Time-Critical Ray-Cast Direct Volume Rendering

Andrew Corcoran*
John Dingliana†
Trinity College Dublin

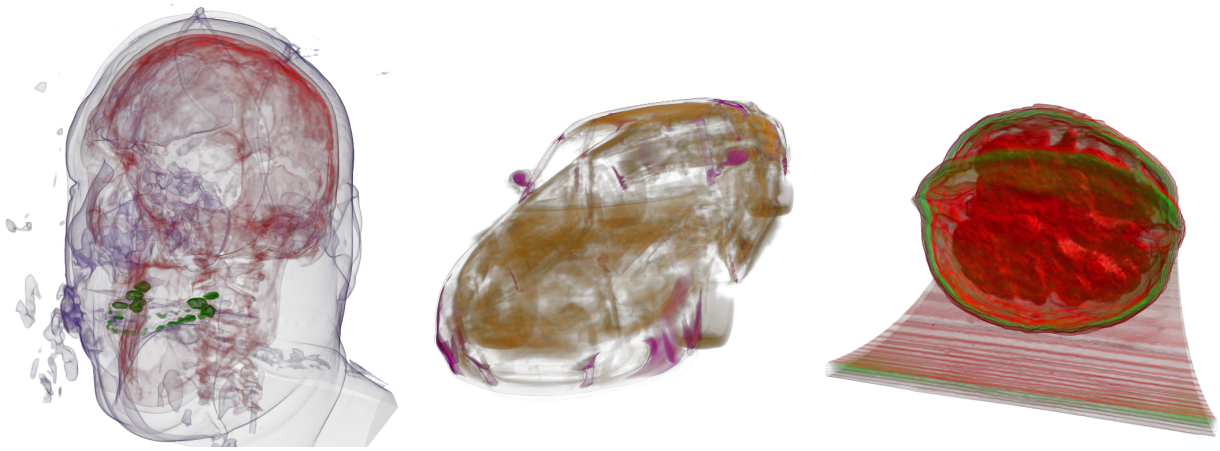

Figure 1: *Sample time-critical renderings of the vismale, porsche and walnut datasets.*

Abstract

We present a system for time-critical ray-cast direct volume rendering which can be easily integrated into existing acceleration techniques. Our system modifies the global sampling rate of the scene based on knowledge of past frame rates and quickly and robustly converges on a user specified frame rate while requiring no overhead to implement. We have tested our technique on a wide variety of datasets and our system quickly adapts to any changes in scene complexity and transfer function and dramatically minimises the large change in frame rates that traditionally occur due to user navigation of a complex volume dataset.

Keywords: volume rendering, dvr, time-critical

1 Introduction

The size of volume datasets continues to grow and to date graphics hardware has failed to keep pace with this increased demand for rendering power, indeed we have reached a stage where some of the larger volume datasets can no longer be resident in GPU memory due to their size. Interactive rendering of high quality volume datasets is a challenging problem and increasingly researchers have investigated methods to accelerate the speed of volume rendering. Methods that have been explored include, dataset compression [Ljung et al. 2004], early ray termination [Kruger and Westermann 2003], volume bricking [Hadwiger et al. 2005] and many more. The majority of the methods investigated rely on the user manually changing some quality metric in order to obtain the desired frame rate. The value of this quality metric can change from dataset to dataset and even from frame to frame and in order to obtain a consistent frame rate the user is required to continually update the quality metric.

We propose a method for dynamically changing the global sampling rate of ray-cast volume rendering in order to achieve consistently interactive frame rates for all volumes and all views without requiring user interaction. Our method can implemented within a standard volume rendering pipeline and can therefore be easily integrated with many of the already proposed acceleration methodologies.

Our method compares the sampling rate and the rendering time of the previous frame and calculates the optimal sampling rate for the next frame in order to achieve the target frame rate which allows us to rapidly increase and reduce the sampling rate in order to quickly achieve the target frame rate.

2 Related Work

Numerous approaches exist for volume visualisation which range from the very accurate to the very interactive [Chen et al. 2001; Preim and Bartz 2007]. The gap between the interactive and the accurate has been maintained despite increases in computation power due to parallel advances in high resolution data capture. While some approaches to volume rendering first obtain a surface representation from the volume before applying standard 3D renderings, direct volume rendering (DVR) techniques operate directly on the volume data. This can be achieved through techniques such as volume ray-casting [Hadwiger et al. 2005; Kruger and Westermann 2003], facilitated through adaptive optimisations for real-time visualisation of implicit surfaces defined by a volumetric grid of sample data. Other approaches describe techniques such as the use of 3D textures, generated from view aligned slices of volume data and optimised using GPU hardware for real-time performance [Cullip and Neumann 1994; Cabral et al. 1994].

Although adaptive sampling has been widely explored in rendering and volume visualisation, it has traditionally been used to vary the sampling detail across a scene based on some local measure of importance. Our strategy of on-demand modulation of the sampling rate to achieve target framerates is most closely related to techniques in dynamic simulation, which deal with adaptive time-

*e-mail: corcora1@cs.tcd.ie
†e-mail: John.Dingliana@scss.tcd.ie

stepping. For instance, Joukhadar et al. [1996] employ such a technique to optimise the computational efficiency of a dynamic simulation system for robotics by reducing numerical integration step-size where it doesn't affect the stability of the system adversely. Baraff and Witkin [1998] modulate animation detail in a similar manner for cloth simulation when greater accuracy is needed.

Related to this, time-critical techniques in computer graphics employ progressive refinement processes which can be terminated before completion if processing time available for the rendering or simulation of the scene has elapsed. A coarser solution is returned whilst the rendering remains constrained to within a defined budget of computation time, thus ensuring target framerates. Graceful degradation techniques will return this coarser result without excessively sacrificing precision, correctness or stability of the frame being rendered thereby optimising tradeoff between speed and accuracy. Some early examples of this are the adaptive rendering technique of Bergman et al. [1986], virtual environment walkthroughs by Funkhouser and Sequin [1993] and the collision detection system of Hubbard [1996]. Gobbetti and Bouvier [1999] investigated time-critical rendering for multiresolution polygonal models and proposed a method for ensuring time-critical rendering which could be implemented with any generic multiresolution data structure. Zach et al. [2004] investigated dynamic switching between point rendered objects and discrete polygonal geometry in order to achieve real-time rendering of large scale datasets.

In volume rendering, adaptive sampling techniques have been used in order to specify areas in the volume where reduced sampling can be used. Roettger et al. [2003] introduce the concept of an importance volume which is used to reduce the sampling rate in areas of the volume classified as failing to contribute significantly to the final image. This importance volume is pre-calculated and must be regenerated when the transfer function is changed. Hadwiger et al. [2005] discussed using adaptive sampling in order to improve intersection detection for iso-surface volume rendering. Their algorithm automatically increases the sampling rate as the ray nears the iso-value being rendered. Time-critical computing has been employed by Liao et al. [2004] with their investigation of time-critical updates to four dimensional time varying volume data. Their technique does not guarantee that the current volume data being rendered is accurate. Instead they prioritise data upload to the GPU in areas of the volume which have changed the most from frame to frame. While their technique is innovative it is unfortunately only applicable to time variant volume data and does not provide any speed up for static data. Li and Shen [2002] have investigated a technique for performing time-critical volume rendering on texture mapping hardware. They propose generating a hierarchical octree structure for storing multiple level of detail(LOD) representations of the volume data and generating an importance value for each node. Their algorithm automatically selects LOD's from the volume structure in order to achieve the targeted rendering time. Their technique generates a texture for every LOD at the initialisation of their program and stores these textures on the GPU. This decision dramatically increases the memory requirements of the data and also requires a complete update of every texture whenever the user changes the transfer function. In addition their technique requires that a new set of view aligned texture slices are generated for each level of detail node selected for rendering and they do not comment on the overhead required for this step. In another paper Li and Shen [2001] propose a fuzzy logic system for automatically selecting the correct level of detail threshold to use in order to achieve the user targeted frame rate. They classify the current frame rate into three possibilities; high, medium and low and define maximum sampling rate changes for each possibility. They are able to achieve a reasonably fast convergence on the targeted frame rate for texture slicing volume rendering.

3 Implementation

3.1 Background

In this subsection we describe general volume rendering techniques which the reader may find useful in order to understand our contribution but which are not directly related to time-critical rendering.

The current state of the art technique for direct volume rendering uses GPU ray-casting in order to generate a complete rendering of the volume dataset. In GPU ray-casting a three dimensional texture containing the volume data is uploaded to the GPU, this texture contains discrete intensity values at each location called voxels. A fragment program on the GPU is then run for each pixel in the scene. This fragment program implements ray-casting by calculating the direction of a ray cast from the view position through each pixel in the scene. The fragment program iterates along this ray, sampling the volume texture at discrete steps. At each step the fragment program obtains the intensity value of the volume data, maps this value to a colour and transparency using a transfer function texture and adds the colour to a running total. When the ray reaches the end of the volume the running total contains the final colour for the current pixel.

The quality of the final image is heavily dependant on the frequency of the samples along the ray. The more frequently the samples are the smoother and more accurate the final image is. The Nyquist-Shannon [1949] sampling theorem states that in order to accurately recreate a discretely sampled signal it is necessary for the frequency of the sampling rate to be twice the highest frequency in the signal. In practical terms this means that in order to sample the volume accurately it is necessary for the frequency of the samples along the ray to be twice the frequency of the volume data (or in other words, one must sample the volume texture twice between each voxel). However, in addition to sampling the volume accurately it is also required to sample the transfer function texture accurately and so in order to obtain an accurate final image the frequency of samples along the ray must be twice the frequency of whichever has the highest frequency; the volume data or the transfer function.

Unfortunately users very rarely generate a smoothly changing transfer function, instead they tend to want to highlight distinct surfaces in the volume which requires a transfer function which dramatically changes. It is quite common to create transfer functions with infinite frequency due to the requirement to see surface boundaries in the volume data which therefore means it is impossible to sample at a high enough frequency to accurately render an image. Pre-integrated volume rendering [Engel et al. 2001] has been proposed as a method to precalculate the convolution integral of the transfer function and thus eliminate the high frequency sampling requirement. Unfortunately this technique scales with the square of the range of intensity values in the volume data ($O(n^2)$). The range of intensity values captured by modern volume capture techniques has increased dramatically since pre-integrated rendering was proposed and this technique is no longer feasible due to the high memory requirement of the pre-integrated texture.

3.2 Time-Critical Rendering

For modern datasets it is no longer possible to perform a volume rendering with a high enough sampling rate in order to generate a perfectly accurate image. Current interactive techniques deal with this problem by acknowledging the possibility of a non ideal image and allowing the user to specify the rendering quality and hence the rendering speed. Ray-cast implementations enable this change in rendering quality by allowing the user to change how frequently each ray samples the volume. By increasing the sampling rate

the quality of the rendering is increased as smaller features are more likely to be sampled and aliasing artefacts are minimised. This increase in sampling results in an increase in computational cost, however the relationship between sampling rate and computational cost is not easy to quantify as factors such as viewpoint location, percentage of the volume on screen and transfer function content heavily influence the rendering time. Current acceleration techniques such as early ray termination [Kruger and Westermann 2003], empty space skipping [Kruger and Westermann 2003] and level of detail rendering [Crassin et al. 2009] further complicate this relationship as they provide view dependant decreases in computational cost. For a user who wishes to maintain a certain frame rate in order to interact with a volume dataset it is not sufficient to specify a constant sampling rate and it is required to vary the sampling rate as both viewpoint location and transfer function content change.

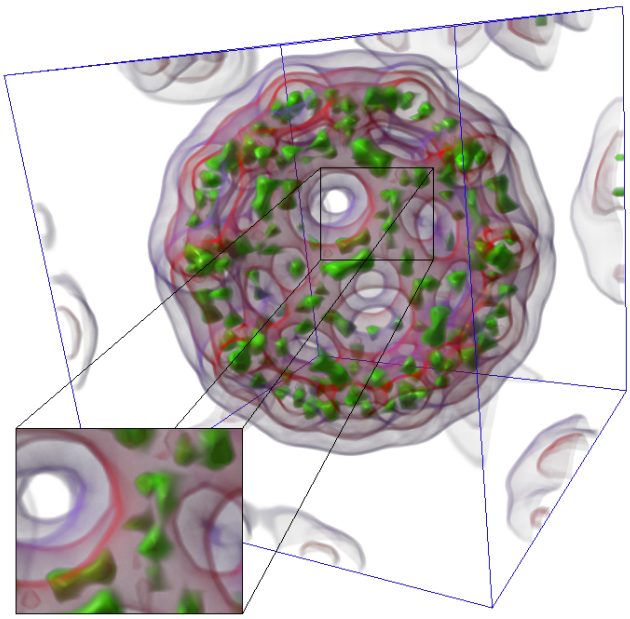

Figure 2: *A screenshot from our implementation with Time-Critical sampling enabled, our technique has identified extra computation cycles and increased the sampling rate in order to generate a higher quality image while still maintaining the target frame rate.*

The rendering time for individual frames may vary widely depending on the factors described above, however for a sufficiently high frame rate, localised frames should be broadly alike and therefore have similar rendering times. We observe that for interactive navigation of a dataset users require frame rates which are sufficiently high that adjacent frames achieve similar rendering times. This observation allows us to use knowledge of the previous frames rendering time and sampling rate in order to tailor the sampling rate of the next frame in order to achieve the target rendering time. By linearly interpolating the sampling rate of the previous frame between the previous frames render time and the desired render time we can automatically vary the sampling rate in order to ensure a constant frame rate. Equation 1 shows the calculation we perform to determine the new sampling rate where S_n is the new sampling rate, S_{n-1} is the previous frames sampling rate, f_{n-1} is the frames per second of the previous frame and f_{target} is the user selected target frame rate.

$$S_n = S_{n-1} * (f_{n-1}/f_{target}) \quad (1)$$

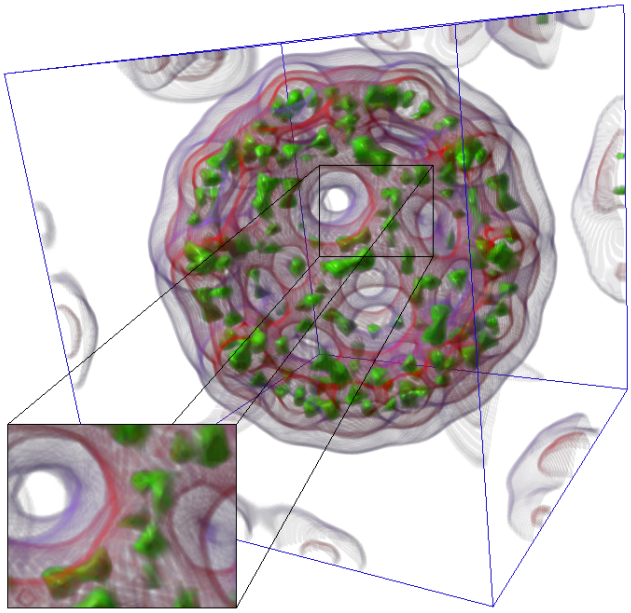

Figure 3: *A screenshot from a standard volume rendering implementation, notice how the quality of the image remains poor even though the frame rate has increased beyond the original frame rate selected by the user. The user had originally selected a sampling rate which provided interactive frame rates at low quality but when the view was zoomed further out this reduced sampling rate failed to take advantage of spare computational cycles available.*

Our algorithm dynamically adjusts the sampling rate of a ray-cast direct volume rendering on a frame by frame basis in order to ensure that the highest possible quality rendering for a given time budget is achieved. By ensuring that the only change from frame to frame is the global sampling rate we ensure that our method can be integrated with a wide variety of existing acceleration techniques. To date we have integrated it with both early ray-termination and volume bricking [Hadwiger et al. 2005] but our algorithm can be applied to nearly all existing ray casting acceleration methods as the only requirement is the ability to vary the sampling rate on a per frame basis, something which is widely supported by ray-cast rendering.

At very low sampling rates significant wood grain artefacts occur in the majority of ray-cast volume rendering systems. Our technique is no different, however due to our method of dynamically changing the sampling rate with every frame these artefacts will tend to move slightly from frame to frame and temporal coherence is lost. This is the only drawback of our technique but fortunately a method already exists to eliminate wood grain artefacts. Ray-cast jittering [Hadwiger et al. 2006] is a method for randomly varying the start position of each ray which eliminates the artefacts and introduces some slight blurring. This blurring is much less noticeable than wood grain artefacts and with jittering enabled our technique maintains frame to frame coherence. There is some slight overhead to enabling jittering in the form of generating a small texture containing random values however this only needs to be done once at initialisation and there is no ongoing cost on a frame by frame basis. Therefore the only significant drawback to our technique can be eliminated by an already proven method which contains no overhead bar an initial generation of a small texture.

3.2.1 Implementation Details

Our technique uses a high precision CPU timer available through the Windows API in order to accurately account for how long it takes to render each frame and is implemented in OpenGL using the Voreen [Meyer-Spradow et al. 2009] volume rendering engine. However we need to take account of the buffered nature of the OpenGL API which allows the graphics system to buffer commands before executing them. We need to obtain an accurate and high precision timing of how long it took to generate each individual frame but OpenGL by default does not guarantee that each command will be executed immediately. In order to circumvent this we need to force the OpenGL graphics system to clear all its buffers and execute all waiting instructions before we stop our frame timer. The command glFinish() blocks waiting for all OpenGL commands to execute, this command causes a stall in CPU execution and performs a full round trip to the graphics card and can be quite computationally expensive if called indiscriminately. However by ensuring that we have performed all necessary CPU instructions and waiting till the very end of the pipeline before calling glFinish() we can ensure that we have completed all the rendering steps and avoid any costly blocking overhead.

Once we can accurately obtain the rendering time for each individual frame we can use the sampling rate and the computation time of the previous frame in order to calculate the sampling rate for the next frame in order to obtain the user specified target frame rate.

In a standard volume ray-casting implementation the GPU ray-casting routine calculates the distance between samples along the ray based on a static variable representing the sampling rate. In order for our technique to dynamically change the sampling rate on a per frame basis we must update the sampling rate variable on the GPU every frame. There is inbuilt support for fast updates to variables in the GPU and we can calculate the new sampling distance and update the GPU variable with virtually no overhead.

4 Evaluation

We tested our technique by recording the frame rate and the sampling rate for a 360° rotation around a number of random volume datasets (See Table 1). We then compared the differences between a standard rendering and our time-critical rendering. As expected the frame rate for a standard rendering can vary quite dramatically, in same cases there can be as much as a 40% difference between the maximum and the minimum frame rate for the same dataset (eg. the walnut dataset). In contrast our technique quickly adapts to any changes in scene complexity and dynamically adjusts the samping rate in order to maintain the targetted frame rate. As can be seen from Figures 4-11 and Table 2 our technique is successful in smoothing out variations in frame rate and providing the user with a constant interactive frame rate in order to easily navigate the volume.

All calculations were performed on a machine equipped with a Intel Core 2 Quad Q6600 processor, 4 Gigabytes of RAM and a Nvidia GeForce 9800 GX2 graphics card, additionally all renderings were performed with a window size of 512 x 512 pixels.

Name	Size
Vismale[1]	128 x 256 x 256
Walnut[2]	128 x 96 x 114
Porsche[1]	559 x 1023 x 347
Bucky[1]	32 x 32 x 32

Table 1: *The dimensions of the datasets we used in our evaluation.*

	Standard Rendering		Time-Critical Rendering	
	Average	σ	Average	σ
Vismale	11.03	0.73	11.01	0.25
Walnut	13.32	2.09	14.01	0.35
Porsche	6.25	1.13	6.27	0.17
Bucky	13.83	0.98	13.00	0.27

Table 2: *Table showing the average frames per second and the standard deviation for each dataset. Our time-critical technique significantly reduces the standard deviation resulting in smoother and less varied frames per second.*

Our technique integrates into the core rendering loop of any ray-cast direct volume rendering system and only requires a very simple linear interpolation calculation and a timing query in order to work. Therefore the overhead is minimal. From analysis of the frames per second numbers produced from our testing we observed that our technique produced no significant overhead. Our method is able to quickly and consistently determine the ideal sampling rate in order to achieve the targeted frame rate. This ability to guarantee a given frame rate is invaluable in generating volume renderings for interactive user navigation.

As can be seen from Figures 2 and 3 we can reduce the appearance of artefacts associated with low sampling rate renderings and improve final image quality. In addition our technique requires very little modification to a normal ray-cast volume rendering pipeline and therefore can be integrated into many already existing volume rendering optimisations.

5 Conclusions and Future Work

We have presented a system which allows for dynamic variation of the sampling rate of ray-cast direct volume rendering in order to maintain a stable frame rate. Our system is robust and quick to respond to both user input and significant computational loads and rapidly converges on the target frame rate. We have tested our technique with a wide variety of datasets and it has succeeded in maintaining the target frame rate on all datasets at a wide variety of target frame rates. Our technique can be integrated into existing ray-casting optimisation techniques and contains no significant overhead.

In addition we are investigating varying the sampling rate on a local basis in image space in order to allocate higher sampling rates in regions of the final image that are more likely to have significant volume information. By increasing the sampling rate in important areas and reducing the sampling rate elsewhere we can increase the overall image quality without any extra computation. Once again we can use our knowledge of the similarity between adjacent frames in order to generate an importance map to guide the local sampling rate. By combining both our local and global sampling methods together we intend to generate a high quality time critical rendering with low computation cost.

References

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 43–54.

BERGMAN, L., FUCHS, H., GRANT, E., AND SPACH, S. 1986. Image rendering by adaptive refinement. In *Proceedings of*

[1]Data from http://www9.informatik.uni-erlangen.de/External/vollib/

[2]Data from http://www.voreen.org/

the 13th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '86, 29–37.

CABRAL, B., CAM, N., AND FORAN, J. 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 symposium on Volume visualization*, ACM, Tysons Corner, Virginia, United States, 91–98.

CHEN, M., KAUFMAN, A., AND YAGEL, R., Eds. 2001. *Volume Graphics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

CRASSIN, C., NEYRET, F., LEFEBVRE, S., AND EISEMANN, E. 2009. GigaVoxels: ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, Boston, Massachusetts, 15–22.

CULLIP, T. J., AND NEUMANN, U. 1994. Accelerating volume reconstruction with 3D texture hardware. Tech. rep., University of North Carolina at Chapel Hill.

ENGEL, K., KRAUS, M., AND ERTL, T. 2001. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM, Los Angeles, California, United States, 9–16.

FUNKHOUSER, T. A., AND SÉQUIN, C. H. 1993. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '93, 247–254.

GOBBETTI, E., AND BOUVIER, E. 1999. Time-critical multiresolution scene rendering. In *Proceedings of the conference on Visualization '99: celebrating ten years*, IEEE Computer Society Press, Los Alamitos, CA, USA, VIS '99, 123–130.

HADWIGER, M., SIGG, C., SCHARSACH, H., BÜHLER, K., AND GROSS, M. 2005. Real-Time Ray-Casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum 24*, 3, 303–312.

HADWIGER, M., KNISS, J. M., REZK-SALAMA, C., WEISKOPF, D., AND ENGEL, K. 2006. Stochastic jittering. In *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 220–223.

HUBBARD, P. M. 1996. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graph. 15* (July), 179–210.

JOUKHADAR, A., LAUGIER, C., AND ALPES, I. R. 1996. Adaptive time step for fast converging dynamic simulation system. In *In Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 418–424.

KRUGER, J., AND WESTERMANN, R. 2003. Acceleration techniques for GPU-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, IEEE Computer Society, 38.

LI, X., AND SHEN, H. 2002. Time-critical multiresolution volume rendering using 3D texture mapping hardware. In *Volume Visualization and Graphics, 2002. Proceedings. IEEE / ACM SIGGRAPH Symposium on*, 29–36.

LI, X., AND WEI SHEN, H. 2001. Adaptive volume rendering using fuzzy logic. *IN PROCEEDINGS OF JOINT EUROGRAPHICS-IEEE TCVG SYMPOSIUM ON VISUALIZATION*.

LIAO, S., LA, J. Z. C., AND CHUNG, Y. 2004. Time-critical rendering for time-varying volume data. *Computers & Graphics 28*, 2, 279 – 288.

LJUNG, P., LUNDSTROM, C., YNNERMAN, A., AND MUSETH, K. 2004. Transfer function based adaptive decompression for volume rendering of large medical data setssdfsdfsd. In *VV '04: Proceedings of the 2004 IEEE Symposium on Volume Visualization and Graphics*, IEEE Computer Society, Washington, DC, USA, 25–32.

MEYER-SPRADOW, J., ROPINSKI, T., MENSMANN, J., AND HINRICHS, K. 2009. Voreen: A Rapid-Prototyping environment for Ray-Casting-Based volume visualizations. *IEEE Computer Graphics and Applications 29*, 6, 6–13.

PREIM, B., AND BARTZ, D. 2007. *Visualization in Medicine: Theory, Algorithms, and Applications (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

ROETTGER, S., GUTHE, S., WEISKOPF, D., ERTL, T., AND STRASSER, W. 2003. Smart Hardware-Accelerated volume rendering.

SHANNON, C. E. 1949. Communication in the presence of noise. *Proc. Institute of Radio Engineers 37*, 1 (Jan.), 10–21.

ZACH, C., MANTLER, S., AND KARNER, K. 2004. Time-Critical rendering of huge ecosystems using discrete and continuous levels of detail. *Presence: Teleoper. Virtual Environ. 13* (Dec.), 656–667.

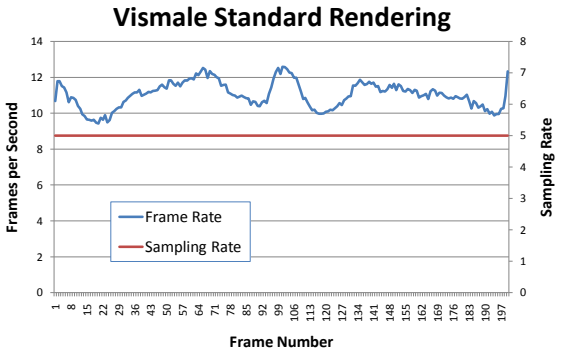

Figure 4: *Results of a standard rendering of the Vismale dataset.*

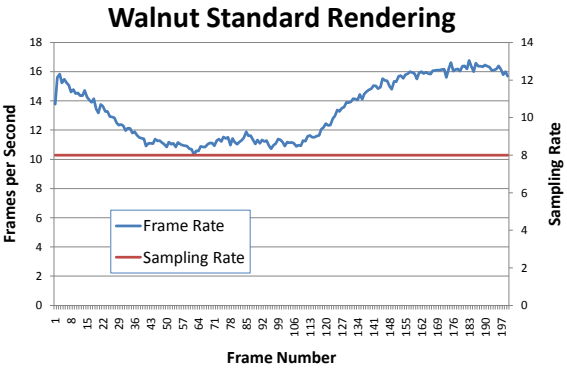

Figure 5: *Results of a standard rendering of the Walnut dataset.*

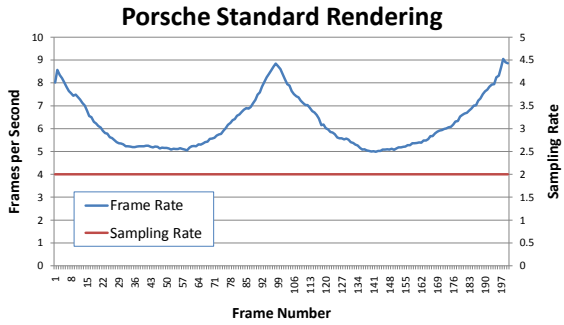

Figure 6: *Results of a standard rendering of the Porsche dataset.*

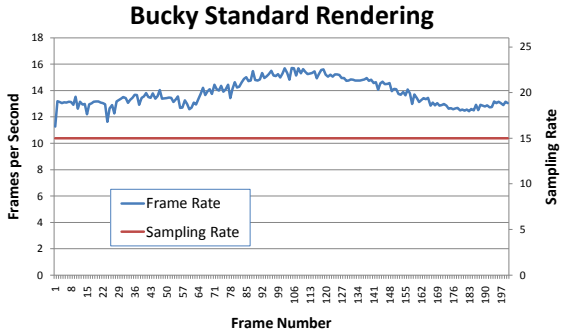

Figure 7: *Results of a standard rendering of the Bucky dataset.*

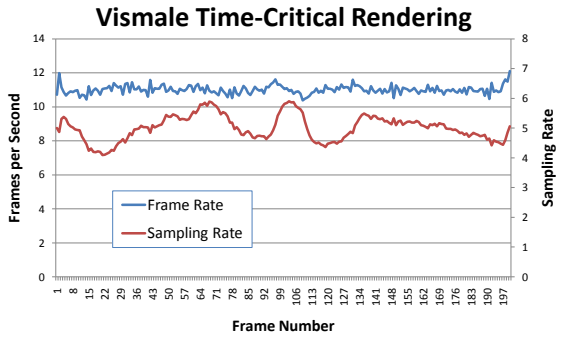

Figure 8: *Results of a time-critical rendering of the Vismale dataset.*

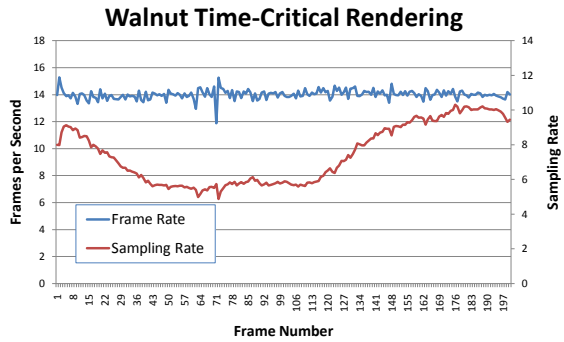

Figure 9: *Results of a time-critical rendering of the Walnut dataset.*

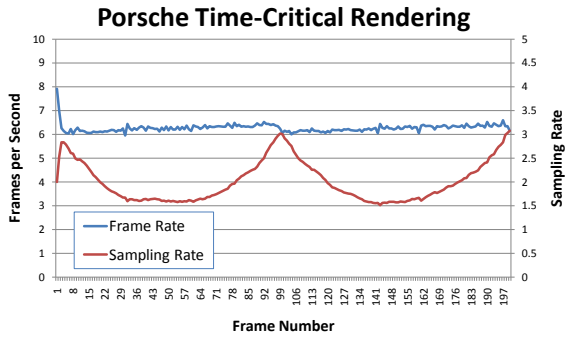

Figure 10: *Results of a time-critical rendering of the Porsche dataset.*

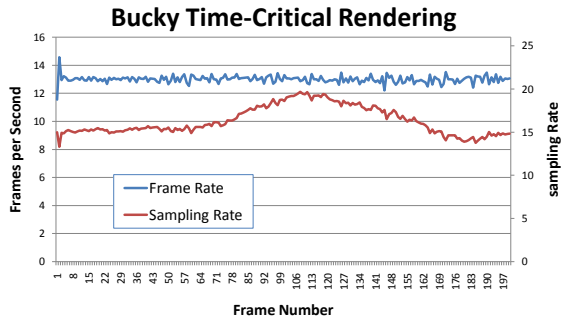

Figure 11: *Results of a time-critical rendering of the Bucky dataset.*