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

Adaptive Sampling in Single Pass, GPU-based Raycasting of Multiresolution Volumes

Patric Ljung†

Visual Information Technology and Applications, Linköping University

Abstract

This paper presents a novel direct volume rendering technique for adaptive object- and image-space sampling density of multiresolution volumes. The raycasting is implemented entirely on the GPU in a single pass fragment program which adapts the sampling density along rays, guided by block resolutions. The multiresolution volumes are provided by a transfer function based level-of-detail scheme adaptively loading large out-of-core volumes. Adaptive image-space sampling is achieved by gathering projected basic volume block statistics for screen tiles and then allocating a level-of-detail for each tile. This combination of techniques provides a significant reduction of processing requirements while maintaining high quality rendering.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Viewing algorithms; I.4.10 [Image Processing and Computer Vision]: Volumetric;

1. Introduction

The rendering of gigabyte sized volumes on desktop PCs is a highly desired but challenging task. Interactive exploration and analysis of these data sets calls for methods that are not restricted to a pre-determined set of features being used to compress the data. In medicine, in particular, these data sets cannot be compressed using lossy methods and the original data must be retrievable. Consequently, in cases where computing and memory resources are not abundant, these data sets have to be dealt with using techniques thatcan , at run-time, optimize the level-of-detail (LOD) with respect to available resources and the current task. One way to determine the current task is by using a transfer function based LOD selection that provides dynamic loading of blocks at varying resolutions depending on their contribution to the final image. The definition of available resources typically refers to the amount of texture memory on the GPU.

The work presented in this paper is based on a flat blocking multiresolution LOD volume scheme [LLYM04] and it is shown how the blockwise LOD can guide the sampling density along rays in the volume raycasting. When a ray passes through a low resolution block the sampling density is decreased and when it passes through a high resolution block it is increased. This technique thus constitutes a more adaptive sampling strategy compared to the discrete nature of space-leaping approaches. The method is implemented entirely on the GPU as a single-pass raycasting fragment program. In addition to the adaptive object-space sampling the paper presents an adaptive image-space sampling technique where higher frame rates are achieved by rendering tiles of the screen at varying resolutions rather than lowering the image-space resolution uniformly. Tiles with rays intersecting high resolution volume blocks will be given a higher priority and rendered at higher resolution than tiles having rays intersecting only low resolution blocks.

Casting rays on the GPU in single pass programs is highly desired since it improves the precision of the volume integral computation compared to texture slicing techniques where framebuffer precision is usually limited to 8 or 16 bits per component. High quality rendering of large data sets often requires many hundreds to thousands of composited slices generating a huge framebuffer bandwidth demand.

This paper is organized as follows. Section 2 reviews related work and section 3 then presents the object-space adaptive sampling. Section 4 describes the adaptive sampling

† plg@itn.liu.se

scheme in image-space. Results and benchmarks are presented in section 5 and section 6 presents our conclusions and future work directions.

2. Related Work

Multiresolution volume representations of regular volumes have been proposed by several researchers. Hierarchical schemes, so called octrees, have been presented and used [LHJ99, WWH*00, BNS01, GWGS02]. If a volume block in the hierarchy is found to be of inadequate resolution it is replaced by eight 'children', all having the same number of samples but with each covering 1/8 of the spatial volume of its parent. Flat blocking schemes have also been proposed and used [IP99, BIP01, NS01, LLYM04]. Here each block occupies a constant spatial domain and, instead, its content is replaced with more or fewer samples, also usually with a factor of 8.

Several measures and schemes for determining the LOD for block resolutions have been proposed. A strictly view-dependent criterion, distance to viewer, was proposed by LaMar et al. [LHJ99] and Weiler et al. [WWH*00]. Data errors between resolution levels was presented by Boada et al. [BNS01] and Guthe et al. [GWGS02]. LaMar et al. [LHJ03] use frequency tables and derive a grey scale Transfer Function (TF) error between resolution levels. Guthe and Strasser [GS04] approximate the image-space error by means of maximum voxel deviation and overestimation of a TF-based difference in the RGB-channels. Gao et al. [GHSK03] presented a scheme using a Plenoptic Opacity Function (POF) for each block, based on a set of TF basis functions. Ljung et al. [LLYM04] introduced a simplified histogram representation and estimate a post-TF maximum error between the lowest resolution and the simplified histogram, representing the full resolution block. The error is derived in the perceptually adapted CIELUV color space. A coarse value histogram has also been used by Gao et al. [GHJA05].

Rendering of multiresolution volumes on graphics hardware has mostly been accomplished by means of texture slicing [CCF94] in the papers referenced above. Based on view-dependent LOD, decreased sampling density along the viewing rays is provided by increased interslice distance for more distant slices [LHJ99, WWH*00]. This approach has also been taken by Guthe et al. [GWGS02, GS04] where the block resolution dictates the interslice distance. The issue of interpolation of samples between blocks in hierarchical schemes has been addressed by voxel replication and replacement, where a single-sided replication only requires a 20% increase of the reduced data size, using 16^3 voxel blocks. For flat blocking schemes this overhead is increased significantly and this issue has been addressed in Ljung et al. [LLY06] by introducing a direct interblock interpolation scheme that does not require any voxel replication and which interpolates smoothly between blocks of arbitrary resolutions. Since a flat blocking scheme provides a simple block structure it is easier to access arbitrary parts of the volume by means of a uniform lookup scheme and the packed LOD volume and block meta-data can be stored in just two textures.

Krüger and Westermann [KW03] presented a multipass raycasting scheme for regular volumes where a fixed number of ray steps are executed in each pass. Stegmaier et al. [SSKE05] presented a GPU-based technique for rendering regular uniform volumes using a raycasting single-pass fragment program. This provides for improved visual quality, in terms of computational precision, and a single-pass approach provides a more flexible framework for advanced rendering features such as reflection mapping and refraction.

There is a large amount of work devoted to adaptive image-space sampling, particularly in the ray tracing domain. Adamson et al. [AAN05] describe the concepts of progressive refinement of the entire image, adaptive refinement of local areas with greater visual error/variation, and spatial/temporal coherence using previous and neighboring information to speed up image generation. Adaptive refinement was applied to volume rendering by Levoy [Lev90] and Klein et al. [KSSE05] presented spatial/temporal coherence techniques for single-pass raycasting on the GPU, employing empty space leaping and reprojection [GR90, YS93]. Space skipping is also presented in [KW03] by means of skipping rays in passes where they encounter empty space.

3. Adaptive Object-Space Sampling

This section presents the adaptive object-space sampling of multiresolution volumes. First the structure of the multiresolution volumes are described since it forms the basic structure for the renderer to work with. Then the details of the single-pass raycaster and how adaptive sampling is achieved are presented.

3.1. Multiresolution Volumes

The multiresolution scheme used in this work is based on a flat blocking scheme using a transfer function based LOD selection technique [LLYM04]. The blocking scheme divides the volume into uniform blocks, in this case consisting of 16^3 voxels. The spatial dimensions of each block are constant, independent of the resolution level at which a block is selected. Compression of the data sets has not been employed since medical data sets are primarily being used; instead the data sets are stored block-wise at all resolution levels (in powers of two) which requires a small increase, just 14%, in the size of the volume data.

For a given memory budget, such that a multiresolution volume fits in the memory of the GPU, the LOD selection optimizes the block resolutions to minimize an estimated error based on the perceptually adapted CIELUV

color space error, ΔE [Poy97, Fai98]. The loader then constructs a packed texture of all the blocks at varying resolution levels and an index texture, holding block locations and sizes in the packed texture, in a manner similar to the Adaptive Texture Maps in [KE02]. When the TF is changed a new LOD selection is performed and blocks with changed resolution are read from disk.

3.2. Multiresolution Raycasting

GPU-based single-pass raycasting refers to executing the entire ray traversal in a single instance of a fragment program, in contrast to texture slicing where each pass only computes the result for a single sample or a ray segment. In [SSKE05, KSSE05] single-pass approaches for rendering of uniform, regular volumes are presented. For clarity in this description some details will be provided here as well.

The raycaster initializes the rays by rendering the volume bounding box, culling back faces, and providing texture and geometry coordinates at the vertices. These attributes will then be interpolated over the surfaces and be readily available to the fragment program as ray entry points in texture coordinates, $\boldsymbol{x}_0$, and geometric coordinates, $\boldsymbol{p}_0$. The camera location, $\boldsymbol{c}$, is retrieved from the last row of the inverse, transposed model view matrix such that a per-fragment correct view direction can be determined. In addition, the program needs the minimum and maximum texture coordinates in order to determine the full ray length through the volume, required to conditionally terminate the program. Furthermore a set of additional parameters are required, including the sampling step-length, Δ, and the geometry to texture coordinate transform matrix, $\mathbf{T}$. The ray step vector, $\boldsymbol{d}$, in texture coordinate space is then defined by

$$\boldsymbol{d} = \mathbf{T} \cdot \Delta ||\boldsymbol{p}_0 - \boldsymbol{c}||. \quad (1)$$

A ray, $\boldsymbol{r}$, through the volume is then denoted as

$$\boldsymbol{r}(\lambda) = \boldsymbol{x}_0 + \lambda \boldsymbol{d}, \quad (2)$$

where $\boldsymbol{x}_0$ is the ray entry point and $\boldsymbol{d}$ is the ray direction. A set of discrete sample points, $\boldsymbol{x}_k$, is then defined using $\lambda = k$, $k \in \mathbf{N}$. For each loop in the raycasting shader program the ray position is increased by $\eta = 1$, such that $k_{i+1} = k_i + \eta$. (η will be used later).

Extending the volume sampling of regular volumes to multiresolution volumes requires the introduction of a texture indirection with lookup of block location and scale in an index texture. The sample volume now contains blocks at different resolution levels tightly packed in a single texture object. Figure 1 illustrates a set of rays passing through a cross-section of a flat multiresolution volume. The black dots along the rays in the figure indicate the uniform discrete sampling points, $\boldsymbol{x}_k$. In order to retrieve a sample for a position, $\boldsymbol{x}$, the block index and intrablock position, $\boldsymbol{x}'$, are needed. Using the number of blocks, (N_x, N_y, N_z), as the coordinate domain the intrablock position is conveniently computed as $\boldsymbol{x}' = \text{frac}(\boldsymbol{x})$. The block index is given by $\lfloor \boldsymbol{x} \rfloor$, in practice achieved through nearest neighbor lookup. From the index texture the block location in the packed sample texture, $\boldsymbol{q}$, and the scale, κ, are retrieved. The block scale is encoded as $\kappa = \sigma / \sigma_{\max}$, where σ is the block size, the number of voxels along each dimension and $\sigma_{\max}$ is the maximum block size ($\sigma_{\max} = 16$).

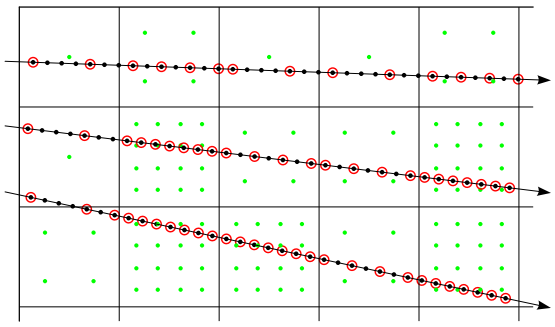


Figure 1: *A set of rays intersecting a cross-section of a volume. Black points illustrate uniform sampling along a ray. Adaptive sample locations are indicated by red circles.*

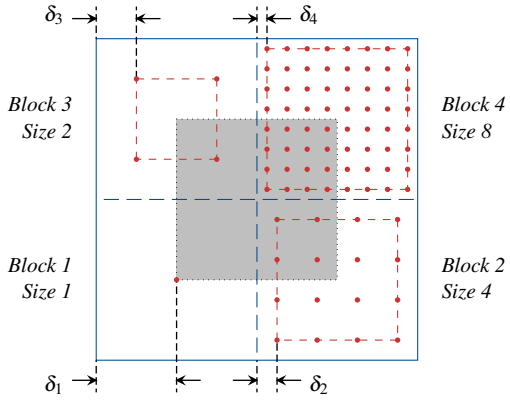


Figure 2: *Four blocks in a neighborhood. Sample values within each tile are indicated by red points. A sample boundary, the red dashed lines, mark the domain for texture coordinates within each block. The distances from the sample boundary to the block borders are indicated with δ_t for each block, t. The grey area between block centers indicate the domain for interblock interpolation.*

Figure 2 illustrates, in 2D, a neighborhood of four blocks, for each block the sample boundary is shown as red dashed lines, exactly enclosing the sample data of a block. It is obvious that sampling will be needed outside of the sample boundaries between blocks and these blocks are not necessarily neighbors in the packed volume and, further, may be of different resolutions, as illustrated in the figure. Only in-

trablock volume sampling is considered in this paper and the sample location must therefore be clamped to the block's sample boundary, $[\delta, 1-\delta]$, where $\delta = 1/2\sigma$. The final texture coordinate, $\boldsymbol{u}$, for an intrablock sample is then given by

$$\boldsymbol{u} = \mathbf{S}\kappa\mathbf{C}_{\delta}^{1-\delta}(\boldsymbol{x}') + \boldsymbol{q}, \tag{3}$$

where $\mathbf{S}$ is the packed texture scaling matrix and $\mathbf{C}_a^b(x)$ clamps the value x to the range $[a,b]$.

3.3. Adaptive Sampling of Multiresolution Volumes

The red circles in figure 1 indicate the actual samples taken for adaptive sampling. In essence, a low sample density is desired in reduced resolution blocks and a high sample density in full resolution blocks. In a naive approach, the inverse of the current block scale, κ, could be used to advance the ray with $\eta = 1/\kappa = \sigma_{\max}/\sigma$ steps. Consider entering, however, a low resolution block corner, the ray could then advance deep into a consecutive, potentially, high resolution block. Therefore, limiting the stepping so as to not advance beyond the first discrete sample point in the following block seems appropriate. Stepping by a discrete number of intervals enables the use of a compact 2D TF, using the step increment, η, for the second dimension where the TF opacity and color weighting are adjusted accordingly. In this case 16 different step lengths are used, defined by the maximum block size $\sigma_{\max}$.

In order to limit the stepping along the ray, the length of the ray inside a block must be determined. In general, the ray length, l, from a position, $\boldsymbol{x}$, to the boundary of a box is given by

$$l = \min\left(\frac{\mu_x - x_x}{d_x}, \frac{\mu_y - x_y}{d_y}, \frac{\mu_z - x_z}{d_z}\right), \tag{4}$$

where

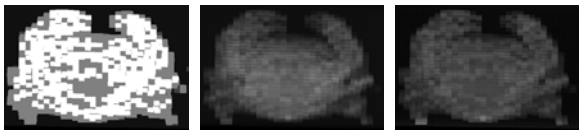
$$\mu_\rho = \begin{cases} x_{\min,\rho}, & \text{if} \quad d_\rho < 0 \\ x_{\max,\rho}, & \text{if} \quad d_\rho \geq 0. \end{cases},$$

$\boldsymbol{x}_{\min}$ and $\boldsymbol{x}_{\max}$ defining the box. Starting from an intrablock position, $\boldsymbol{x}'$, and moving along ray direction, $\boldsymbol{d}$, (4) is used with $\boldsymbol{x}_{\min} = (0,0,0)$ and $\boldsymbol{x}_{\max} = (1,1,1)$ to determine the remaining ray length inside a block. The number of steps to take then becomes $\eta = \min(1 + \lfloor l \rfloor, 1/\kappa)$.

4. Image-Space Adaptive Sampling

The goal of image-space adaptive sampling is to reduce the number of evaluated rays in the rendering while maintaining image quality such that the rendering time can be reduced in a controllable way, thus providing better interactivity. In addition it should be suitable for implementation using graphics hardware and support efficient reconstruction of the final image. A tile-based approach is therefore proposed such that the resolution of each tile can be adapted, in a single rendering pass or over time.

The framebuffer is divided into a number of equally large tiles, denoted with the tilesize $\tau_{\max}$. Each tile is given a specific LOD, τ, and rendered at that resolution level. Resolutions are not restricted to be in powers of two. The final image is reconstructed using an intertile interpolating shader, an adaptation to 2D of the technique presented in [LLY06].



a) Max *b) Average* *c) RMS*

Figure 3: *Example of low resolution block statistics renderings using the LOD priority schemes Max, Average, and RMS of block resolutions (κ). See also figures 8 and 10.*

4.1. Level-of-Detail Selection in Image-Space

The multiresolution volume being rendered already holds important information about resolution levels. By using the raycasting framework the index texture can be rendered and block resolution information is collected from all blocks intersected by the ray. To make this rendering faster, besides rendering at low resolution, only one sample per block is taken by advancing the ray by $\eta = \lfloor l \rfloor + 1$ steps from (4) using the discretized block entry point, $\boldsymbol{x}'$. In this rendering the maximum resolution is collected as the block scale, κ, of blocks along the ray. The average value of block scales, the root-mean-square (RMS), and the number of blocks encountered are also gathered. This information is stored in the framebuffer and copied back for tile priority classification. This image is rendered at a resolution corresponding to the number of tiles plus one, along x and y-directions, to sample the corners of each tile. The following tile priority schemes have been defined:

Max Set the tile priority to the maximum value of the Max value-samples of the tile corners. This is a conservative scheme and can potentially dictate a high priority for all tiles.

Average Set the tile priority to the average of the Average value-samples of the tile corners. The average scheme gives higher priority to tiles hitting contours, since the number of high resolution blocks is higher along such rays.

RMS Set the tile priority to the RMS value of the RMS value- samples of the tile corners. This is intended to emphasize high resolution blocks.

In figure 3 example renderings of the block statistics are shown for the crab data set, the final results can be seen in figure 8 and 10.

The tile priorities are also recorded in a small array. The

number of slots is defined by the full resolution volume block size, $\sigma_{\max}$. This provides a compact description of the tile priority distribution which is used to compute the tile size allocation. Tiles none of whose corners intersect the volume can be discarded. These tiles have a priority of zero. An upper bound for tile sizes, Q_{H}, is used which is assigned to tiles in the highest non-zero histogram slot. The tile priority value is linearly interpolated between this value and a user specified lower bound, Q_{L}. This enables experimentation with the different schemes in a simple and robust way.

4.2. Tile Rendering and Image Reconstruction

Given the LOD selection from the process described above the tiles are rendered according to their assigned size. This is achieved by changing the viewport and projection matrix in OpenGL and then rendering the volume bounding box. This procedure is repeated for all tiles. The framebuffer now contains an image which looks like a mosaic, see top image in figure 4. Since the tiles in the framebuffer are not tightly packed, it is also possible to update individual tiles and reconstruct incrementally improved images without having to re-render the entire scene. This is, however, left for future work.

Changing the viewport and projection matrix per tile might be costly and a second tile rendering technique was therefore developed. This method begins with rendering the volume bounding box and uses a framebuffer object to store texture entry points, $\boldsymbol{x}_0$, and view direction for each pixel, similar to [KW03]. In a second pass polygons of size τ_t are rendered for each tile t, mapping the full tile size, $\tau_{\max}$, in the previously rendered framebuffer object.

As the last step of this adaptive image-space sampling technique the final image must be reconstructed from the mosaic image. To this end the 3D interblock interpolation from [LLY06] has been adapted to 2D and renamed intertile interpolation, briefly described here. Figure 2 illustrates a neighborhood of four tiles and a sample, φ, lying somewhere (in the grey area) between the tile centers needs to be computed. A sample, φ_t, from each tile, t, is taken using texture coordinate clamping (3). A local intertile coordinate, $\boldsymbol{x}^* = \mathrm{frac}(\boldsymbol{x}+0.5)-0.5$, is then used to compute edge weights, $e_{i,j}$, between tiles, i and j, sharing sides.

$$e_{i,j}(\rho) = \mathbf{C}_0^1((\rho+\delta_i)/(\delta_i+\delta_j)), \tag{5}$$

where ρ denotes either x_x^* or x_y^* and $\delta_t = 1/2\tau_t$. The sample, φ, is then computed as the normalized sum

$$\varphi = \frac{\sum_{t=1}^{4} \omega_t \varphi_t}{\sum_{t=1}^{4} \omega_t}, \tag{6}$$

with the tile weights, ω_t, defined as

$$\begin{aligned} \omega_1 &= (1-e_{1,2})\cdot(1-e_{1,3}), \\ \omega_2 &= e_{1,2}\cdot(1-e_{2,4}), \\ \omega_3 &= (1-e_{3,4})\cdot e_{1,3}, \\ \omega_4 &= e_{3,4}\cdot e_{2,4}. \end{aligned}$$

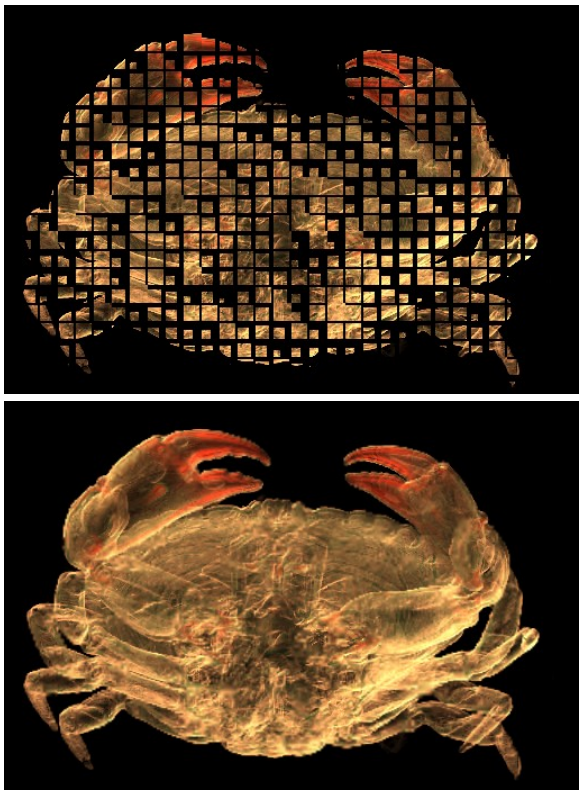
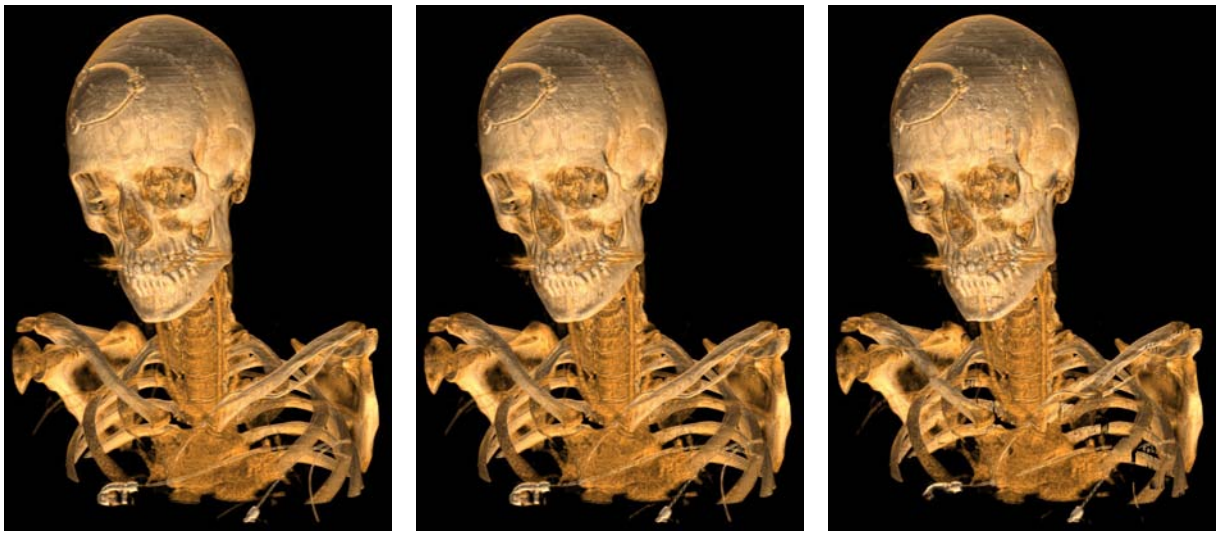


Figure 4: *The Crab dataset rendered using random screen-space adaptive sampling. Top image shows the original framebuffer with the rendered tiles. Bottom shows the reconstructed image using intertile interpolation.*

5. Results

This section presents the results and analysis of the methods presented in this paper. All tests were performed on an AMD AthlonX2 64 using an ATI X1800XT GPU (ATI) with 512 MB of memory and a Pentium 4 using an NVidia GeForce 7800GTX GPU (NV) with 256 MB of memory. All data sets are stored in 16 bit integers and the gradients are precomputed using a Sobel operator. Gradient data is stored using four bytes, the first three holding the normalized gradient direction and the fourth holding the square-root of the gradient length. The threaded run-time data loader updates the data set in a fraction of a second for smaller changes in the TF or for changed cropping. A complete change of TF may require a few seconds of loading in the process background. The target memory budget for the data reduction using the TF-based LOD selection is typically 96 MB, including gradient data. The data sets used are listed in table 1. Shaders have been implemented in the OpenGL Shading Language and the OpenGL ARB Fragment Program extension.



a) Full (5.1 FPS) *b) Limited (14.1 FPS)* *c) Naive (19.8 FPS)*

Figure 5: *Rendering the female data set with adaptive sampling along the ray. Left image shows full uniform sampling, middle image shows adaptive stepping with block boundary limits and right image shows naive stepping. The naive approach suffer from visual errors due to view-dependent under-sampling. Images were rendered in a 1024×1024 viewport. The data reduction is 11:1 through the run-time TF-based LOD-selection.*

Table 1: *All data sets are stored in 16 bit integer precision with precomputed gradients in four bytes. Size indicate original and gradient data, and size on disk include all resolution levels.*

Data Set	Dimensions	Size	Size on disk
Female	512×512×628	942 MB	1.3 GB
Crab	512×512×512	768 MB	877 MB

Table 2: *Rendering performance of the adaptive object-space sampling in a 1024×1024 viewport. Data reductions are given for each data set. TS refers to texture slicing and is comparable with the full single-pass raycasting. Numbers specify FPS (speed-up vs. full sampling).*

Dataset	GPU	TS	Full	Limited	Naive
Female 11:1	ATI	5.3	5.1	14.1 (2.8)	19.8 (3.9)
Female 11:1	NV	3.6	2.3	6.0 (2.6)	9.7 (4.2)
Female 30:1	ATI	5.5	5.2	17.4 (3.3)	27.5 (5.3)
Female 30:1	NV	3.6	2.3	6.5 (2.8)	11.1 (4.8)
Crab 9:1	ATI	3.4	2.9	6.2 (2.1)	8.7 (3.0)

5.1. Adaptive Object-Space Sampling Performance

Previous presented GPU-based raycasting results [KW03, SSKE05, KSSE05] indicate a performance drop of a factor about 2, compared with traditional texture slicing without optimizations. This ratio is also confirmed for the multiresolution case on the GF7800 but not for the X1800, where it performs equally well as its texture slicing counterpart. For a data reduction factor of g it can be expected that the adaptive sampling along the ray will yield a $g^{\frac{1}{3}}$ speed-up since sampling is adapted in only one dimension. From the performance results in table 2 it can be seen that this expected speed-up is achieved by the limited sampling, a data reduction 11:1, 30:1 and 9:1 yield expected speed-ups of 2.2, 3.1 and 2.1, respectively. The frame-rates are further increased by the naive scheme, but then visible errors are introduced. Renderings of the female data set, at a data reduction of 11:1, are shown in figures 5 and 9.

5.2. Adaptive Image-Space Sampling Performance

Volume rendering is clearly output-sensitive and typically scales linearly with the number of pixels being processed. The goal of the tile-based rendering is to reduce the overall processing requirement by rendering tiles at varying resolutions. For an ideal situation the performance gain should scale with the reduction of the number of pixels. The initial concern is therefore the overhead cost of the imposed tile-rendering approach. A simple surface shader was first used to measure performance versus tile size (τ_{max}) and reducing rendered tile size (τ), having τ_{max} constant. The results are shown in figure 6 and 7. Figure 6 clearly shows that decreased tile size decreases the frame rate and the overhead of the viewport method is thus not suitable for small tiles, preventing fine-grained LOD adaption. Polygon based tile

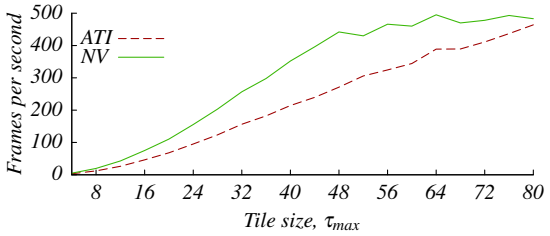


Figure 6: *Benchmark of tile managed rendering. A simple, non-volume rendering shader was used. The performance in terms of framerate is plotted against the tile size (τ_{max}). Tiles are rendered using the viewport method.*

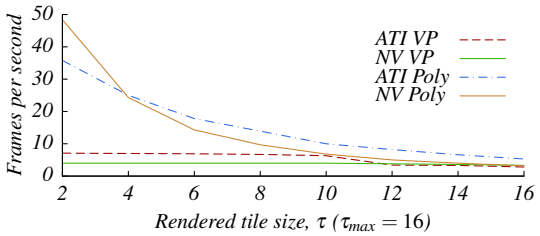


Figure 7: *Performance of uniformly reducing the rendered tile size, τ using a constant tiling of $\tau_{max} = 16$ in a 1024×1024 viewport. VP refers to the viewport method and Poly refers to the polygon method. The viewport method has significant scaling issues.*

rendering for the simple surface shader is constantly high for tile sizes over 12 pixels, about 800 FPS for NVidia and 160 FPS for ATI. The lower frame rate on ATI is related to the use of the FBO. The polygon approach shows a significant speed-up when reducing the tile size, τ, holding the maximum tile size constant, as can be seen in figure 7. The performance is, however, not scaling linearly with the number of rendered pixels as intended.

Examples of the three presented methods for screen-space LOD selection, Max, Average, and RMS are shown in figure 8. The Max selection is the most conservative and in order to meet a target frame-rate the highest allowed tile quality, Q_{H}, must be reduced, whereas the Average and RMS methods have shown to yield an expected larger variation between resolutions. The RMS method selects slightly higher resolutions at object boundaries in the volumes.

6. Conclusions

In this paper we have presented a single pass, GPU-based raycasting scheme for multiresolution volume datasets. A flat blocking structure are used for the volumes which provides a simple and homogeneous access scheme that can effectively be implemented in a fragment program. The result of adapting the sampling density to the surrounding resolution shows a significant increase in performance that scales with the data reduction, g, as $g^{\frac{1}{3}}$ without sacrificing visual quality.

The image-space adaptive LOD rendering also presents significant potential speed-up. The number of rendered pixels can be significantly reduced while maintaining high quality renderings. The current implementation used on the tested GPUs, however, still shows that some overhead involved with the use of framebuffer objects needs to be reduced. Future research directions include the development of improved screen-space LOD selection schemes and the integration of interframe coherence strategies improved interactive work with multiresolution volume rendering.

Acknowledgements

This work has been funded by the the Swedish Research Council, grant 621-2001-2778 and 621-2003-6582, and the Swedish Foundation for Strategic Research, grant A3 02:116. The Center for Medical Image Science and Visualization (CMIV) and, in particular, Anders Persson and Petter Quick is acknowledged for the Crab data set. The female data set is provided by Siemens. Many thanks to professor Anders Ynnerman for helpful discussions.

References

[AAN05] ADAMSON A., ALEXA M., NEALEN A.: Adaptive sampling of intersectable models exploiting image and object-space coherence. In *Proceedings SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2005), pp. 171–178.

[BIP01] BAJAJ C., IHM I., PARK S.: 3D RGB image compression for interactive applications. *ACM Transactions on Graphics 20*, 1 (January 2001), 10–38.

[BNS01] BOADA I., NAVAZO I., SCOPIGNO R.: Multiresolution volume visualization with a texture-based octree. *The Visual Computer 17* (2001), 185–197.

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization* (New York, NY, USA, 1994), ACM Press, pp. 91–98.

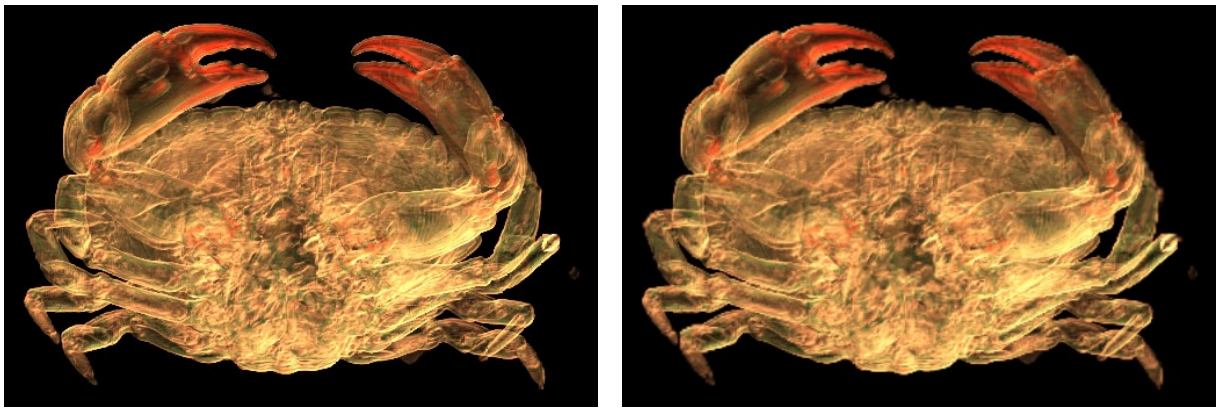
[Fai98] FAIRCHILD M. D.: *Color Appearance Models*. Addison Wesley Longman, Inc., 1998.

[GHJA05] GAO J., HUANG J., JOHNSON C. R., ATCHLEY S.: Distributed data management for large volume visualization. In *Proceedings IEEE Visualization 2005* (2005), IEEE, pp. 183–189.

[GHSK03] GAO J., HUANG J., SHEN H.-W., KOHL J. A.: Visibility culling using plenoptic opacity functions for large volume visualization. In *Proceedings IEEE Visualization 2003* (2003), IEEE, pp. 341–348.

[GR90] GUDMUNDSSON B., RANDÉN M.: Incremental generation of projections of CT-volumes. In *Proceedings of The First Conference on Visualization in Biomedical Computing* (1990), pp. 27–34.

[GS04] GUTHE S., STRASSER W.: Advanced techniques for high quality multiresolution volume rendering. In *Computers & Graphics* (2004), vol. 28, Elsevier Science, pp. 51–58.



a) Original: 8.7 FPS, 100% *b) Max: 14.1 FPS, 37%*

Figure 8: *Comparing the crab rendered at full resolution (a) with a rendering using the Max LOD selection and the high quality threshold $Q_H = 0.9$, thus rendering only 37% of the pixels (b). The polygon based tile rendering method is used in a 512×512 viewport using a tile size of 16 pixels. The image in b has been reconstructed using the intertile interpolation technique.*

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive rendering of large volume data sets. In *Proceedings IEEE Visualization 2002* (2002), pp. 53–60.

[IP99] IHM I., PARK S.: Wavelet-based 3d compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum 18*, 1 (1999), 3–15.

[KE02] KRAUS M., ERTL T.: Adaptive texture maps. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware* (2002), pp. 7–15.

[KSSE05] KLEIN T., STRENGERT M., STEGMAIER S., ERTL T.: Exploiting frame-to-frame coherence for accelerating high-quality volume raycasting on graphics hardware. In *Proceedings IEEE Visualization 2005* (2005), pp. 223–230.

[KW03] KRÜGER J., WESTERMANN R.: Acceleration techniques for gpu-based volume rendering. In *Proceedings IEEE Visualization 2003* (2003), pp. 287–292.

[Lev90] LEVOY M.: Volume rendering by adaptive refinement. *The Visual Computer 6* (1990), 2–7.

[LHJ99] LAMAR E. C., HAMANN B., JOY K. I.: Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings IEEE Visualization 1999* (1999), pp. 355–362.

[LHJ03] LAMAR E. C., HAMANN B., JOY K. I.: Efficient error calculation for multiresolution texture-based volume visualization. In *Hierachical and Geometrical Methods in Scientific Visualization* (2003), Springer-Verlag, pp. 51–62.

[LLY06] LJUNG P., LUNDSTRÖM C., YNNERMAN A.: Multiresolution interblock interpolation in direct volume rendering. In *Proceedings Eurographics/IEEE Symposium on Visualization 2006* (2006), pp. 259–266.

[LLYM04] LJUNG P., LUNDSTRÖM C., YNNERMAN A., MUSETH K.: Transfer function based adaptive decompression for volume rendering of large medical data sets. In *Proceedings IEEE Volume Visualization and Graphics Symposium 2004* (2004), pp. 25–32.

[NS01] NGUYEN K. G., SAUPE D.: Rapid high quality compression of volume data for visualization. *Computer Graphics Forum 20*, 3 (2001).

[Poy97] POYNTON C.: Frequently asked questions about color. http://www.poynton.com/PDFs/ColorFAQ.pdf, March 1997. Acquired January 2004.

[SSKE05] STEGMAIER S., STRENGERT M., KLEIN T., ERTL T.: A simple and flexible volume rendering framework for graphics-hardware–based raycasting. In *Eurographics/IEEE Volume Graphics Symposium* (2005), Eurographics.

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMAN K., ERTL T.: Level–of–detail volume rendering via 3d textures. In *Proceedings IEEE Volume Visualization and Graphics Symposium 2000* (2000), ACM Press, pp. 7–13.

[YS93] YAGEL R., SHI Z.: Accelerating volume animation by space-leaping. In *Proceedings of IEEE Visualization '93* (1993), pp. 62–69.