Rendering-driven occlusion culling

We shall now turn to early ray termination acceleration, aka occlusion culling (the mechanism for GPU-D^{2}VR is illustrated in Fig. 4). For this, we must reconstruct the slice stack in front-to-back order. When a new slice is computed from all projections, we composite it with the current frame buffer. We then examine every fragment's opacity value and compare it to the preset threshold, which usually varies between 0.0 to 1.0, depending on the rendering mode (iso-surface or full volume rendering). All those fragments whose opacity values exceed the threshold will be recorded to update the depth buffer. The updated depth buffer will then be used for the reconstruction of the next slice to prevent the GPU from generating fragments at those marked positions (which is the early z-buffer kill mechanism). The technique effectively eliminates the need for reconstructing voxels that do not contribute to rendering, hence it greatly reduces the effort consumed on the computational intensive component. For iso-surface rendering, where emitted rays are generally terminated early, this rendering-driven technique can achieve good speed-ups. Implementation-wise, the effect of early z-culling largely relies on how long the rendering of a fragment would take. Low computational effort ratio with respect to a single fragment could offset the advantage brought by early z-cull mechanism. Therefore, the spreading method that incorporates a longer fragment shader tends to benefit more from the above acceleration strategies, compared to the projective texture method, where texture mapping imposes relatively light rendering efforts on individual fragments.

5. Results

We experimented with the GPU-D^{2}VR framework on an AMD Athlon 2.2GHz dual core PC with 1GB RAM and a GeForce 6800 GT. Shaded images were rendered into either a 128^2 or a 256^2 viewport.

Timings for the different strategies described, using various CT datasets, are presented in Table 1. We observe a speedup of a factor 1.5 for density-only D^{2}VR over density+gradient D^{2}VR. The 256^3 size of volume could not be reconstructed with both density and gradients since it exceeds the current maximum size of the GPU memory. The

©The Eurographics Association 2006.

acceleration techniques mentioned in Section 4.4 also yielded a factor of 1.5 speedup, as compared to their respective basic GPU D^2VR counterparts.

A near interactive performance (2.5 fps) is obtained for volumes of size 128^3 when rendered into a 128^2 viewport. Rendering into a larger viewport of 256^2 with in-slice interpolation of 128^3 reconstructions (as described above) only decreases performance by a small amount (1.7 fps). On the other hand, rendering from a 256^3 reconstruction volume takes 4 times as much (0.6 fps).

The timings reported in [Rau05] always match the reconstruction volume resolution with the viewport resolution. Their CPU and GPU implementations take about 1453 s and 0.25 fps, respectively, for 128 128^2 projections and a 256^2 viewport. Our framework offers a fairly large speedup over these. The timings presented in Table 1 use the floating point pipeline. If we use the dual-channel 16-bit pipeline, as mentioned in Section 3, we can obtain another speedup of 2.

Rendering results are presented in Fig. 5. We produced images with the GPU D^2VR projection-based volume rendering with density only as well as with density+gradient renderings. The images are similar in quality than those reported in [RCG*06]. We show images rendered with all methods discussed in the paper, including one rendered with the 16-bit pipeline. We observe that the images rendered with D^2VR reconstructing the gradients in volume space seem to have higher feature definition on zooms than those where the gradients were backprojected. A theoretical justification for this was presented in Section 4.2. At the same time, the method is also more computationally efficient.

We also observe that reconstruction into a larger viewport indeed does not require a reconstruction into a volume grid of identical resolution. Fig. 5(i)(k) show that an upsampling on a slice-basis, followed by classification and shading produces very similar results (compare with Fig. 5(h)(j)). Finally, we also observe (in Fig. 5(g)) that the 16-bit pipeline produces high-quality images as well.

6. Conclusions

We have described an efficient GPU-based method to accelerate the D^2VR method. For this we have built on our GPU-accelerated CT framework and extended it in the following ways. First, we allowed the reconstruction of arbitrary oriented volumes, which is not needed for CT, but is necessary for D^2VR since we must generate a matrix of samples exactly aligned with the image plane, whose orientation is completely arbitrary. Second, we incorporated various acceleration techniques to limit the reconstruction effort to only the visible and non-occluded (the relevant) matrix samples. Our system enables framerates of up to 2 frames/s for realistic dataset sizes, which is 1-2 orders of magnitudes faster than the software solution. Next, we plan to extend the method to functional CT data, which reconstructs an emission volume, not a density volume, in which voxels "glow" on their own. We also plan to apply more accurate reconstruction schemes, such as the inverse Radon transform, and the D^2VR rendering directly from spiral CT data. Finally, in the spirit of D^2VR we also plan to volume-render MRI k-space data directly from the frequency domain.

Acknowledgements

This research was supported, in part, by NSF CAREER grant ACI-0093157 and NIH grant 5R21EB004099-02.

Dataset	Projections	Volume	Viewport	D^2VR	D^2VR-G	D^2VR+B	D^2VR+B+OC	Rasterized Voxels (%)	in Fig. 5
Foot (iso 1)	128×128^2	128^3	128^2	0.62	0.85	0.54	0.41 (2.4 fps)	37%	N/A
Foot (iso 1)	128×128^2	128^3	256^2	0.8	0.95	0.72	0.59 (1.7 fps)	37%	(e)(f)
Foot (iso 2)	128×128^2	128^3	256^2	0.8	1.0	0.72	0.66 (1.5 fps)	36.1%	(k)
Foot (iso 2)	128×128^2	256^3	256^2	2.3	N/A	2.0	1.67 (0.6 fps)	36.0%	(j)
Chapel Hill Head	128×128^2	128^3	256^2	0.8	1.1	0.94	0.7 (1.4 fps)	24.2%	(c)(d)
Toes	256×256^2	256^3	256^2	4.93	N/A	4.26	3.5 (0.3 fps)	31.6%	(l)

Table 1: Rendering performance in seconds for various datasets under different strategies: **D^2VR** is projection-based volume rendering with reconstruction of density only (GFS); **D^2VR-G** is the projection-based volume rendering with reconstruction of both density and gradient properties (GFP); **B** uses the bounding volume empty-space culling strategy, and **OC** uses occlusion culling. All of the above timings are measured with shading.

©The Eurographics Association 2006.

References

[BBT96] M. Bentum, B. Lichtenbelt, and T. Malzbender, "Frequency analysis of gradient estimators in volume rendering," *IEEE Transactions on Visualization and Computer Graphics,* 2(3):242-254, 1996.

[CCF94] B. Cabral, N. Cam and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 symposium on Volume visualization*, pages 91–98. ACM Press, 1994.

[CM03] K. Chidlow and T. Möller. Rapid emission tomography reconstruction. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, pages 15–26. ACM Press, 2003.

[EKE01] K. Engel, M. Kraus and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/ EUROGRAPHICS workshop on on Graphics hardware*, pages 9–16. ACM Press, 2001.

[FDK84] L. A. Feldkamp, L. Davis and J. W. Kress. Practical cone beam algorithm. *Journal of the Optical Society of America*, pages 612–619, 1984.

[Gra91] P. Grangeat. Mathematical framework of cone beam 3D reconstruction via the first derivative of the Radon transform. *Lecture notes in Mathematics*, pages 66–97, 1991.

[JRH*04] T. Jansen, B. von Rymon-Lipinski, N. Hanssen and E. Keeve. Fourier volume rendering on the GPU using a Split-Stream-FFT. *Proceedings of the Vision, Modeling, and Visualization Conference 2004*, pages 395–403, 2004.

[Kat04] A. I. Katsevich. An improved exact filtered backprojection algorithm for spiral computed tomography. *Adv. Appl. Math.*, 32(4):681–697, 2004.

[KS88] A. C. Kak and M. Slaney. *Principles of Computerized Tomographic Imaging*, IEEE Press, 1988.

[KW03] J. Krüeger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization 2003*, 2003.

[KND00] H. Kudo, F. Noo and M. Defrise. Quasi-exact filtered backprojection algorithm for long-object problem in helical cone-beam tomography. *IEEE Transactions of Medical Imaging*, 19(9):902–921, 2000.

[MX06] K. Mueller and F. Xu. Practical considerations for GPU-Accelerated CT. In *IEEE International Symposium on Biomedical Imaging*, 2006.

[MY00] K. Mueller and R. Yagel. Rapid 3-D cone-beam reconstruction with the simultaneous algebraic reconstruction technique (SART) using 2-D texture mapping hardware. *IEEE Transactions on Medical Imaging*, 19(12):1227–1237, 2000.

[NM05] N. Neophytou and K. Mueller. GPU accelerated image aligned splatting. In *Volume Graphics 2005*, pages 197–205, 2005.

[Rau05] P. Rautek. D^2VR high-quality volume rendering of projection-based volumetric data. *Master's Thesis*, 2005.

[RCG*06] P. Rautek, B. Csébfalvi, S. Grimm, S. Bruckner, E. Groller. D^2VR: High-quality volume rendering of projection-based volumetric data. In *Joint Eurographics - IEEE TCVG Symposium on Visualization*, 2006.

[RSE*00] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. In *Proceedings of the Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 109–118, 2000.

[SKv*92] M. Segal, C. Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, pages 249–252. ACM Press, 1992.

[SSE05] T. Klein S. Stegmaier, M. Strengert and T. Ertl. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Volume Graphics Workshop*, pages 187–195, 2005.

[TSS98] K. C. Tam, S. Samarasekera and F. Sauer. Exact cone beam CT with a spiral scan. *Physics in Medicine and Biology*, 43:1015–1024, 1998.

[TD00] H. Turbell and P.-E. Danielsson. Helical cone beam tomography. *International Journal of Imaging Systems and Technology*, 11:91–100, 2000.

[Tuy83] H. K. Tuy. An inversion formula for cone-beam reconstruction. *SIAM Journal on Applied Mathematics*, 43:546–552, 1983.

[XM05] F. Xu and K. Mueller. Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware. *IEEE Transactions on Nuclear Science*, 52(3):654–663, 2005.

©The Eurographics Association 2006.

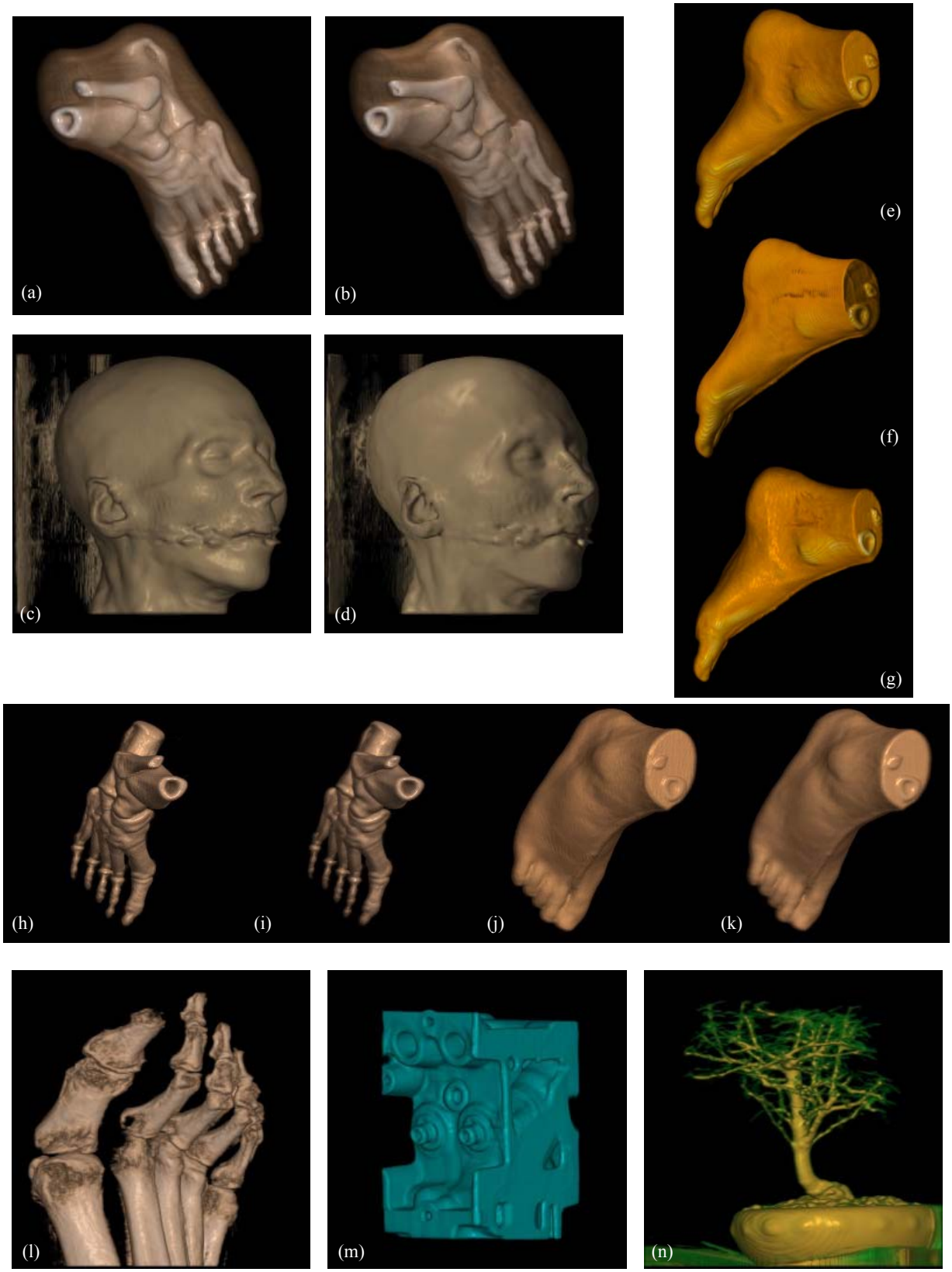

Figure 5: Rendering results: D^2VR+GFS (a, c, e, g, h-n); D^2VR+GFP (b, d, f); (g) is rendered from 16-bit pipeline; (h, j) are rendered from matched volume and viewport resolution, and (i, k) are rendered by upsampling on reconstructed volume slices.