Graphics Hardware (2006)
M. Olano, P. Slusallek (Editors)

GPU-Accelerated Deep Shadow Maps for Direct Volume Rendering

Markus Hadwiger Andrea Kratz Christian Sigg[*] Katja Bühler

VRVis Research Center [*] ETH Zürich

Abstract

Deep shadow maps unify the computation of volumetric and geometric shadows. For each pixel in the shadow map, a fractional visibility function is sampled, pre-filtered, and compressed as a piecewise linear function. However, the original implementation targets software-based off-line rendering. Similar previous algorithms on GPUs focus on geometric shadows and lose many important benefits of the original concept. We focus on shadows for interactive direct volume rendering, where shadow algorithms currently either compute additional per-voxel shadow data, or employ half-angle slicing to generate shadows during rendering. We adapt the original concept of deep shadow maps to volume ray-casting on GPUs, and show that it can provide anti-aliased high-quality shadows at interactive rates. Ray-casting is used for both generation of the shadow map data structure and actual rendering. High frequencies in the visibility function are captured by a pre-computed lookup table for piecewise linear segments. Direct volume rendering is performed with an additional deep shadow map lookup for each sample. Overall, we achieve interactive high-quality volume ray-casting with accurate shadows. To conclude, we briefly describe how semi-transparent geometry such as hair could be integrated as well, provided that rasterization can write to arbitrary locations in a texture. This would be a major step toward full deep shadow map functionality.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

1. Introduction

Although the computation of accurate shadows is not trivial, they are very important in order to provide intuitive visual cues for depth and shape. Regular shadow maps sample the depth of the occluder closest to the light source, and thus the corresponding step in the visibility function from fully visible to fully occluded. For semi-transparent objects and media, however, the visibility is not a simple step function but decreases monotonically as the objects absorb more and more of the light originally emitted by the source. For thin structures such as hair, the shadow map needs to be pre-filtered to avoid aliasing, which also leads to visibility functions with a fractional visibility for each depth. Deep shadow maps [LV00] represent a visibility function for each pixel in the shadow image plane, corresponding to the light attenuation at any possible depth. Visibility is approximated and stored in compressed form as a piecewise linear function. The nodes of the approximation are sorted for efficient lookup during rendering. For each fragment position, the light attenuation can be read from the corresponding position in the deep shadow map. The (x,y)-position is determined by the shadow image plane, while the z-position has to be searched within the sorted list of sample nodes. The algorithms for construction and lookup of deep shadow maps have been designed for off-line rendering and are inherently software-based. The varying number of sample nodes across the shadow plane as well as the heap data structure employed for depth peeling during construction prohibit a simple and efficient implementation on GPUs.

Our goal is dynamic high-quality shadows for GPU-based volume ray-casting with a minimal amount of memory usage. GPU ray-casting is becoming more and more popular for direct volume rendering due to its superior image quality, efficiency, and flexibility in comparison to slice-based volume rendering. For example, efficient empty space skipping and early ray termination are much simpler to achieve, and adaptive sampling rates are easy to implement [RGW*03].

In this paper, we demonstrate that deep shadow maps for volumetric datasets can be implemented on GPUs and yield high-quality anti-aliased shadows for GPU-based ray-casting. We employ a blocked memory layout to efficiently store the deep shadow map in a 3D texture. Both deep shadow map construction and rendering are based on ray-casting, including empty space skipping and early ray termination. In order to accurately capture discontinuities in the visibility function by the discrete sampling process, the visibility between two possible successive samples is pre-computed and *pre-compressed*. This idea is similar to pre-integrated volume rendering [EKE01], but instead of only storing one integral between two samples, our lookup table stores nodes of the corresponding visibility function. This allows one to effectively capture discontinuities, e.g. from isosurfaces, during deep shadow map generation with moderate sampling rates.

2. Related Work

An early approach for volumetric shadows [BR98] slices the volume and pre-computes and stores shadow information on a regular 3D grid, e.g. the voxel positions of the original volume itself. It can be used with ray-casting for rendering, but consumes a significant amount of memory even for medium quality shadows. Shadows can also be computed with half-angle slicing, either by using simultaneous slicing for rendering and shadow computation [KPHE02], or combined with splatting [ZC03]. Half-angle slicing re-computes the shadow every time the volume is rendered, which we want to avoid, and cannot be combined with ray-casting.

Deep shadow maps [LV00] store a representation of the shadow volume in light space. For each pixel of the map, the fractional visibility function is sampled, pre-filtered, compressed and stored for fast access with spatial coherence. Current similar GPU implementations exclusively target rendering of geometry. Opacity shadow maps [KN01] generate a regular sampling of the visibility function, which is stored in texture maps without compression. In a way, this is more similar to storing shadow information on a volume grid [BR98] than to deep shadow maps. Irregular sampling can be achieved by exploiting the fact that hair strands result in clustered attenuation, and sampling the visibility in histogram bins corresponding to these clusters [MKBR04]. In practice, this is restricted to a small number of bins and cannot be used for direct volume rendering.

3. Algorithm

We sample the visibility function in light space with a ray-casting approach. Pre-filtering and compression are carried out in a fragment program. The sequence of nodes (vertices) comprised of a corresponding *(depth,opacity)* pair each are written to multiple frame buffers and then copied into a 3D texture, where the third texture coordinate represents the index of each node. Because of hardware limitations, a maximum of eight nodes can be generated in each run of the fragment program. Once eight nodes have been determined, the fragment program stops execution and a new pass picks up at the sampling position of the last node to generate the next eight nodes. The visibility function is captured within a specified accuracy [LV00], and thus the total number of nodes depends on the pixel location. For example, rays which do not hit the volume do not require any nodes while rays which intersect multiple structures require a large number of nodes.

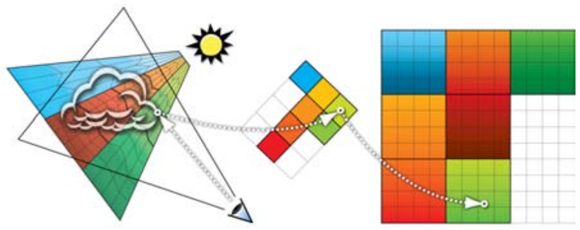

Figure 1: *The deep shadow map is constructed in a tile by tile manner in light coordinates. Blocks are dynamically allocated to pack the generated nodes into the 3D texture that constitutes the shadow map. An additional low-resolution layout texture references the storage position of each block.*

Thus, a direct mapping of the sampling nodes into a 3D texture results in a significant memory overhead, because the number of texture layers has to match the size of the longest sequence of nodes. Instead, we dynamically allocate blocks in the 3D texture to store stacks of samples as they are generated by the ray-casting process. For this, the shadow plane is partitioned into tiles and a constant number of consecutive sample layers of one tile are stored in a block of the texture. An additional layout texture references the storage location of each block in the packed deep shadow texture, which can be used as in bricked volume ray-casting [HSS*05], where often many bricks are inactive. This is illustrated in Figure 1.

In order to employ fast empty space skipping for completely transparent areas during both shadow map construction and volume rendering, the original volume is also partitioned into bricks, and a data structure containing the density range of each brick is maintained. Rays are started and stopped as close to non-transparent regions as possible by rasterizing the bounding geometry of these bricks [HSS*05]. A screen-aligned quad then initiates the sampling process, which reads the sampling segment from the two buffers previously generated. Additional empty space inside the bounding geometry is implicitly skipped during compression of the visibility function.

3.1. Ray Propagation and Pre-Filtering

The deep shadow map data structure is computed via multi-pass ray-casting from the light source's point of view. Each pass computes a set of new nodes for each pixel in the deep shadow map. The sampling process starts at the bounding geometry of non-empty bricks for the first pass, or at the previous node position for all consecutive passes. Pre-filtering is carried out by combining the samples of multiple sub-pixel rays which are cast by the same fragment program. An opacity transfer function maps the density sampled from the volume texture to light attenuation, i.e. opacity. Using a constant sampling rate of the visibility function, discontinuities in the transfer function lead to aliasing in the shadow map, which can only be mitigated by high sampling rates. In order to avoid these sampling artifacts, we pre-compute a table of pre-compressed visibility functions.

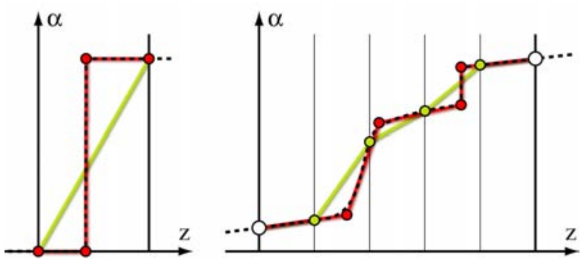

Figure 2: *The volume density at the borders of a linear segment is used to lookup a pre-compressed visibility function. The interpolation nodes (red / dark gray) of the lookup table approximate the visibility function (dashed line) much better than regular sampling (green / light gray) because they can be placed arbitrarily within the segment.*

3.2. Pre-Compressing Transfer Function Frequencies

The main idea of pre-integrated volume rendering [EKE01] is to pre-compute the volume rendering integral for all possible combinations of two successive density samples. Instead of storing only a single integral per sample pair, we pre-compute and compress the visibility function of the segment between two successive density samples using the same compression method used for deep shadow map generation itself [LV00]. This effectively decouples the sampling rate during deep shadow map construction from the frequencies in the transfer function. The sampling rate is chosen according to the volume, and the influence of high frequencies in the opacity transfer function is captured by *pre-compression*. Figure 2 (left) illustrates this idea for capturing a step function due to an isosurface transfer function. Two successive samples one sampling distance apart completely miss the discontinuity which results in a very inaccurate visibility function and finally leads to self-shadowing of isosurfaces as illustrated in Figure 3 (left). The pre-compression table stores the nodes of a compressed visibility function for each possible combination of successive samples. The red nodes in Figure 2 (left) accurately capture the isosurface, which leads to the results shown in Figure 3 (right).

Figure 2 (right) illustrates the difference between simply using a higher sampling rate (green nodes) in-between samples of a lower sampling rate (white nodes), and the pre-compressed visibility function with the same number of nodes (red). Note that the sampling rate used for computing the pre-compressed visibility functions can be chosen according to the transfer function domain. Hence it is as high as necessary to capture all transitions [EKE01]. As a result of the optimal placement of the nodes (red versus green), the discontinuity comprising the fourth red segment in Figure 2 (right) accurately captures a semi-transparent isosurface. Also the locations of all other nodes capture the real visibility function more accurately. The only trade-off we make is the maximum number of nodes in a pre-compressed visibility function, which is set to a small constant, e.g. four. Two nodes of a pre-compressed function can be stored in

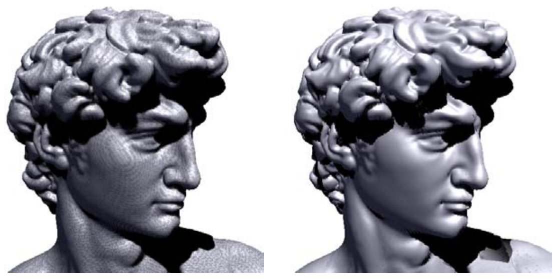

Figure 3: *Capturing high transfer function frequencies with pre-compression. Visibility functions of isosurface discontinuities computed with equi-distant samples result in erroneous self-shadowing (left), which is removed by using accurate pre-computed nodes (right). Both images use the same number of samples/nodes during deep shadow map creation.*

a single 2D lookup table. Overall, the required number of nodes is much smaller than using a high number of equidistant samples in order to achieve similar results.

3.3. Compression During Deep Shadow Map Creation

We employ the original compression algorithm [LV00] in the fragment program, which reduces the number of interpolation nodes written to the frame buffer and copied into the deep shadow map. This greedy algorithm combines multiple successive visibility nodes by approximating more and more nodes by a linear function until a specified error bound would be violated. Then, the end point of the current linear segment is stored and a new segment is started. Each resulting node consists of a *(depth,opacity)* pair. We store two successive nodes in the RG an BA components of the shadow map texture. With a maximum of four render targets on current GPU architectures, this implies that we can compute and write out eight successive nodes in a single rendering pass.

3.4. Memory Management

The total number of nodes required to approximate the visibility function varies from pixel to pixel. Thus, storing the nodes directly in layers of a 3D texture would be highly inefficient, because only the first few layers would be fully utilized. Instead, we pack the nodes into a texture using a block layout. Each block corresponds to, e.g. eight, consecutive layers for one tile of the shadow map. During shadow map generation, the blocks are dynamically allocated for each tile and the storage position is referenced in an additional low-resolution layout texture. We use occlusion queries to determine if the construction of all visibility functions of a tile is completed either because it has reached full opacity, or the ray has exit the volume. In these cases, all fragments of the tile are discarded, which is checked by the occlusion query, and the tile is removed from further processing. Otherwise, a new block is allocated and the ray-casting process is resumed by drawing an additional tile-aligned quad. See Figure 1 for an illustration of this process.

3.5. Rendering

Actual rendering uses the same ray-casting process with empty space skipping. Each density sample is mapped to color and opacity values by a transfer function and is then modulated by the light intensity extracted from the deep shadow map. For this, the sorted *(depth,opacity)* pairs of the closest pixel in the shadow plane are searched linearly for the depth of the sample in light coordinates. This node traversal requires a lookup into the layout texture to retrieve the storage position of the current block of nodes (see Figure 1). The spatial coherence of the shadow map lookup is exploited by starting the search at the position of the last sample. Note that we do not use bilinear interpolation of four neighboring visibility values, because in order to be correct this cannot be done by hardware-native filtering. However, with some additional cost it could be done in the fragment program.

4. Results and Conclusion

Rendering of the views in Figures 4 and 6 (left) with an already constructed deep shadow map, i.e., when the light source is not moving, can be performed with 3-6 fps on an ATI Radeon X1800 (512x512 viewport). Times for shadow map construction with different resolutions are shown in Table 1. A lower resolution and higher error bound during interaction allow moving the light source with 1-2 fps including rendering. A major performance factor is the number of nodes in the resulting compressed visibility function, since it directly determines the number of rendering passes and layers that must be copied to the 3D shadow map texture.

Our implementation has shown that high-quality anti-aliased deep shadow maps for direct volume rendering can be computed and rendered on current GPUs. Shadow map quality is improved considerably by using pre-compressed visibility functions. Our deep shadow map representation could be extended to additionally support semi-transparent geometry such as hair, when fragments can be written to arbitrary locations in a 3D texture during rasterization. Each *(depth,opacity)* pair could be extended with a link field that allows rasterizing additional geometry "in-between" already written visibility nodes. Alternatively, nodes could be computed out of order and then sorted into depth order in a post-processing step.

The VRVis Research Center is funded in part by the Austrian Kplus project. The medical data are courtesy of Tiani MedGraph. The David model is courtesy of the Digital Michelangelo Project.

[fps]	**128x128**	**256x256**	**512x512**
$\varepsilon = \mathbf{0.02}$	2–3	1–2	0.3–0.5
$\varepsilon = \mathbf{0.06}$	5–6	2–3	1–1.5

Table 1: *Shadow map computation times for the setting shown in Figure 4 with different map resolutions and error bounds ε for compression. A larger ε does not decrease the number of samples during shadow map construction, but decreases the resulting number of nodes and thus the number of rendering passes and time for copying to the 3D texture.*

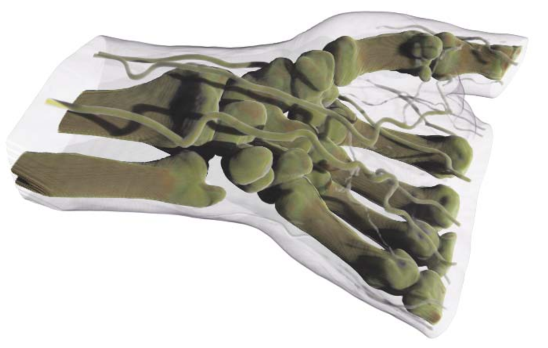

Figure 4: *CT scan of a human hand with direct volume rendering and shadowing. Shadow map resolution is 512x512 with a 5-tap pre-filter kernel.*

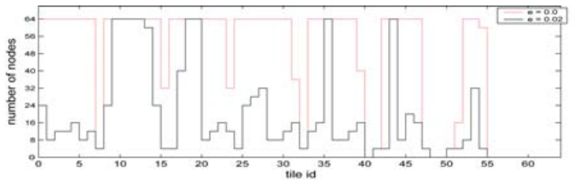

Figure 5: *The number of nodes in the visibility function varies per pixel and thus per tile (here: 16x16 pixels) in the shadow map plane. Many tiles require only a small number of nodes, depending on the error bound for compression (ε).*

References

[BR98] BEHRENS U., RATERING R.: Adding shadows to a texture-based volume renderer. In *Proceedings of IEEE VolVis* (1998), pp. 39–46.

[EKE01] ENGEL K., KRAUS M., ERTL T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proc. of Graphics Hardware* (2001), pp. 9–16.

[HSS*05] HADWIGER M., SIGG C., SCHARSACH H., BÜHLER K., GROSS M.: Real-time ray-casting and advanced shading of discrete isosurfaces. In *Proc. of Eurographics* (2005), pp. 303–312.

[KN01] KIM T.-Y., NEUMANN U.: Opacity shadow maps. In *Proc. of Eurographics Symposium on Rendering* (2001), pp. 177–182.

[KPHE02] KNISS J., PREMOZE S., HANSEN C., EBERT D.: Interactive translucent volume rendering and procedural modeling. In *Proc. of IEEE Visualization* (2002), pp. 109–116.

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. In *Proc. of ACM Siggraph* (2000), pp. 385–392.

[MKBR04] MERTENS T., KAUTZ J., BEKAERT P., REETH F. V.: A self-shadow algorithm for dynamic hair using density clustering. In *Proc. of Eurographics Symposium on Rendering* (2004), pp. 173–178.

[RGW*03] RÖTTGER S., GUTHE S., WEISKOPF D., ERTL T., STRASSER W.: Smart hardware-accelerated volume rendering. In *Proc. of VisSym* (2003), pp. 231–238.

[ZC03] ZHANG C., CRAWFIS R.: Shadows and soft shadows with participating media using splatting. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 139–149.

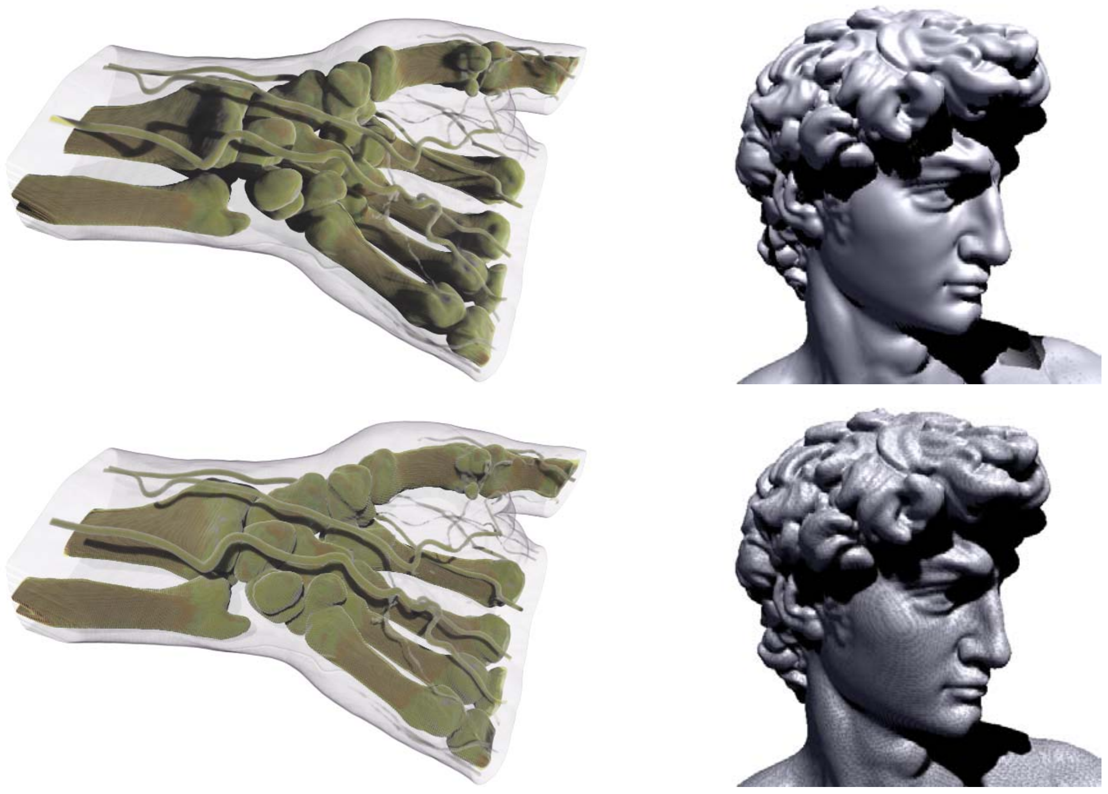

Figure 6: *Renderings of a CT scan of a human hand with direct volume rendering and shadowing with deep shadow maps (left images). Shadow map resolution is 512x512 and a 5-tap pre-filter kernel has been used. The right images illustrate capturing high transfer function frequencies with pre-compression. Visibility functions of isosurface discontinuities computed with equidistant samples result in erroneous self-shadowing (bottom), which is removed by using accurate pre-computed nodes (top).*