Clustered Deep Shadow Maps for Integrated Polyhedral and Volume Rendering

Alexander Bornik[1], Wolfgang Knecht[2],
Markus Hadwiger[3], and Dieter Schmalstieg[4]

[1] Ludwig Boltzmann Institute – Clinical-Forensic Imaging (CFI), Graz, Austria
alexander.bornik@cfi.lbg.ac.at
[2] Vienna University of Technology, Vienna, Austria
[3] King Abdullah University of Science and Technology, Thuwal, Saudi Arabia
[4] Graz University of Technology, Graz, Austria

Abstract. This paper presents a hardware-accelerated approach for shadow computation in scenes containing both complex volumetric objects and polyhedral models. Our system is the first hardware accelerated complete implementation of deep shadow maps, which unifies the computation of volumetric and geometric shadows. Up to now such unified computation was limited to software-only rendering . Previous hardware accelerated techniques can handle only geometric or only volumetric scenes - both resulting in the loss of important properties of the original concept. Our approach supports interactive rendering of polyhedrally bounded volumetric objects on the GPU based on ray casting. The ray casting can be conveniently used for both the shadow map computation and the rendering. We show how anti-aliased high-quality shadows are feasible in scenes composed of multiple overlapping translucent objects, and how sparse scenes can be handled efficiently using clustered deep shadow maps.

1 Introduction

To address the need for shadows cast by semi-transparent objects, the deep shadow map (DSM) [1] extends the concept of a conventional shadow map by storing a visibility function for each pixel in the shadow image plane. Hadwiger et al. [2] present a GPU implementation of DSM that is suitable for direct volume rendering, but cannot accommodate polygonal or polyhedral geometry, and does not scale well to large, sparse scenes.

Our goal is the integration of dynamic high-quality shadows in a hardware-accelerated rendering framework supporting both polygonal representations and volumetric objects. This has important applications in creating realistic scenes for games and virtual environments, but also in medical applications. We show that a hardware-accelerated DSM implementation on the GPU is feasible based on the Compute Unified Device Architecture (CUDA) [3]. Deep shadow map computation and rendering are both based on ray casting. The ray casting considers segments along each ray, which are homogeneous in terms of object

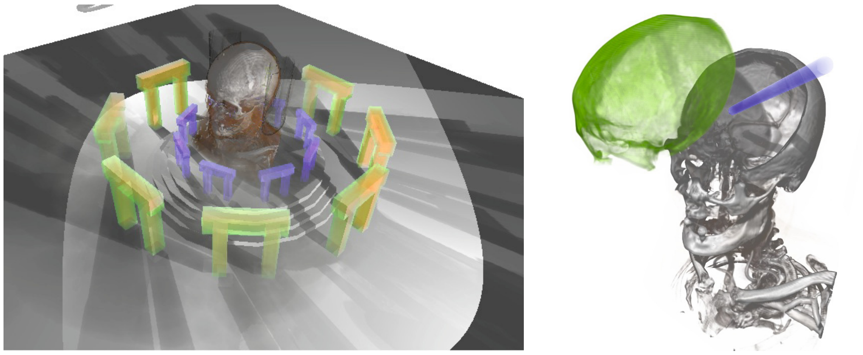

Fig. 1. Examples of deep shadow map rendering in hybrid volumetric/geometric scenes: The left image shows shadow interaction in a complex scene consisting of opaque and translucent geometry with transparent interior material as well as volumetric smoke and a CT dataset. The image on the right shows benefits of deeps shadow maps for depth perception in surgical simulation, showing a polygonal surgery tool (blue rod).

occupancy. Volume rendering acceleration techniques such as early ray termination and empty space skipping allow for interactive frame rates.

This approach yields high-quality anti-aliased shadows and has the following distinguishing properties:

– Arbitrary combinations of *soft shadow casting between polygonal and volumetric objects* are possible. Polygonal objects can be transparent and overlapping with volumetric objects.
– Perspective aliasing of shadows in large, sparse scenes due to insufficient DSM resolution is effectively suppressed through a novel concept, *clustered deep shadow maps*. This approach analyzes the scene structure and adaptively allocates the nodes of the deep shadow map structure where they contribute most. We also present a hardware-friendly chunked memory layout for clustered deep shadow maps.

2 Related Work

The most common methods for computing shadows in geometric scenes are either based on shadow volumes [4] or on shadow mapping [5], both of which are originally limited to opaque geometry and hard shadows.

In contrast to shadow volumes, shadow mapping employs an image-space discretization, which makes the extension toward volume sampling easier. A major problem of shadow mapping methods are several types of aliasing, most of all perspective aliasing. Common approaches reduce perspective aliasing via specific perspective transforms, e.g., in post-projective view space [6] or light space [7], or by using logarithmic transforms [8]. Aliasing artifacts can also be reduced by using adaptive shadow map resolutions like in [9]. Aliasing due to sample positions in the shadow map can be removed entirely using irregular rasterization [10]. Perspective aliasing can also be reduced significantly via cascaded shadow maps [11] or parallel-split shadow maps [12]. However, these approaches are inefficient in sparse scenes. The filtering of shadow maps has been improved by using tailored approaches such as variance shadow maps [13], which can also be layered [14], or convolution shadow maps [15].

Most shadow mapping approaches focus on hard shadows, although soft shadows caused by area light sources can be approximated by superimposing the contribution of multiple point lights. A recent approach for colored stochastic shadow maps of translucent objects is given in [16]. However, approaches working from a surface representation obviously cannot take into account soft shadows caused by continuous absorption of light by participating media or volume data.

In volume rendering, shadow computation is an important topic [17]. The two most common approaches either perform half-angle slicing [18] on-the-fly, or compute an additional volume that stores the amount of light reaching each voxel from the light source [19], which is then used during the actual volume rendering. The latter approach can be combined with ray casting for the volume rendering pass [20], but requires significant additional volume storage. Half-angle slicing does not store any additional volume data, but is restricted to slice-based volume rendering . More recent work has taken on the basic idea for volume ray casting [21], but is still closely linked the regular dataset grid, which prohibits its application to multi-volume-grid scenes and integration with polyhedral objects. An alternative are approaches based on Monte-Carlo sampling [22].

The work in this paper builds on the Deep Shadow Map approach [1], which conceptually unifies the computation of volumetric and geometric shadows. Instead of storing a single depth value per shadow map pixel, a visibility function is computed and stored per pixel.

For volume rendering, DSMs can be implemented efficiently on GPUs [2], which yields real-time frame rates on current hardware even when combined with scattering effects [17]. However, this approach focuses on volume data and does not incorporate geometry.

Several approaches focus on achieving similar functionality for hair geometry with less computational requirements than DSMs [23][24][25][26][27]. Adaptive volumetric shadow maps [28] are an approximate solution to volumetric shadows, which offers fast on the fly lossy compression, efficient lookup, and global shadow support given a fixed map size. However, these algorithmic advantages only apply to a traditional rendering pipeline with unsorted fragments, while we use sorted fragments lists.

3 Basic Algorithm

The implementation of DSM rendering presented in this paper is derived from the ray casting system described in [29]. The whole rendering pipeline including DSM is implemented in CUDA and executes purely on the GPU. It operates on a data structure similar to a volume scene graph, which consists of a tree with volumetric Boolean operations in interior nodes and *polyhedral objects* (topologically closed triangular meshes) in the leaves. First, boundary polygons are rasterized, and all produced fragments that fall onto a given pixel location are depth-sorted. Then a ray is traversed from the frontmost fragement, and visits other boundary fragments for a given location in depth order, while accumulating opacity.

3.1 Deep Shadow Map Generation

The structure of the DSM calculation algorithm is very similar to polyhedral bounded volume rendering. In fact, it reuses a large portion of the kernel code from the ray casting and only differs in the determination of each fragment's 'payload'. Instead of sampling a color and opacity value along a ray, a piecewise linear visibility function $V(z)$ storing the remaining light intensity is calculated from the light's viewpoint as described in [1]. We also perform pre-compression in analogy to [2].

All DSM computation is performed in the per pixel kernel, right after fragment depth sorting. For each homogeneous ray segment, we compute a compressed piecewise linear representation $V'(z)$ of the discrete visibility function $V(z)$, which describes the light attenuation along a ray from the light source through the scene. Its computation is based on opacity samples and represented as a list of nodes storing depth and opacity. The compressed version $V'(z)$ is an approximation of $V(z)$ where

$$|V'(z) - V(z)| \leq \epsilon \tag{1}$$

holds for a maximum error of ϵ [1]. $V'(z)$ is computed on the fly, depending on the types of objects present in a particular ray segment. If the ray is inside of a volumetric object, standard opacity accumulation is used.

Surfaces of polyhedral objects are unconditionally added as nodes to $V'(z)$. In order to accurately represent the step in $V(z)$, two entries with the same z are added, storing the α_{acc} before and after fragment opacity comprehension. In case of more than one light source, omnidirectional light sources or clustering, multiple DSMs are computed concurrently.

3.2 Rendering Using Deep Shadow Maps

During rendering, the precomputed DSM are used to compute the amount of light penetrating to the observed point. The sample location is transformed into light space as for standard shadow mapping. The sample's depth value in light space is used to find the two closest visibility function entries in $V'(z)$, which are linearly interpolated.

Shading is done using the *Phong Illumination Model*. Rendering quality is further improved by bilinear filtering of the four surrounding texel values at lookup time. The contribution of multiple light sources is the sum of all individual contributions, which are computed using the corresponding DSM.

4 Clustering

Using a single shadow map often leads to perspective aliasing artifacts because the map resolution drops with increasing distance of scene objects from the camera. This effect is particularly strong if the current camera location is far from the camera setup used for map calculation. Furthermore, sparse scenes can easily lead to bad shadow map utilization. The above-mentioned problems are also prominent with DSMs.

We attack the aliasing problem with multiple equally-sized DSMs per light source. Each DSM represents a different part of the scene. However, unlike *parallel-split shadow maps* [12], our DSM arrangement is adaptive to the scene content: Instead of splitting the scene along the camera's view vector, scene objects are clustered. Each cluster obtains its own shadow map. Figure 2 gives an overview of the situation.

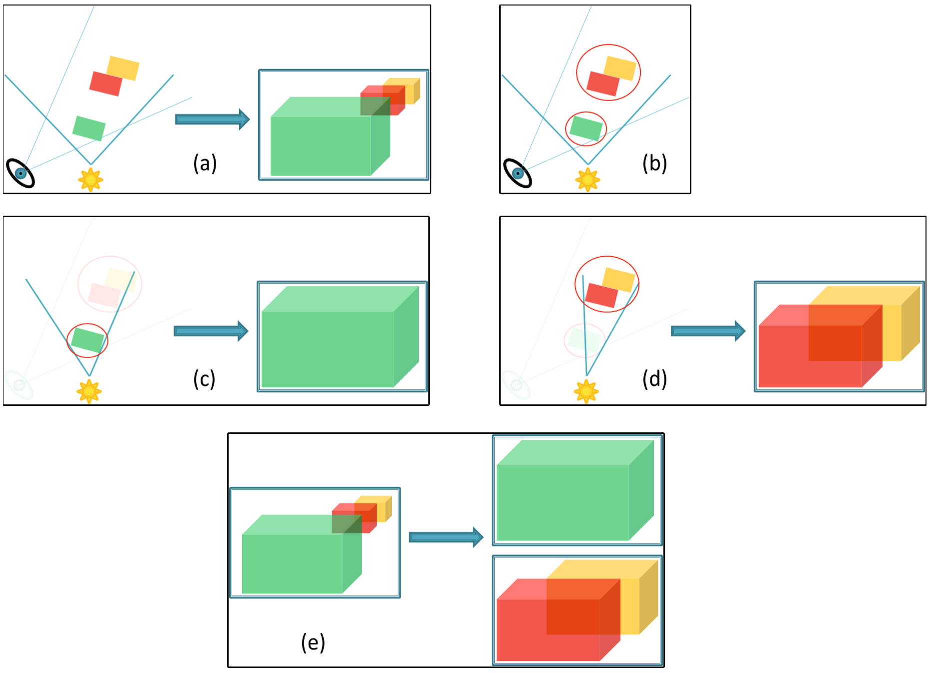

Fig. 2. Illustration of the basic idea of clustering. (a) Overview of the scene and the corresponding single shadow map. (b) Nearby objects are clustered. (c, d) Focusing of the shadow map on the first cluster and on the second cluster. (e) Single shadow map compared to the two focused shadow maps.

Clustered DSM computation is a two stage process. First, clustering groups objects with little shadow contribution into larger clusters, while objects with major shadow contribution are placed in smaller clusters. Every cluster is assigned to one DSM. The rationale is that in order to best represent the shadow with limited resources, the number of DSM entries per object should be proportional to the object's contribution to the shadow.

After cluster assignment, shadow map viewing volumes are fit to the clusters to optimize shadow map usage by adjusting the light camera's orientation, field of view, and near/far planes.

Cluster building is performed in light-space (see [7]). We consider every scene object's bounding box and derive extreme points pp_{min} and pp_{max} as the minimum and maximum projected extents in x and y. The distance between two bounding boxes A and B is defined as follows:

$$d(A,B) = \max |pp_{min}(A) - pp_{min}(B)|, |pp_{max}(A) - pp_{max}(B)| \tag{2}$$

For omnidirectional lights, $d(A,B)$ is the maximum of the values computed for six different projection directions. The resulting distances are the basis for cluster generation. We use the following heuristic based on a user defined distance threshold d_0, since we want to compute up to m cluster DSMs, where m is the maximum number of DSMs, which fits into pre-allocated shadow map storage (for details on storage management, see Section 5):

1. Sort pairs of objects O_i, O_j with $(i \neq j)$ by $d(O_i, O_j)$
2. Insert objects O_i into existing cluster C_k where $\forall O_j \in C_k : d(O_i, O_j) \leq d_0$. OR insert objects O_i into new cluster C_l if for all clusters C_k: $\exists O_j \in C_k \; with \; d(O_i, O_j) > d_0$ and $l \leq m$.
3. Insert remaining objects O_i into the cluster C_k with $\{max(d(O_i, O_j) \mid O_j \in C_k\} \rightarrow min$.

From the rendering point of view, clustered DSMs are used similarly to DSMs for multiple light sources. Their contributions are appropriately combined along the light ray. Spotlights require an inside/outside illuminated area test per cluster.

Clustering leads to local maps and good resource utilization as well as higher resolution maps for objects farther from the viewer. Re-computation is only required, if the light, the scene or the clustering changes. Note that frame-to-frame coherence means that clustering can often be re-used over multiple frames for a moving camera, and that clustering is computationally cheap compared to DSM computation. An additional advantage of *Clustered Deep Shadow Maps* is the fact that omnidirectional light sources may require fewer than six shadow maps, depending on the scene configuration.

5 Memory Management

The DSM storage scheme used in our implementation is chunked memory. A memory chunk can store a predefined number of L_C depth and opacity values (64 in our case), each pair representing a depth layer for one texel in a DSM. Furthermore each chunk entry stores a link to the global chunk index of the next chunk and the entry number in that chunk. A separate initialization buffer stores the starting chunk number and entry for each DSM texel. Values are stored in a large *float4* texture.

For chunked memory DSM computation, the viewport is split into tiles of 8×8 pixels, corresponding to blocks of 64 parallel threads in CUDA. Each block may reserve a maximum number C_B out of a total number C_{max} of available chunks. The corresponding chunk indices are stored in $b_g[b][i]$, where b is the block index and i the block chunk index. All indices are initialized to N_A (not allocated). Furthermore there is an array $b_l[b]$ used for counting the overall number of samples stored in each block. It is incremented using CUDA atomic operations.

As samples to be added to the DSM arrive, we can calculate $b_c = \frac{b_l[b]}{L_C}$, the current block's chunk number. The current chunk's global index c_d is obtained using an allocation that relies on CUDA atomic operations in order to synchronize access to shared state across threads. Waiting for another thread to exit the transient *pending* state is accomplished by busy waiting since the expected time to wait (if any) is very short (and CUDA does not offer idle wait primitives anyway).

Chunk data is written to separate lists for depth, opacity and indices at index $c_d * (b_l \bmod L_C)$ to take advantage of coalesced writes during computation, while the list elements are copied to the corresponding *float4* texture. The per texel start chunk is initialized with first value obtained when incrementing b_l for a texel and the index of the the corresponding chunk.

Chunk based memory avoids memory wastage, and the number of depth layers per DSM sample is only limited by the amount of GPU memory dedicated to DSM chunks, chunk size L_C, and management entries in $b_g[b]$. However, lack of spatial coherence requires a linear DSM list traversal to find the appropriate depth layer, which results in computational overhead. This overhead is reduced by exploiting fast access to textures for lookup structures and chunk data during rendering.

6 Results

The following results were all obtained on a 2.6GHz Intel i7 system equipped with an NVIDIA GTX 480 GPU running on Windows Vista 64Bit.

6.1 Clustering

The example in Figure 3 depicts the advantages of scene clustering. The region outlined red is far away from the light source. Therefore the shadow map resolution is low in that area, even for a high overall map resolution of 1024×1024, which is visible in the left image. Scene clustering based on the actual viewpoint for $n = 4$ clusters results in a separate DSM for the smoking cup of coffee, which provides much higher local shadow resolution. Note, that the overall memory consumption is the same, since each of the four cluster DSMs is only 512×512.

Figure 4 shows a different view of the same scene and the corresponding clustering, which again guarantees high resolution shadows maps for objects nearby the camera.

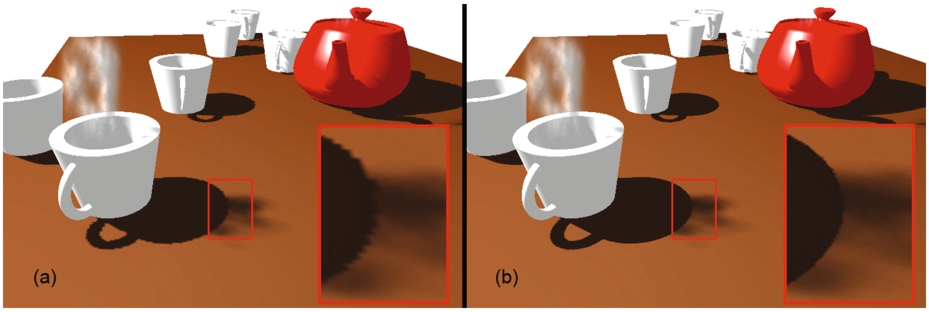

Fig. 3. (a) Scene rendered using a single deep shadow map. (b) Same scene rendered using multiple deep shadow maps computed based on scene object clustering. The difference is clearly visible in the magnified highlighted region.

6.2 Performance Analysis

We analyzed the performance of our implementation by varying a number of parameters including DSM resolution, volumetric dataset resolution, number of objects and error tolerance threshold.

In order to study the influence of DSM resolution, we rendered the scene shown in Figure 3 with different DSM resolutions and recorded map computation as well as rendering times with and without filtering. The viewport size was set to 720 $\times$ 480 pixels. The test scene consisted of one spotlight, two volumetric and eight polyhedral objects. Rendering without any shadow mapping took 43ms on the test system. The results with shadows are summarized in Table 1.

Table 1. Computation times in [ms] for varying DSM resolutions and 3D texture based memory management. Numbers in brackets are timings with clustering turned on.

[ms]	**256^2** (**4** $\times$ **128^2**)	**512^2** (**4** $\times$ **256^2**)	**1024^2** (**4** $\times$ **512^2**)
DSM generation	11 (21)	31 (40)	65 (80)
Rendering	46 (54)	46 (55)	46 (57)
Rendering (filt.)	54 (70)	53 (71)	53 (72)

The resolution of volumetric datasets has a significant impact on DSM computation times and even more on rendering. These results – summarized in Table 2 – are in line with ray casting performance in general. A higher dataset resolution also leads to a higher number of depth layers.

As expected, the DSM resolution has a significant impact on performance. Filtering slightly increases rendering times. Clustering timings are compared for four clusters and halved map resolution. Clustering leads to higher map computation and rendering times. The overhead of the n rendering passes and

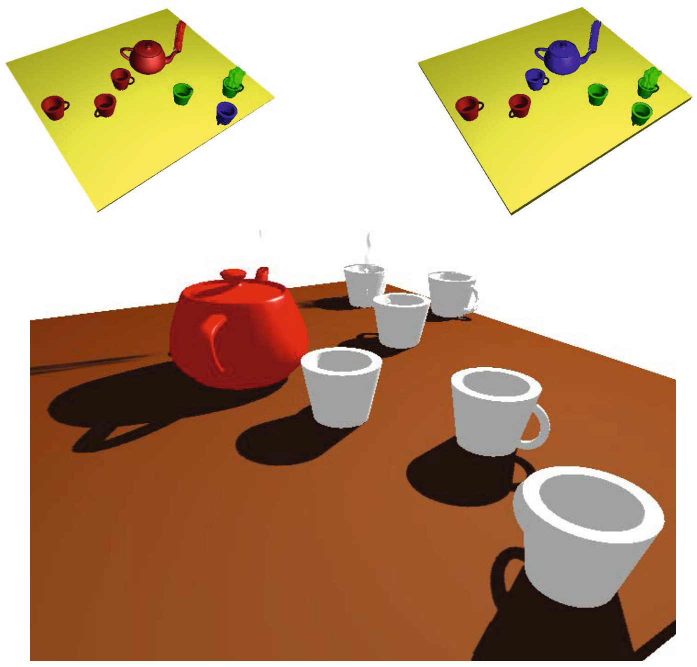

Fig. 4. Top: Clustering overview for the view from Figure 3 and for a totally different view (bottom). Objects of the same color enter the same DSM.

lookup while rendering is more costly than dealing with a higher resolution map. However, clustering is justified if demands on image quality are high.

Table 3 shows that chunk based memory management decreases DSM computation times, but slightly increases rendering times.

The choice of ϵ as error tolerance threshold for DSM pre-compression influences the number of nodes in the visibility functions as shown in Table 4. The equal maximum number of depth layers per DSM texel for $\epsilon = 0.05$, $\epsilon = 0.1$ and $\epsilon = 0.2$ reflects a scene-dependant lower bound for accurate representation of the visibility function in some region. It can only be lowered by an excessively higher ϵ, resulting in major rendering artifacts.

Rendering times and DSM computing times scale rather linearly with the number of scene objects, regardless of the memory management methods used. Chunked memory management constantly results in faster DSM computation but slightly higher rendering times. However, chunked memory has the additional advantage of lower memory requirements, which can be important for large volumetric scenes. Memory management was tested on a scene with up to 24 similar polyhedral and volumetric objects.

Since memory for DSMs has to be allocated at startup, the 3D texture management size has to be chosen based on the maximum number of depth layers required to store the DSM obtained through pre-compression using a given ϵ.

Table 2. Computation times in [ms] for varying volumetric dataset resolutions. Chunked Memory is used as the memory management method ($4 \times 256 \times 256$ DSM).

	time	Avg. # layers	Max. # layers
DSM generation			
$512 \times 512 \times 96$	74 ms	3.74	31
$256 \times 256 \times 48$	55 ms	3.64	27
$128 \times 128 \times 24$	52 ms	3.64	24
Rendering (filt.)			
$512 \times 512 \times 96$	846 ms		
$256 \times 256 \times 48$	340 ms		
$128 \times 128 \times 24$	130 ms		

Table 3. Computation times in [ms] for varying DSM resolutions and chunk based memory management. Numbers in brackets are timings with clustering turned on.

[ms]	$\mathbf{256^2}$ $(\mathbf{4 \times 128^2})$	$\mathbf{512^2}$ $(\mathbf{4 \times 256^2})$	$\mathbf{1024^2}$ $(\mathbf{4 \times 512^2})$
DSM generation	7 (17)	17 (30)	54 (79)
Rendering	48 (55)	49 (56)	49 (57)
Rendering (filt.)	56 (69)	56 (69)	57 (70)

Table 4. The number of layers / DSM texel depends on the error tolerance ϵ

	$\boldsymbol{\epsilon = 0.01}$	$\boldsymbol{\epsilon = 0.05}$	$\boldsymbol{\epsilon = 0.1}$	$\boldsymbol{\epsilon = 0.2}$
Avg. # layers	4.01	3.32	3.12	3.00
Max. # layers	28	21	21	21

For our test scene (Figure 3), this number (21 for $\epsilon = 0.05$) can be found in Table 4. With chunk based memory, giving an estimated average number of layers is sufficient. Therefore chunk based memory management reduces memory consumption by $\approx 80\%$ for the test scene.

7 Conclusion

We have shown a fully hardware-accelerated implementation of deep shadow maps for scenes combining direct volume rendering with polygonal models. This rendering system yields high quality soft shadows at interactive frame rates. It features a unified handling of volumetric and non-volumetric objects, and can also deal with large, sparse scenes due to its application of multiple deep shadow maps after scene clustering. We believe that our approach is an important step towards the adoption of volumetric shadows in everyday rendering tasks.

We are planning several extensions of the system. Special translucent geometry, in particular hair, can be supported by extending the rasterization stage.

Alternatively, the new CUDA interface in OpenGL 4.0 allows to combine the OpenGL rasterizer with the CUDA raycasting kernel. We have verified in early experiments that this approach increases framerates of typical scenes from 5 to 15-20 fps. Relying the OpenGL rasterizer also makes integration of our approach with conventional game engines much easier. Finally, an extension of our DSM approach to colored shadows inspired by [16] seems straight forward.

References

1. Lokovic, T., Veach, E.: Deep shadow maps. In: SIGGRAPH 2000: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, pp. 385–392. ACM Press/Addison-Wesley Publishing Co (2000)
2. Hadwiger, M., Kratz, A., et al.: GPU-accelerated deep shadow maps for direct volume rendering. In: GH 2006: Proc. of the 21th SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware, pp. 49–52. ACM (2006)
3. Nickolls, J., Buck, I., Garland, M.: Scalable parallel programming with CUDA. ACM Queue 6, 40–53 (2008)
4. Crow, F.C.: Shadow algorithms for computer graphics. In: SIGGRAPH 1977: Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques, pp. 242–248. ACM (1977)
5. Williams, L.: Casting curved shadows on curved surfaces. In: SIGGRAPH 1978: Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques, pp. 270–274. ACM (1978)
6. Stamminger, M., Drettakis, G.: Perspective shadow maps. In: SIGGRAPH 2002: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, pp. 557–562. ACM (2002)
7. Wimmer, M., Scherzer, D., Purgathofer, W.: Light space perspective shadow maps. In: Keller, A., Jensen, H.W. (eds.) Rendering Techniques 2004 (Proceedings Eurographics Symposium on Rendering), Eurographics, pp. 143–151 (2004)
8. Lloyd, D.B.: Logarithmic perspective shadow maps. PhD thesis, Chapel Hill, NC, USA (2007), Adviser-Manocha, Dinesh
9. Lefohn, A., Sengupta, S., Owens, J.: Resolution-matched shadow maps. ACM Trans. Graph. 26, 20 (2007)
10. Aila, T., Laine, S.: Alias-free shadow maps. In: Keller, A., Jensen, H.W. (eds.) Rendering Techniques, Eurographics Association, pp. 161–166 (2004)
11. Dimitrov, R.: Cascaded Shadow Maps. NVIDIA (2007)
12. Zhang, F., Sun, H., Xu, L., Lun, L.K.: Parallel-split shadow maps for large-scale virtual environments. In: VRCIA 2006: Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and its Applications, pp. 311–318. ACM (2006)
13. Donnelly, W., Lauritzen, A.: Variance shadow maps. In: I3D 2006: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, pp. 161–165. ACM (2006)
14. Lauritzen, A., McCool, M.: Layered variance shadow maps. In: GI 2008: Proceedings of Graphics Interface 2008, pp. 139–146. Canad. Information Processing Society (2008)
15. Annen, T., Mertens, T., Bekaert, P., Seidel, H.P., Kautz, J.: Convolution shadow maps. In: Rendering Techniques 2007: Eurographics Symposium on Rendering, Grenoble, France, Eurographics, pp. 51–60 (2007)

16. McGuire, M., Enderton, E.: Colored stochastic shadow maps. In: Proceedings of the ACM Symposium on Interactive 3D Graphics and Games (2011)
17. Rezk-Salama, C., Ropinski, T., Hadwiger, M., Ljung, P.: Advanced Illumination Techniques for GPU-Based Raycasting. ACM SIGGRAPH Course Notes (2009)
18. Kniss, J., Premoze, S., Hansen, C., Ebert, D.: Interactive translucent volume rendering and procedural modeling. In: VIS 2002: Proceedings of the Conference on Visualization 2002, pp. 109–116. IEEE Computer Society, Washington (2002)
19. Behrens, U., Ratering, R.: Adding shadows to a texture-based volume renderer. In: VVS 1998: Proceedings of the 1998 IEEE Symposium on Volume Visualization, pp. 39–46. ACM, New York (1998)
20. Ropinski, T., Döring, C., Rezk-Salama, C.: Advanced Volume Illumination with Unconstrained Light Source Positioning. IEEE Comp. Graph. & Applications 30, 29–41 (2010)
21. Sundén, E., Ynnerman, A., Ropinski, T.: Image plane sweep volume illumination. IEEE Transactions on Visualization and Computer Graphics 17, 2125–2134 (2011)
22. Salama, C.R.: Gpu-based monte-carlo volume raycasting. In: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications, pp. 411–414. IEEE Computer Society (2007)
23. Kim, T.Y., Neumann, U.: Opacity shadow maps. In: Proceedings of the 12th Eurographics Workshop on Rendering Techniques, pp. 177–182. Springer (2001)
24. Mertens, T., Kautz, J., Bekaert, P., Van Reeth, F.: A self-shadow algorithm for dynamic hair using density clustering. In: SIGGRAPH 2004: ACM SIGGRAPH 2004 Sketches, p. 44. ACM (2004)
25. Sintorn, E., Assarsson, U.: Real-time approximate sorting for self shadowing and transparency in hair rendering. In: I3D 2008: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games, pp. 157–162. ACM (2008)
26. Yuksel, C., Keyser, J.: Deep opacity maps. In: Computer Graphics Forum (Proceedings of EUROGRAPHICS 2008), vol. 27 (2008)
27. Sintorn, E., Assarsson, U.: Hair self shadowing and transparency depth ordering using occupancy maps. In: I3D 2009: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, pp. 67–74. ACM (2009)
28. Salvi, M., Vidimče, K., Lauritzen, A., Lefohn, A.: Adaptive volumetric shadow maps. In: Eurographics Symposium on Rendering, pp. 1289–1296 (2010)
29. Kainz, B., Grabner, M., Bornik, A., Hauswiesner, S., Muehl, J., Schmalstieg, D.: Efficient ray casting of volumetric datasets with polyhedral boundaries on manycore gpus. In: SIGGRAPH Asia 2009: ACM SIGGRAPH Asia 2009 papers. ACM, New York (2009)