

Maintaining Constant Frame Rates in 3D Texture-Based Volume Rendering

Daniel Weiskopf Manfred Weiler Thomas Ertl

Institute of Visualization and Interactive Systems

University of Stuttgart

{weiskopf|weiler|ertl}@vis.uni-stuttgart.de

Abstract

3D texture-based volume rendering is a popular way of realizing direct volume visualization on graphics hardware. However, the slice-oriented texture memory layout of many current GPUs may lead to a strongly view-dependent performance, which reduces the fields of application of volume rendering. In this short technical note, we propose a slight modification of texture-based volume rendering that maintains roughly constant frame rates on any GPU architecture. The idea is to split the volume into smaller sub-volumes. These bricks can be oriented in different directions; thus the varying performance for different viewing directions is averaged out.

1. Introduction

Volume rendering is a widely used technique in many fields of application, ranging from the direct volume visualization of scalar fields for engineering and sciences to medical imaging and finally to the realistic rendering of clouds or other gaseous phenomena in visual simulations and virtual environments. In recent years, texture-based volume rendering on consumer graphics hardware has become a popular approach for direct volume visualization. The performance and features of GPUs (Graphics Processing Units) have been increasing rapidly, while their prices have been kept attractive even for low-cost PCs. The OpenGL 1.2 standard includes support for 3D textures and, therefore, 3D textures are available on a large number of current GPUs.

Texture-based volume rendering with view-aligned slices in combination with 3D textures [4] offers some advantages compared to the alternative of using stacks of 2D textures. First, only one third of the texture memory is required for the 3D texture method; the 3D approach just holds a single instance of the volume, while the 2D approach has to store the complete volume for each of the three main axes. Second, view-aligned slicing through a 3D texture avoids the artifacts that occur when texture stacks

are switched. Third, the intrinsic trilinear interpolation in 3D textures directly allows us to use an arbitrary number of slices with an appropriate resampling on these slices, i.e., the quality of the rendering can be easily adjusted by adapting the slice distance.

Since trilinear interpolation needs access to a neighborhood of eight voxels, volume rendering is associated with high bandwidth requirements for data transfer from texture memory. Usually, the texture cache is optimized for fetch operations in 2D textures, which are important for most applications. On the other hand, different hardware architectures use different memory layouts for 3D textures. In one approach, the volume is essentially built from a collection of 2D textures in a slice-by-slice fashion. Here, a good caching strategy is only available within a 2D slice, but not along the stacking axis. Therefore, the cache behavior greatly depends on the order of accessing the 3D texture and the performance of volume rendering can become view-dependent. This characteristic is inappropriate for real-time applications that have to maintain constant frame rates. In a second approach, a memory layout is chosen that achieves a comparable cache coherence along all directions. Here, the volume is not constructed slice-by-slice, but in a more complex, staggered manner.

Both memory layouts can be found in today's GPUs since there are good reasons for both approaches. For example, the latter architecture shows a uniform performance behavior for volume rendering and therefore is well-suited for real-time applications like virtual environments. On the other hand, the slice-by-slice method allows the application to update a volume slice very efficiently (because of its direct mapping to the memory); with ATI's superbuffer (uberbuffer) extension [13], even a direct rendering into a volume slice is possible. With the increasing popularity of computing numerical simulations on GPUs, this ability will become even more important in the future, e.g., see the implementation of level-set methods by Sherbondy et al. [15].

The goal of this short technical note is to propose a slight modification of 3D texture-based volume rendering that maintains roughly constant frame rates on any GPU ar-

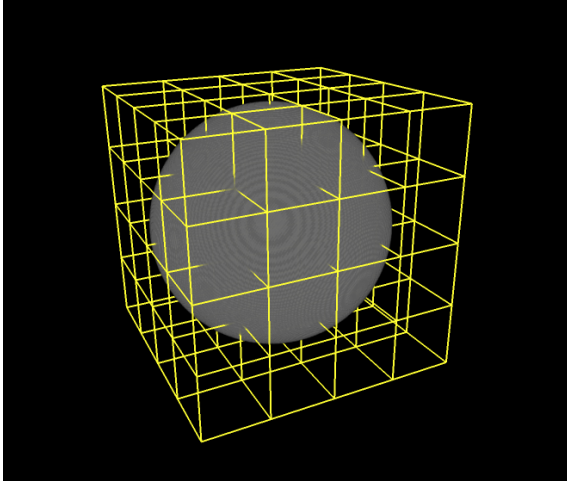


Figure 1. Partitioning a volume data set into 4^3 bricks.

chitecture. The idea is to split the volume into smaller sub-volumes, i.e., bricks [7]. These bricks can be oriented in different directions; thus the varying performance for different viewing directions is averaged out within a single image.

2. Previous Work

Cabral et al. [4] were early to use general purpose graphics hardware and its 3D texture support in order to implement interactive volume rendering with viewport-aligned slicing. They could make use of the Reality Engine [2], which supports 3D textures. An improved performance for volume rendering was provided by the successor Infinite-Reality [12]. These hardware architectures have the advantage of a 3D texture memory layout that provides good cache coherence for any viewing direction. The original rendering technique [4] supports only the model of a gas that directly absorbs and emits light. More recent research on texture-based volume rendering has led to advanced volumetric illumination and shading techniques, e.g., see the work by Van Gelder and Kim [16], Dachille et al. [5], Westermann and Ertl [18], Engel et al. [6], or Kniss et al. [8, 10]. Rezk-Salama et al. [14] describe a method for trilinear interpolation on additional slices within a 2D texture-based approach. Their technique relies on multi-texturing and requires changes of the core rendering routine.

3. Multi-Oriented Bricking

To overcome the potential view dependency in texture-based volume rendering, we propose to partition the volume into smaller bricks. Figure 1 shows an example, where the

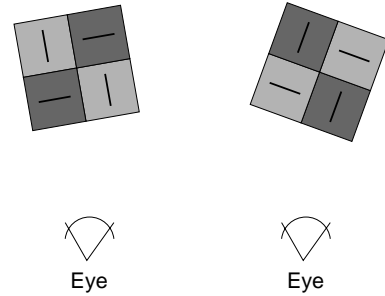


Figure 2. For any viewing direction, two bricks are oriented along the viewing direction and the other two bricks are perpendicular.

volume data set (that is visualized as a sphere) is split into 4^3 sub-volumes. The borders of the bricks are marked by yellow lines.

These bricks can be oriented in different directions; thus the varying performance for different viewing directions is averaged out within a single image. Figure 2 illustrates (in the simplified analog of a 2D drawing) that the number of bricks which are oriented along the viewing direction can be kept constant. Therefore, the number of elements that can be rendered at high speed is independent of the viewing direction. Note that the “fast” bricks may change. For example, in the left image of Figure 2, the bricks in light gray lead to higher rendering speed, whereas the bricks in dark gray are faster to render in the right image.

Of course, this idea works only as long as the whole volume is in the visible view frustum and roughly the same number of fragments are associated with each brick. In real applications, these two assumptions are only partly fulfilled. In these cases, the uniformity of rendering speed is the better the finer the granularity of the bricking is chosen, i.e., the smaller the sub-volume sizes become. On the other hand, finer granularity leads to a larger number of slicing polygons to be computed and rendered because each slice through the complete volume has to be partitioned into sub-slices for the bricks. Moreover, a brick should have a one-*texel*-wide overlap with neighboring bricks [7]. Therefore, additional texture memory is needed, the consumption of which increases when the number of bricks is increased. Note, however, that the size of the additional bricks can be chosen independently of the size of the other bricks and thus only little memory is wasted by the restriction to power-of-two textures.

In summary, the granularity should be optimized by taking into account the above aspects. The optimal number of bricks is highly dependent on the application and the characteristics of user navigation. An extensive discussion of this issue would be beyond the scope of this short paper.

According to our experiments, bricking into 2^3 or 4^3 sub-blocks leads to reasonable results, as long as the viewing parameters are not too extreme. Corresponding performance measurements can be found in the following section.

4. Performance Comparison

In this section, we show a performance analysis for the bricking approach and compare this method with the original non-bricking rendering. We use two typical representatives for the two types of memory layout: The ATI Radeon 9800 Pro as an example for a GPU with slice-oriented layout and the NVidia GeForce FX 5950 Ultra as an example for a graphics board with an optimized memory structure for volume rendering. The size of our test data set is 256^3 and shows a spherically symmetric behavior with a linear dependency on the distance from the center point. Figure 1 depicts the volume visualization of this data set. We use pre-classification with a two-component 3D texture that stores pre-computed luminance and alpha values. The slicing distance is kept constant, i.e., the number of slices may change with the viewing direction. Just the fixed-function vertex and fragment pipeline is used. In this way, our test case is focused on the 3D texture access speed. The viewport has a size of 800^2 pixels in all performance tests.

Figure 3 compares the rendering speed for the two architectures, based on a single brick (i.e., standard view-aligned volume rendering). The rendering speeds are shown in fps (frames per seconds) along the vertical axis. The horizontal axis describes the viewing direction in degrees. The angles denote a rotation of the volume about the y axis, which forms a vertical line on the viewing plane. We assume that the z axis is parallel to the stacking axis for building the 3D texture from slices. For an angle of zero, the viewer looks along the z axis of the texture. Figure 3 shows that the rendering performance of the slice-oriented architecture heavily depends on the viewing angle. In this case, the minimum and maximum frame rates form a ratio of 1:4.8. In contrast, the other hardware architecture maintains a much more uniform frame rate; there is only a 12 percent difference between minimum and maximum frame rates.

Figure 4 demonstrates that multi-oriented bricking achieves a much more even rendering performance than the standard approach for the Radeon 9800 Pro. Here, a 4^3 bricking is applied. The directions of the bricks are alternated between the x, y, and z axes, based on the index of the bricks. The difference between minimum and maximum frame rates is reduced to 1:1.2.

Figure 5 shows the same comparison between 4^3 bricking and standard volume rendering for the FX 5950 Ultra. The performance numbers indicate that the bricking approach is a little bit slower, which is mainly caused by

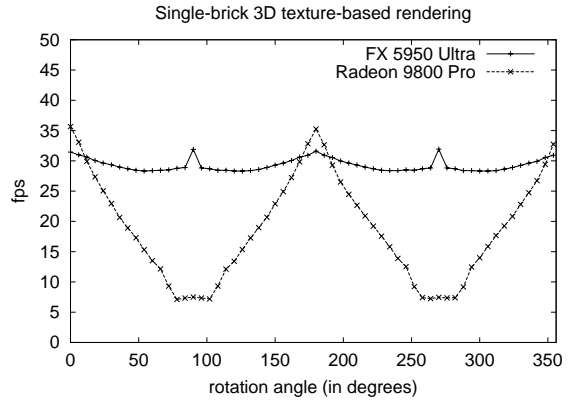


Figure 3. Single-brick volume rendering on FX 5950 Ultra and Radeon 9800 Pro.

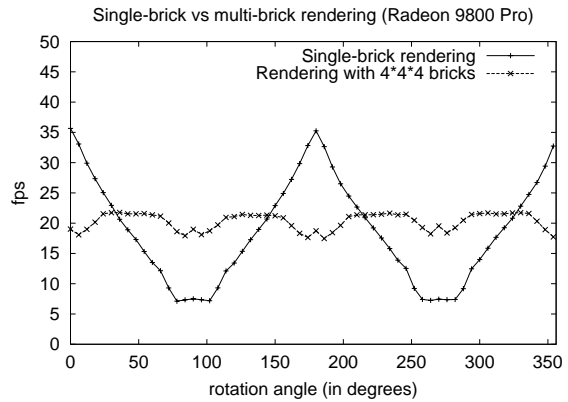


Figure 4. Comparing volume rendering with single brick and 4^3 bricks on Radeon 9800 Pro.

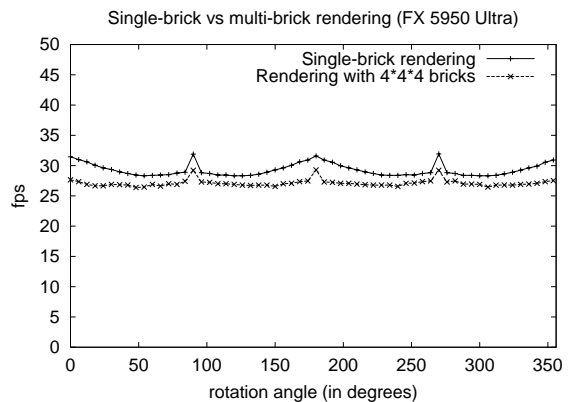


Figure 5. Comparing volume rendering with single brick and 4^3 bricks on FX 5950 Ultra.

the additional work for the increased number of slice polygons. These measurements illustrate how much performance penalty is associated with bricking: The overall rendering speed is reduced by less than five percent.

5. Conclusion

We have proposed a simple and practical improvement for 3D texture-based volume rendering on an important class of current GPUs that have a slice-oriented memory layout for volumetric textures. The idea is to compensate the different rendering performance along different viewing directions by shuffling the orientations of sub-blocks of the volume according to an equal distribution of all possible three orientations.

Bricking is frequently used in volume rendering already. For example, large datasets that do not fit into texture memory at once can only be handled by bricking [9, 11, 17]. Moreover, bricking overcomes the waste of texture memory that occurs when data sets of arbitrary size have to be extended by empty space to meet the restriction to power-of-two textures. Therefore, bricking is already available in many volume rendering applications (e.g., in OpenGL Volumizer [3] or the volume node of OpenSG [1]) and our extension causes only minimal additional implementation efforts. We think that multi-oriented bricking is a practical and easy-to-handle solution that, in particular, improves the usage of volume rendering in real-time sensitive applications such as virtual reality and visual simulations. Typical implementations of virtual environments freeze the overall frame rate to the lowest achievable frame rate, which would be unacceptable if the ratio between maximum and minimum rendering speeds is large. Therefore, our approach leads to a significant performance increase in such environments.

Acknowledgments

The first author thanks the Landesstiftung Baden-Württemberg for support.

References

- [1] OpenSG Web Page. <http://www.opensg.org>, 2003.
- [2] K. Akeley. Reality Engine graphics. In *Proceedings of ACM SIGGRAPH 1993*, pages 109–116, 1993.
- [3] P. Bhaniramka and Y. Demange. OpenGL Volumizer: A toolkit for high quality volume rendering of large data sets. In *2002 Symposium on Volume Visualization and Graphics*, pages 45–54, 2002.
- [4] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *1994 Symposium on Volume Visualization*, pages 91–98, 1994.
- [5] F. Dachille, K. Kreeger, B. Chen, I. Bitter, and A. Kaufman. High-quality volume rendering using texture mapping hardware. In *1998 Eurographics / SIGGRAPH Workshop on Graphics Hardware*, pages 69–76, 1998.
- [6] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *2001 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 9–16, 2001.
- [7] R. Grzeszczuk, C. Henn, and R. Yagel. Advanced geometric techniques for ray casting volumes. *ACM SIGGRAPH 1998 Course #4 Notes*, 1998.
- [8] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002.
- [9] J. Kniss, P. McCormick, A. McPherson, J. Ahrens, J. Painter, A. Keahey, and C. Hansen. Interactive texture-based volume rendering for large data sets. *IEEE Computer Graphics and Applications*, 21(4):52–61, 2001.
- [10] J. Kniss, S. Premoze, C. Hansen, P. Shirley, and A. McPherson. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):150–162, 2003.
- [11] E. C. LaMar, B. Hamann, and K. I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *IEEE Visualization 1999*, pages 355–362, 1999.
- [12] J. S. Montrym, D. R. Baum, D. L. Dignam, and C. J. Migdal. InfiniteReality: A real-time graphics system. In *Proceedings of ACM SIGGRAPH 1997*, pages 293–302, 1997.
- [13] J. Percy and R. Mace. OpenGL extensions: SIGGRAPH 2003. <http://mirror.ati.com/developer/techpapers.html>, 2003.
- [14] 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 *2000 Eurographics / SIGGRAPH Workshop on Graphics Hardware*, pages 109–118, 2000.
- [15] A. Sherbondy, M. Houston, and S. Napel. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *IEEE Visualization 2003*, pages 171–176, 2003.
- [16] A. Van Gelder and K. Kim. Direct volume rendering with shading via three-dimensional textures. In *1996 Symposium on Volume Visualization*, pages 23–30, 1996.
- [17] W. R. Volz. Gigabyte volume viewing using split software/hardware interpolation. In *2000 Symposium on Volume Visualization*, pages 15–22, 2000.
- [18] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *Proceedings of ACM SIGGRAPH 1998*, pages 169–178, 1998.