Efficient Shadows for GPU-based Volume Raycasting

Timo Ropinski Jens Kasten Klaus Hinrichs
Department of Computer Science
University of Münster
Einsteinstraße 62
48149 Münster, Germany
{ropinski, j_kast03, khh}@math.uni-muenster.de

ABSTRACT

GPU-based raycasting has emerged as the defacto standard for interactive volume rendering on off-the-shelf graphics hardware. Even though in theory this technique can be easily extended by shadow feelers in order to support shadows, this obvious approach has a major impact on the rendering performance. In this paper we will investigate shadowing extensions for GPU-based volume raycasting and compare them with respect to their quality and their performance. In particular, we will consider shadow rays, shadow mapping and deep shadow maps. For these techniques we will address their implementations using current graphics boards, and we will compare their visual results as well as their runtime behavior.

Keywords
Volume Rendering, Shadow Generation, Shadow Mapping, Deep Shadow Maps.

1 INTRODUCTION

Volume rendering has evolved into a mature area of computer graphics. Many domains such as medical acquisition or seismic sonography techniques produce volumetric data sets which have to be explored interactively. Besides an immediate visual feedback also good perception of the spatial relationships is important. It has been shown that the used lighting model has a major impact on the spatial comprehension [LB00]. Besides diffuse interreflections also shadows serve as an important depth cue [SSMK05]. While volume rendering techniques developed in recent years mainly had the goal to achieve interactive frame rates, the objective of realistic renderings has been neglected. Now that interactive volume rendering is possible on standard graphics hardware [KW03] it is time to strive for producing more realistic images in order to support better spatial comprehension. In this paper we focus on shadow algorithms which can be integrated into volume rendering in order to provide additional depth cues (see Figure 1). We consider in particular those algorithms and implementations which have the potential to permit interactive shadowing. However, for efficient shadow generation it is important to adapt the used techniques to the volume rendering techniques exploited for image generation. Although different algorithms have been proposed for slice-based volume rendering [KPHE02], we will concentrate on techniques suitable to integrate shadows in GPU-based volume raycasters [KW03], since GPU-based volume raycasting has emerged as the state-of-the-art volume rendering technique which is found in most recent volume rendering frameworks. While the benefits of GPU-based raycasting include an easy as well as flexible implementation, it is also often stated that GPU-based raycasting allows an easy integration of shadowing effects. However, while the brute force approach for shadows can be truly considered as easy to integrate, it is not efficient at all. In fact, it is just an adaptation of the shadow feeler rays known from conventional raytracers, with all their downsides. To provide the reader with a more detailed knowledge about shadows in volume rendering, this paper reviews different shadow algorithms and their implementations in the context of GPU-based volume raycasting. In particular we will compare the brute force approach, i.e., casting of shadow rays, with a shadow mapping technique for GPU-based raycasting as well as deep shadow maps.

The remainder of this paper is structured as follows. In Section 2 we will discuss related work regarding shadow generation by specifically addressing algorithms suitable for interactive volume rendering. In Section 3 our implementations of the shadow algorithms are discussed, and they are compared in Section 4. The paper concludes in Section 5 by giving some overview over future work.

2 RELATED WORK

The underlying GPU-based volume raycasting technique, in the following called GPU-based raycasting, has been introduced by Roettger et al. [RGWE03].

Figure 1: The engine data set ($256 \times 256 \times 128$ voxel) rendered using phong shading in combination with shadow mapping.

Krüger and Westermann have proposed extensions to better exploit the capabilities of current graphics hardware [KW03]. GPU-based raycasting enables very efficient volume rendering on commodity graphics hardware by casting rays through the volume data set, which is represented as a 3D texture. To determine the entry and exit points for the rays the bounding box of the volume data set is rendered in a color coded manner, such that the values in the RGB color channels can be interpreted as volume texture coordinates.

Various shadow algorithms have been developed in the past. Crow has proposed a shadow volume technique for generating shadows for scenes containing polygonal data [Cro77]. To compute shadows of an object, its silhouette is extracted and extruded in direction of the light rays in order to generate shadow polygons which form the shadow volume of an object. During rendering each object is tested whether it lies inside or outside a shadow volume, and thus it can be determined whether the object is shadowed. Due to the polygonal nature of this algorithm, it is not suited for volume rendering. Another common approach for generating shadows when rendering polygonal data is shadow mapping which has been presented in 1978 by Williams [Wil78]. Shadow mapping is an image-based approach which exploits an additional rendering pass in which the scene is rendered from the light source's point of view in order to determine the structures closest to the light source. With this knowledge, a fragment-based shadow test can be introduced in the main rendering pass, i.e., each fragment is tested whether it is further away from the light source than the corresponding texel in the shadow map. While shadow mapping allows very efficient shadows on a fragment basis, it does not support semi-transparent occluders which can be found in volume rendering. In order to address semi-transparent structures opacity shadow maps serve as a stack of shadow maps, which store alpha values instead of depth values in each shadow map [KN01].

A more compact representation for semi-transparent occluders are deep shadow maps [LV00]. The used data structure consists also of a stack of textures, but in contrast to the opacity shadow maps, an approximation to the shadowfunction is stored in these textures. Thus it is possible to approximate shadows by using fewer hardware resources. Deep shadow mapping has already been applied to volume rendering [HKSB06]. In this paper we will propose an alternative implementation.

Although work has been done regarding shadows in slice-based volume rendering approaches [BR98], to our knowledge the integration of shadows into GPU-based raycasting, which is considered the state-of-the-art volume rendering technique, has not been addressed so far. However, due to the similar ray paradigm, it should be noted that shadows have been integrated in volume ray-tracing systems [WKB+02, WFMS05]. An overview of these techniques can be found in [MFS06].

While the previously cited work as well as this paper focus on volume rendering, it should also be mentioned that some researchers address the combination of polygonal and volumetric data when considering light interactions [ZXC05]. Also soft-shadowing techniques [CD03, ZC03, HLHS03] are not covered in this paper, but it could be investigated how they can be extended to volume rendering.

3 IMPLEMENTATION OF SHADOW TECHNIQUES

In this section we address the implemented shadowing techniques, which we will compare in Section 4. We review the concept and implementation details of shadow rays (see Subsection 3.1), describe our implementation of shadow maps to exploit them efficiently in GPU-based raycasting (see Subsection 3.2) and will describe our improved deep shadow map implementation (see Subsection 3.3).

3.1 Shadow Rays

Shadow rays are the most obvious way to integrate shadows into a GPU-based raycaster. For each sample along the viewing ray a shadow feeler is sent to the light source in order to determine whether the current sample is shadowed.

We have implemented three different variations of this shadowing technique. To support opaque occluders only, each shadow feeler is cast until a sample is reached where the intensity exceeds the shadow threshold. If such a sample lies along the shadow feeler, the current sample is assumed to lie in shadow. However, due to the fact that only the first occurrence of a sample exceeding the shadow threshold is taken into account, shadowing becomes a binary effect, i.e., a sample is either shadowed or unshadowed. To achieve different

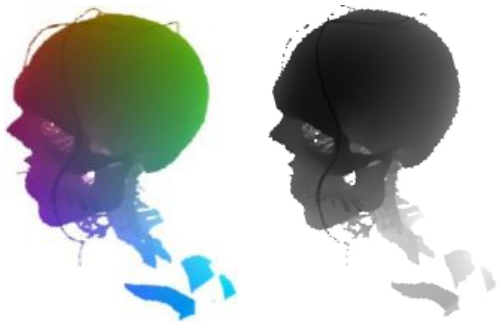

Figure 2: The first hit points in color-coded volume coordinates as seen from the light source (left) and the resulting depth map (right). Both maps have a resolution of 512×512 texel and are generated during rendering the visible human head data set (see Figure 8).

visual results, the shadowing threshold can be changed through the GUI.

The straight-forward extension to support semi-transparent occluders incorporates the intensities of the samples along the shadow feelers. When casting the shadow feelers, the rendering integral is evaluated and for each sample the intensity is considered as the alpha value describing the degree of absorption. Since there is no early ray-termination as when using opaque occluders only, this technique has a recognizable impact on the performance.

For both shadow ray techniques, a shadow feeler needs to be cast for each sample along a viewing ray. Thus the runtime is directly proportional to the screen resolution and the sampling rate. Especially when high quality images are desired this may prevent interactive frame rates. Therefore we have combined the shadow ray technique allowing opaque occluders with a render to 3D texture functionality to be able to cache the results of the shadow feelers: When casting the shadow feeler for a sample, the resulting shadow value is stored within a shadow lookup volume. By using this caching the shadow feelers need only be casted when either the light source or the occluders are changed.

3.2 Shadow Mapping

Shadow mapping can be used as an alternative approach to the shadow rays supporting opaque occluders. In this section we describe our shadow mapping implementation adapted for GPU-based raycasting. In analogy to the general shadow mapping approach (see Subsection 2), we generate a depth map in order to store light source visibility. Therefore we compute the first hit positions as seen from the light source and use them to compute the light source distance (see Figure 2). When rendering polygonal models a depth value is properly defined. In contrast, volume rendering does not provide depth values in general. Therefore we again introduce a shadow threshold value, which can be controlled using the GUI. Thus the intensity values representing opaque geometry can be changed, and with it the content of the generated shadow map. The example shadow map shown in Figure 2 has the same resolution as the viewport, i.e., 512×512 pixel, which allows to generate shadows without introducing aliasing artifacts (see Figure 8).

In comparison to shadow rays, shadow mapping is expected to run faster, since the light source visibility does not need to be recalculated for each sample, but for each light ray only. This results in the fact that no semi-transparent occluders can be supported. However, the benefit of shadow mapping is that soft shadows can be approximated by exploiting percentage closer filtering [RSC87]. This is demonstrated in Figure 3, where the visible human torso data set is rendered with both, hard shadows (left) and fake soft shadows (right). Furthermore, when using shadow mapping the combination with polygonal models can be supported much more easily, since the geometry can also be represented within the shadow map.

3.3 Deep Shadow Maps

As mentioned above, deep shadow maps allow to capture shadows of semi-transparent occluders. For demonstration purposes we have applied our implementation to a simple synthetic scene consisting of $128 \times 128 \times 128$ voxel (see Figure 4). The blue ball, which is set to be semi-transparent by modifying the transfer function, casts a *transparent* shadow on the back wall. The difference to the shadow of the opaque purple box can be noticed, especially when looking at the shadow borders.

In our implementation the deep shadow map consists of eight layers, which are shown on the right in Figure 4. Similar to the approach proposed in [HKSB06], we exploit current graphics hardware in order to generate the shadow layers on-the-fly by using multiple render targets. Although current graphics hardware allows to address up to eight rendering targets in a sin-

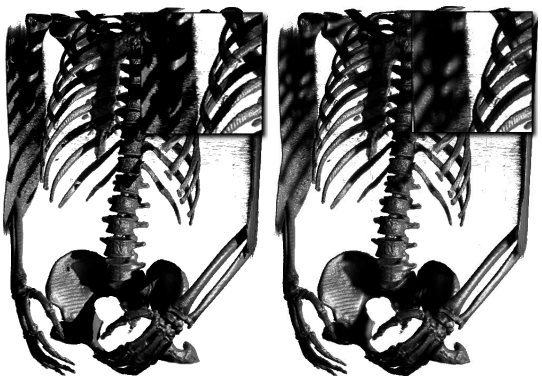

Figure 3: The visible human torso data set ($256 \times 256 \times 512$ voxel) rendered with hard shadows (left) and with fake soft shadows by using shadow mapping exploiting percentage closer filtering (right).

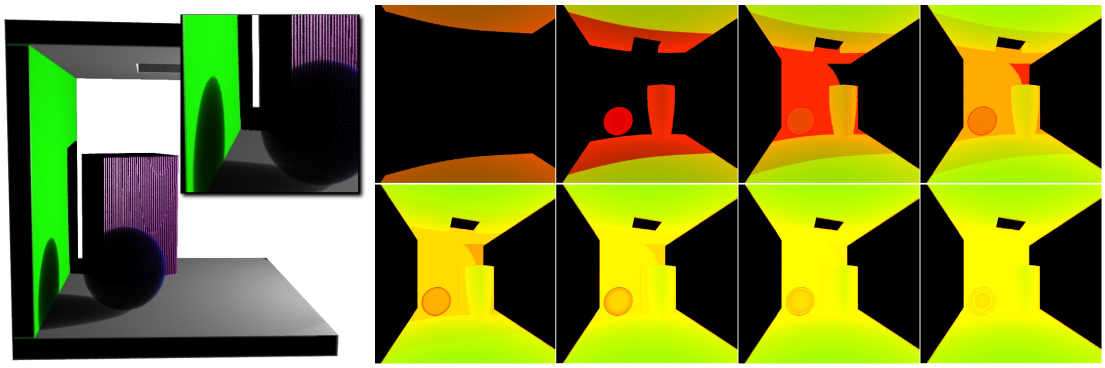

Figure 4: A synthetic scene ($128 \times 128 \times 128$ voxel) rendered using deep shadow mapping. The shadow of the semi-transparent blue ball is correctly captured. The images on the right show the successive layers of the deep shadow map.

gle pass, we only use four rendering targets and create eight deep shadow layers by performing simple channel splitting. While the original deep shadow map approach [LV00] stores the overall light intensity in each layer, we store the absorption given by the accumulated alpha value in analogy to the volume rendering integral. For each shadow ray, we analyze the alpha function, i.e., the function describing the absorption, and approximate it by using linear functions. Unlike the implementation described in [HKSB06] we restrict the depth interval approximated by a linear function to a maximum number of samples, in our case 255 samples. This is necessary in order to have a termination criterion for the fragment shader which performs the processing. However, when it is determined that the currently analyzed voxels cannot be approximated sufficiently by a linear function, smaller depth intervals are considered. Thus, the approximation works as follows. Initially, the first hit point for each shadow ray is computed, similar as done for the shadow mapping described above. Next, the distance to the light source of the first hit point and the alpha value for this position are stored within the first layer of the deep shadow map. Since we are at the first hit position, the alpha value usually equals zero when the shadow threshold is set accordingly. Starting from this first hit point, we traverse each shadow ray and check iteratively wether the samples encountered so far can be approximated by a linear function. In case this cannot be done, the distance of the previous sample to the light source as well as the accumulated alpha value at the previous sample are stored in the next layer of the deep shadow map. This is repeated until all eight layers of the deep shadow map have been created.

The light source distance and the alpha value are stored in two successive channels, i.e., R and G as well as B and A. Thus, we can represent two shadow layers by using only one *RGBA* texture. However, for illustration purposes, we wrote these values into the R and G channels when generating the pictures shown in Figure 4.

To determine wether the currently analyzed samples can be approximated by a linear function an error value is introduced. In analogy to the original deep shadow mapping technique [LV00], this error value constrains the variance of the approximation. This is done by adding (resp. subtracting) the error value at each sample's position. When the alpha function does not lie anymore within the range given by the error value, a new segment to be approximated by a linear function is started. The effect of choosing a too small error value is shown in Figure 5. As it can be seen, a small error value results in a too close approximation, and the eight layers are not sufficient anymore to represent shadow rays having a higher depth complexity. Thus especially in regions, were the two occluders both intersect a shadow ray, shadow artifacts appear. Obviously this drawback can be avoided by introducing additional shadow layers. This would allow a more precise approximation of the shadow function, but would also result in decreased rendering performance since additional rendering passes are required.

Similar to the previous techniques, we have combined our deep shadow map algorithm with the generation of a shadow volume. Thus, when shadows do not change, the shadow values can be directly fetched from a 3D texture for each sample.

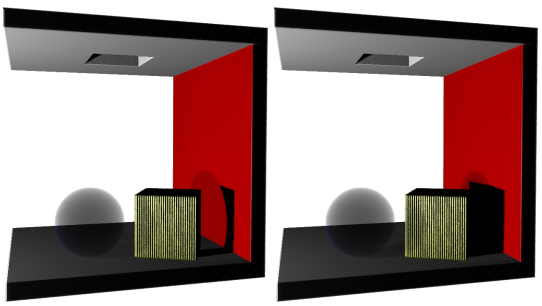

Figure 5: Different error values for deep shadow mapping: 0.00005 (left), 0.01 (right). Artifacts appear when using too small error values.

Shadow Mode	RC	without RC
shadow rays (B)	7.98	7.98
shadow rays (A)	6.70	6.70
shadow rays (B + PP)	4.25	14.90
shadow mapping (B)	12.50	15.00
deep shadow map (A)	10.00	14.90
deep shadow map (A + PP)	8.85	14.90

Table 1: Average frame rates for the different techniques as captured using a GeForce 8800GTX graphics board without optimized graphics settings.

Shadow Mode	RC	without RC
shadow rays (B)	10.03	10.03
shadow rays (A)	10.00	10.00
shadow rays (B + PP)	5.59	46.00
shadow mapping (B)	29.08	45.50
deep shadow map (A)	15.76	34.50
deep shadow map (A + PP)	13.06	45.20

Table 2: Average frame rates for the different techniques as captured using a GeForce 8800GTX graphics board with optimized graphics settings.

4 PERFORMANCE AND RESULTS

To compare not only the capabilities and limitations of each discussed shadowing technique, we have also conducted performance tests. We have scripted an animation which lets the virtual camera rotate around the engine data set shown in Figure 1. The data set has a resolution of $256 \times 256 \times 128$ voxel and has been rendered on a standard desktop computer, having an Intel Core2 CPU 6600 running at 2.40 GHz, 2 GB of main memory, and an nVidia GeForce 8800GTX graphics board.

The results of our performance tests are shown in Table 1 and Table 2; Table 1 shows the frame rates when the graphics drivers are tuned to achieve maximum quality, and Table 2 shows the results when setting the driver to maximum performance. Since this option allows accelerated texture fetches, a vast performance gain can be achieved, though major display artifacts could not be noticed (see Table 2).

The different implementations compared in the tables are three versions of shadow rays, shadow mapping as well as two versions of deep shadow maps. The identifier *A* denotes that semi-transparent occluders with an alpha value unequal to 1 have been supported, while *B* means only opaque occluders are supported. *PP* means that the shadow values are only computed for the first frame and then stored in a 3D texture, which is later on fetched during rendering of successive frames. This *caching* mechanism can be used for rendering successive frames, when only the camera position is changed. *RC* means that shadow computation is done for every frame, i.e., also when neither the light source nor an object has been modified. Thus, the frame rates shown in the column *RC* are the ones to be expected, when light source and/or objects are changed, while the column *without RC* shows the frame rates when for instance only the camera is modified and shadows do not change. This has only an impact on those techniques in which a data structure is used to store the *shadowing state*, but not for the shadow ray traversal in which no knowledge of prior frames can be exploited during rendering.

The results indicate that confining shadow rays to support only opaque occluders does not have a major impact on the frame rates. However, using render to 3D texture functionality in order to generate a 3D shadow lookup texture has a major influence on the performance. When recomputing this 3D shadow texture for each frame, it slows down the system significantly. This is due to the fact that the render to 3D texture functionality only allows to render into one slice of the 3D texture after another. However, when no frequent recomputation is required, the frame rates can be improved when using *PP*. Therefore it should be investigated in how far the shadow recomputation might be predictable for certain cases in order to decide whether the 3D shadow texture should be generated or not. As shown in the tables, shadow mapping allows frame rates which are superior to those achieved by shadow rays. This is due to the fact that a 3D texture fetch is an expensive operation on current graphics boards. Thus, when only opaque occluders are supported, shadow mapping is definitely the technique of choice. When semi-transparent occluders should be taken into account, deep shadow maps also allow a performance gain. However, generating 3D shadow textures does not seem to be as beneficial as when using shadow rays. The reason for this is probably the fact that shadow rays have to operate on 3D textures anyway, while our deep shadow mapping implementation uses 2D textures only when *PP* is disabled.

A comparison of shadow mapping and deep shadow maps with both opaque and semi-transparent occluders is shown in Figure 6. In the Figures 7-10 a visual comparison of the discussed shadowing techniques is shown side by side for different data sets. Compared to the unshadowed images, using shadows results in a higher degree of realism and objects appear less flat. While the shadow rays and the deep shadow maps allow to capture shadows of semi-transparent occluders, shadow mapping supports opaque occluders only. As it can be seen especially in Figure 8 and in Figure 9, deep shadow maps introduce artifacts when thin occluder structures are present. The shadows of these structures show transparency effects although the objects are opaque. This results from the fact that an approximation of the alpha function is exploited. Espe-

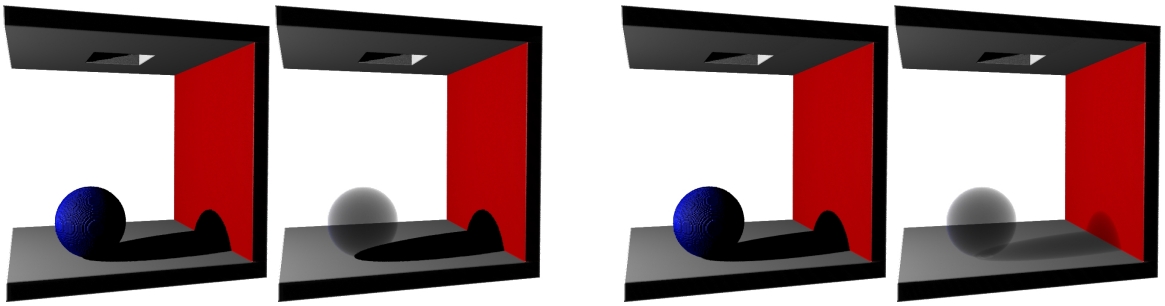

Figure 6: Volume rendering of a synthetic scene ($256 \times 256 \times 256$ voxel) with shadow mapping (left) and deep shadow mapping (right). The transfer function has been changed for both techniques in order to change the ball from being opaque to semi-transparent. Shadow mapping is not sufficient to correctly mimic shadows of semi-transparent shadow casters.

cially thin structures may diminish when approximating over too long depth intervals.

5 CONCLUSIONS AND FUTURE WORK

In this paper we have compared different techniques for combining shadows with GPU-based raycasting. As expected, we were able to show that the most obvious way for generating shadows, i.e., casting shadow rays as done in ray tracing systems, is by far not the optimal technique. However, besides deep shadow mapping it is the only technique which is capable of plausibly capturing shadows cast by semi-transparent objects. We have proposed an efficient implementation of deep shadow maps, which is sufficient to generate the deep shadow map layers in a single rendering pass. Since deep shadow mapping depends on the chosen error values, which are data set dependent, it also can not be considered as the best solution for the general case. Especially when fine structures should cast shadows, deep shadow mapping requires additional effort to provide sufficient results. However, when using the performance improvements proposed in this paper and carefully choosing the error value, deep shadow mapping can produce convincing results.

Because of the identified drawbacks, further research on interactive shadowing techniques is needed in the future. It should be examined in how far deep shadow mapping can be extended to support rather fine structures casting shadows without introducing visual artifacts. Furthermore, it should be investigated in how far a combination of the existing shadowing techniques might help in order to simulate shadows for a wider range of applications. A rather simple extension to the proposed deep shadow mapping technique would be the integration of colored shadows.

Since a lot of research has targeted shadow algorithms for polygonal data sets, it should be examined in how far these concepts can be transferred to volume data sets. For instance, rendering of soft shadows for volumetric data sets has not yet been addressed.

ACKNOWLEDGMENTS

This work was partly supported by grants from the Deutsche Forschungsgemeinschaft (DFG), SFB 656 MoBil Münster, Germany (project Z1). The presented concepts have been integrated into the Voreen volume rendering engine (`www.voreen.org`). The Visible Human is an anatomical data set licensed from the National Library of Medicine and the NCAT data set has been generated using the 4D NCAT phantom.

REFERENCES

[BR98] Uwe Behrens and Ralf Ratering. Adding shadows to a texture-based volume renderer. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 39–46. ACM Press, 1998.

[CD03] Eric Chan and Frédo Durand. Rendering fake soft shadows with smoothies. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 208–218, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. In *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 242–248. ACM Press, 1977.

[HKSB06] Markus Hadwiger, Andrea Kratz, Christian Sigg, and Katja Bühler. Gpu-accelerated deep shadow maps for direct volume rendering. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/Eurographics symposium on Graphics hardware*, pages 49–52, New York, NY, USA, 2006. ACM Press.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows al-

gorithms. *Computer Graphics Forum*, 22(4):753–774, dec 2003.

[KN01] Tae-Yong Kim and Ulrich Neumann. Opacity shadow maps. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 177–182, London, UK, 2001. Springer-Verlag.

[KPHE02] Joe Kniss, Simon Premoze, Charles Hansen, and David Ebert. Interactive translucent volume rendering and procedural modeling. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 109–116. IEEE Computer Society, 2002.

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

[LB00] Michael S. Langer and Heinrich H. Bülthoff. Depth discrimination from shading under diffuse lighting. *Perception*, 29(6):649–660, 2000.

[LV00] Tom Lokovic and Eric Veach. Deep shadow maps. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 385–392, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[MFS06] Gerd Marmitt, Heiko Friedrich, and Philipp Slusallek. Interactive Volume Rendering with Ray Tracing. In *Eurographics State of the Art Reports*, 2006.

[RGWE03] S. Roettger, S. Guthe, D. Weiskopf, and T. Ertl. Smart hardware-accelerated volume rendering. In *Procceedings of EG/IEEE TCVG Symposium on Visualization VisSym '03*, pages 231–238, 2003.

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 283–291. ACM Press, 1987.

[SSMK05] Mirko Sattler, Ralf Sarlette, Thomas Mücken, and Reinhard Klein. Exploitation of human shadow perception for fast shadow rendering. In *APGV '05: Proceedings of the 2nd symposium on Applied perception in graphics and visualization*, pages 131–134. ACM Press, 2005.

[WFMS05] Ingo Wald, Heiko Friedrich, Gerd Marmitt, and Hans-Peter Seidel. Faster isosurface ray tracing using implicit kd-trees. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):562–572, 2005. Member-Philipp Slusallek.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274. ACM Press, 1978.

[WKB+02] Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. Interactive global illumination using fast ray tracing. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 15–24, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[ZC03] Caixia Zhang and Roger Crawfis. Shadows and soft shadows with participating media using splatting. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):139–149, 2003.

[ZXC05] C. Zhang, D. Xue, and R. Crawfis. Light propagation for mixed polygonal and volumetric data. In *CGI '05: Proceedings of the Computer Graphics International 2005*, pages 249–256, Washington, DC, USA, 2005. IEEE Computer Society.

Figure 7: The NCAT phantom data set ($128 \times 128 \times 128$ voxel) rendered without shadows, with shadow rays (A), with shadow mapping and with deep shadow maps (from left to right).

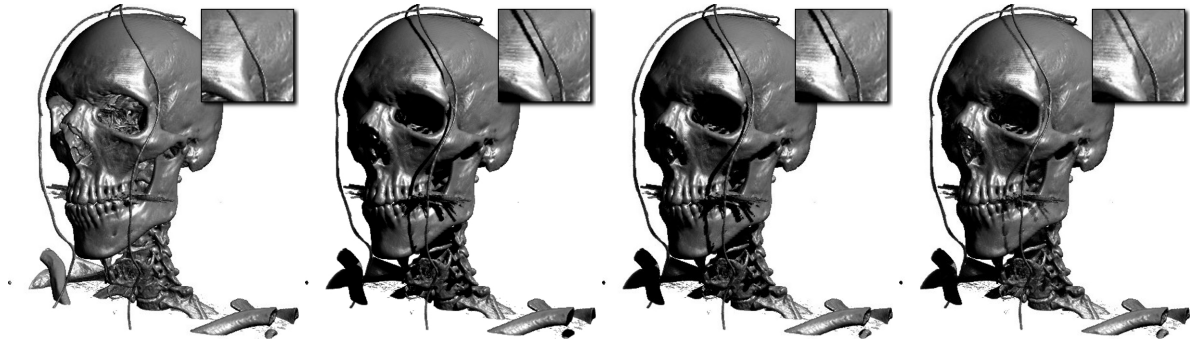

Figure 8: The visible human head data set ($512 \times 512 \times 294$ voxel) rendered without shadows, with shadow rays (A), with shadow mapping and with deep shadow maps (from left to right). The generated shadow map is shown in Figure 2.

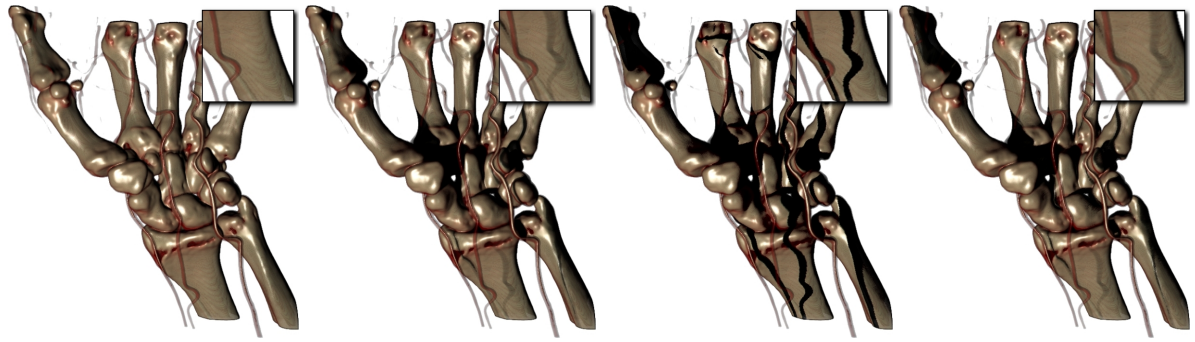

Figure 9: The hand data set ($244 \times 124 \times 257$ voxel) rendered without shadows, with shadow rays (A), with shadow mapping and with deep shadow maps (from left to right). Semi-transparent shadows become visible when using shadow rays or deep shadow maps.

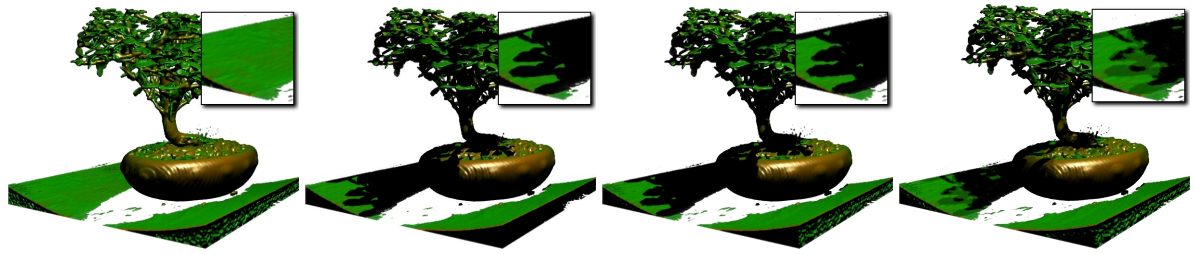

Figure 10: The bonsai data set ($256 \times 256 \times 256$ voxel) rendered without shadows, with shadow rays (A), with shadow mapping and with deep shadow maps (from left to right).