CGI2016 manuscript No.
(will be inserted by the editor)

Intermediate Shadow Maps for Interactive Many-Lights Rendering

Lili Wang · Wenhao Zhang · Nian Li · Boning Zhang · Voicu Popescu

Abstract We present an efficient method for computing shadows for many light sources (e.g. 1,024). Our work is based on the observation that conventional shadow mapping becomes redundant as the number of lights increases. First, we sample the scene with a constant number of depth images (e.g. 10), which we call intermediate shadow maps. Then the shadow map for each light is approximated by rendering triangles reconstructed from the intermediate shadow maps. The cost of rendering these triangles is much smaller than rendering the original geometry of a complex scene. The algorithm supports fully dynamic scenes. Our results show that our method can produce soft shadows comparable to those obtained by conventional shadow mapping for each light source or by ray tracing, but at a higher frame rate.

Keywords Many lights, visibility, shadow mapping

1 Introduction

Rendering complex scenes with complex lighting at interactive rates remains an open research problem. The core challenge is to determine visibility between the scene geometry and the light sources. When the scene consists of millions triangles and when lighting is modeled with hundreds or even thousands of point light sources, determining visibility can be very time-consuming, precluding rendering at interactive rates. The conventional approach for rendering shadows in interactive graphics applications is shadow mapping, which does not scale with scene complexity and with the number of lights as it requires rendering the scene once for each light.

In this paper we present an efficient method for computing shadows for many point light sources (e.g. $1,024$). Our method is based on the observation that conventional shadow mapping becomes redundant as the number of lights increases. Given a shadow map SM_i rendered for a light source L_i, SM_i contains a significant part of the visibility information needed to compute shadows for a different light L_j. Given a set of k intermediate shadow maps, the set contains almost all of the visibility information needed to compute shadows for any number of additional lights. Rendering shadow maps for the additional lights is redundant. Instead, our method approximates the shadow map of a light from the set of intermediate shadow maps.

Our method has two steps. First, we sample the scene with a constant number of depth images (e.g. 10), which we call intermediate shadow maps (INSMs). Then the shadow map for each light L is approximated by rendering the triangles of all the intermediate shadow maps from the viewpoint L. Since the number of intermediate shadow maps is constant, the cost of rendering the triangles reconstructed from them is also constant, and, for complex scenes, this cost is much smaller than rendering the entire scene. We tested our approach on several complex scenes where we obtained high quality shadows and good performance (Fig. 1). We also refer the reader to the accompanying video.

Lili Wang · Wenhao Zhang Nian Li · Boning Zhang
State Key Laboratory of Virtual Reality Technology and Systems, School of Computer Science and Engineering,Beihang University

Voicu Popescu
Purdue Univerisity

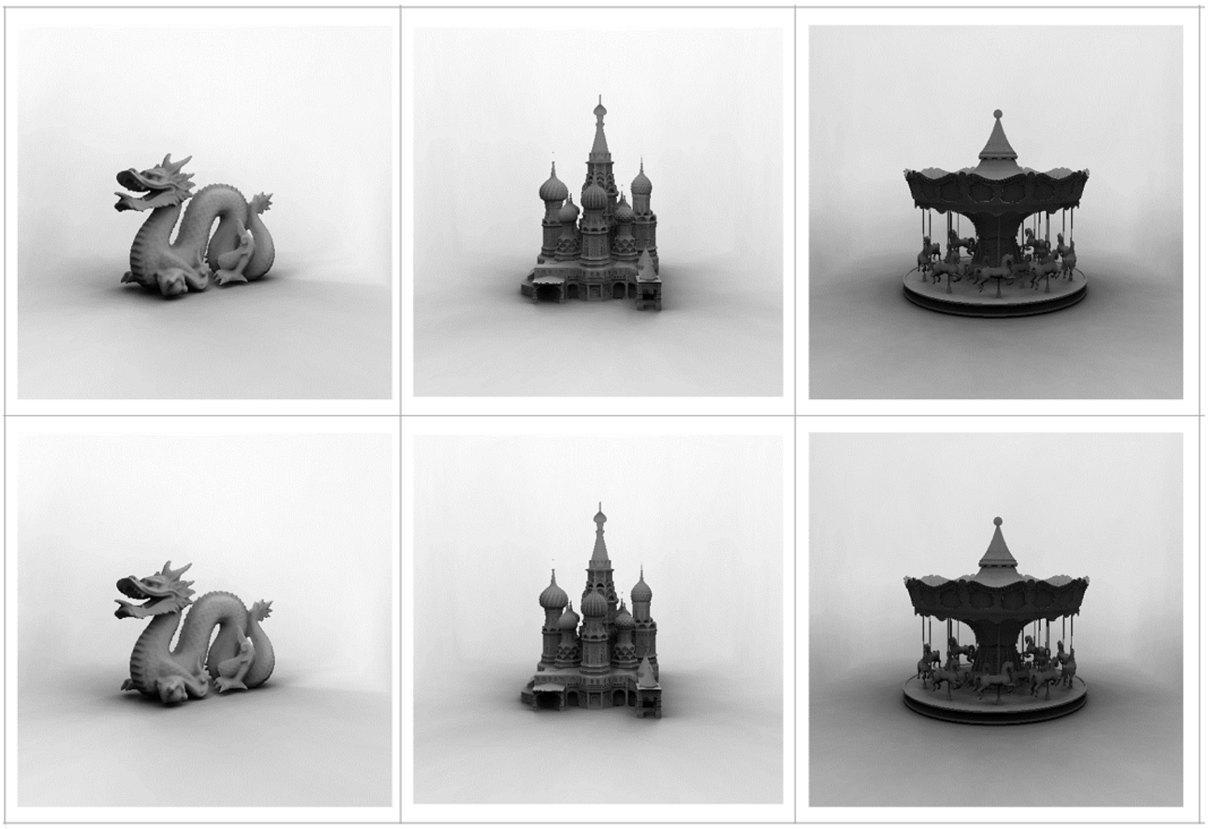

Fig. 1 Scenes with 1,024 lights rendered with our method (row 1), and with ray tracing (row 2). Compared to ray tracing, the average per pixel intensity errors for our method are 0.1%, 0.7% and 1.1%, and the speedups are 12×, 25× and 17×. Compared to conventional shadow mapping, our method brings speedups of 5×, 18× and 10×.

2 Previous Work

The classical methods for shadow computation are shadow mapping and ray tracing. However, these methods are slow for a large number of lights. In order to accelerate visibility computation in the context of shadow rendering, researchers pursued two kinds of approximations: scene geometry approximation, which implies replacing the original scene geometry with a lower cost representation, and lighting approximation, which implies reducing the number of lights by clustering. In addition to our overview of prior work below, we also refer the reader to a recent survey of many-lights techniques [5].

2.1 Ray tracing based methods

Approximating the scene geometry has been pursued in the context of ray tracing. One method is micro-rendering [18], which uses a point tree hierarchy to approximate the geometry of objects in the scenes. The exact visibility of the leaf nodes is determined by ray casting after the cut of tree is computed. Another method is based on incremental voxel-space visibility computations [13], which uses a screen-space voxelization to discretize the scene geometry, and introduces an efficient incremental query to determine the visibility from output samples to light sources.

Several ray tracing based methods focus on simplifying the lights, which brings more time performance advantage for both illumination and visibility, but at the cost of a quality decrease. One method groups lights using an octree [17], and resorts to volumetric visibility approximation. Another method partitions The geometry of the scene into zones and clusters the lights into sets of similar lights per zone, which results in an unstructured light cloud [12]. Lightcut [22] is a popular scalable solution to the many lights problem. A binary light tree is built by clustering the original lights, and cuts through the tree are selected for output samples. The method uses the trivial upper bound of one for the visibility term (i.e. all lights are potentially visible). Many researchers improved visibility computation accuracy based of the original lightcut method (e.g. [23], [3], [4]).

There are some ray tracing based method that approximate both lights and scene geometry. Such as VisibilityClusters in [25]. The method constructs VisibilityClusters with high visibility coherence, and estimate average visibility by exploiting the sparse structure of the matrix and shooting only few shadow rays between clusters.

2.2 Shadow mapping based methods

Compared to ray tracing, shadow mapping based methods are faster, but still not fast enough for interactive performance in the context of a large number of lights and of a complex scene. Shadow mapping acceleration was pursued by scene geometry approximation.

One straight forward method available to practitioners is to simplify the scene geometry with off-the shelf LOD tools such as Simplygon [2] or 3Ds Max [1]. Compared to geometry simplification, our method is robust and it works for any scene, whereas geometry simplification is complex and it requires tuning scene specific parameter values. Our method does not preprocess geometry, so it is suitable to geometry that becomes available in real time, such as geometry acquired with real-time depth cameras. Moreover, simplified geometry will cast acceptable shadows for area light sources that generate soft shadows, but any hard shadow will reveal the coarseness of the underlying geometric model.

If geometry simplification is to take into account the current positions of the lights in order to avoid oversimplifying blockers that cast hard shadows, it can only do so by running for every frame, as the lights are dynamic and the hardness of a shadow cast by one light changes from frame to frame. Furthermore, geometry simplification typically takes into account a single viewpoint, i.e. the eye of the camera that renders the output image. It is difficult to meet the constraints of thousands of viewpoints, each creating a silhouette. For example, the ManyLoDs method [10] uses a bounding volume hierarchy to approximate the scene geometry. For dynamic scenes, the hierarchical structure has to be updated for each frame based on scene graph cuts defined by thousands of lights. In order to complete these steps in real-time, the method has to find some high level cuts, which correspond to a coarse approximation of scene geometry, which might acceptable for faint shadows but not for shadows with higher definition.

Ritschel et al. propose Coherent Shadow Maps (CSM) [20]. The precomputed and compressed depth maps allow visibility tests between moving objects and a high number of lights outside their convex hulls using simple shadow mapping, but the method is unsuitable for virtual point light sources placed on an objects surface, as needed for indirect lighting in global illumination. In order to solve this problem, Coherent Surface Shadow Maps (CSSMs) were proposed, which is a more accurate technique that approximates visibility at scene points using local cube maps [21]. The Virtual Area Lights (VALs) method [7] computes directly the soft shadows cast by the smaller number of VALs using CCSMs with parabolic projection, which has a smaller overall cost than computing hard shadows for all the point light sources. Our method render the intermediate shadow maps in real-time, without preprocessing.

Imperfect Shadow Maps (ISM) [19], is a more general method for many light visibility determination in fully dynamic scenes. A low resolution shadow map is rendered for each light from a coarse point-based approximation of scene geometry by splatting followed by pull-push reconstruction. ISM is a popular method for interactive rendering with many lights, so we compare our method to ISM in detail in the Results Section. The Virtual Shadow Maps technique [16] [15] creates a list of lights influencing each cluster of scene geometry.

In Matrix Row-Column Sampling [9] the columns of a matrix represent all output pixel samples lit by an individual light, and the rows represent an individual sample lit by all lights. A set of representative rows and columns of the matrix are computed first using conventional shadow mapping. A row is computed by rendering the scene from the viewpoint of the sample of the row, and a column is computed by rendering the scene from the light of the column. Then the other matrix elements are approximated by interpolation. The Visibility Clustering method [6] clusters lights, renders a representative shadow map to approximate the visibility in each cluster, and combines the approximate visibility with accurate per light shading. Visibility Clustering requires rendering fewer shadow maps than standard matrix row-column sampling. Matrix row-column sampling was extended to rendering massive scenes with out-of-core geometry and complex lighting [24]. Another extension uses a new matrix sampling-and-recovery scheme to gather illuminations efficiently by only sampling visibility for a small number of representative lights and surface points [11].

Our method falls into the category of shadow mapping based methods. Like in Visibility Clustering our method computes a set of intermediate shadow maps, but then the intermediate shadow maps are reprojected to the viewpoint of each light source, which results in a higher quality approximation of visibility than simply using the representative shadow map for all the lights in the cluster.

3 Intermediate Shadow Maps

Our method avoids the redundancy of rendering hundreds of shadow maps. A small number of intermediate shadow maps are used to approximate the shadow map of each of the many lights. Using the visibility information contained in an intermediate shadow map $INSM_j$ for a light L_i can be done in many ways.

One approach is to leverage epipolar geometry. Given an output image sample S_{uv}, the intersection between the light ray L_i S_{uv} and $INSM_j$ can be computed by projecting L_i S_{uv} onto $INSM_j$ and tracing the projection in search of an intersection. This approach was introduced in inverse 3D image warping [26], and then later used in relief texture mapping [14] and in specular reflection rendering [8]. The advantage is reducing the cost of intersecting a depth image with a ray from $2D$ to $1D$.

However, unlike in the case of inverse 3D image warping, relief texture mapping, and specular reflections where there is a single ray per output image pixel, in our context there are n rays per pixel, where n is the number of lights, which could be in the hundreds or even the thousands. Fortunately, the large set of rays that arises in the context of many-lights rendering is coherent, as the light rays can be grouped in concurrent bundles, with one bundle per light. This enables a second, more efficient approach for using the visibility information of the intermediate shadow map $INSM_j$ for approximating the shadow map of L_i. The second approach, which we adopt, is to transform $INSM_j$ into a triangle mesh and to render the triangle mesh from L_i. This approach leverages the GPU strength of rendering triangles by projection followed by rasterization.

3.1 Algorithm overview

Algorithm 1 outlines the main steps of our approach.

In steps 1-3 the intermediate shadow maps are rendered conventionally from the reference viewpoints V_j that are designed to sample the scene S uniformly and comprehensively. We place the intermediate shadow map reference viewpoints at the midpoints of the eight edges and the centers of the left and right planes in the axis aligned bounding box of S (Fig. 2).

Then each intermediate shadow map $INSM_j$ is converted to a triangle mesh TM_j by defining two triangles for each neighborhood of 2×2 $INSM_j$ samples. No triangles are generated across depth discontinuities. Depth discontinuities are generated by thresholding the second order difference in the depth map. The second order difference is surface orientation independent, *i.e.*

Algorithm 1 Many-Lights Rendering with Intermediate Shadow Maps

Input: scene S with N triangles and n light sources L_i, k reference viewpoints V_j and output view V.
Output: Image I that shows S rendered from V with shadows cast by L_i.

```
1:  for $j = 1$ to $k$ do
2:      Render shadow map $INSM_j$ from ref. viewpoint $V_j$.
3:      Reconstruct triangle mesh $TM_j$ from $INSM_j$
4:  Render $S$ from $V$ without shadows to image $I$
5:  for $i = 1$ to $n$ do
6:      Initialize cube shadow map $SM_i$ to empty
7:      for $j = 1$ to $k$ do
8:          Render $TM_j$ to each face of $SM_i$ from $L_i$
9:      Add to $I$ the shadows from $L_i$ computed using $SM_i$
10: return $I$
```

it is exactly 0 for any plane, no matter its orientation. Step 3 is described in detail in Section 3.2.

Step 4 renders the scene without shadows to image I, which defines the samples for which shadows have to be computed. Steps 5-9 add to I the shadows cast by each light L_i. For each light L_i, an approximate cube shadow map SM_i is constructed first by rendering all intermediate shadow map triangle meshes TM_J from L_i (steps 6-8). Then SM_i is used to compute the shadows from L_i, which are added to I.

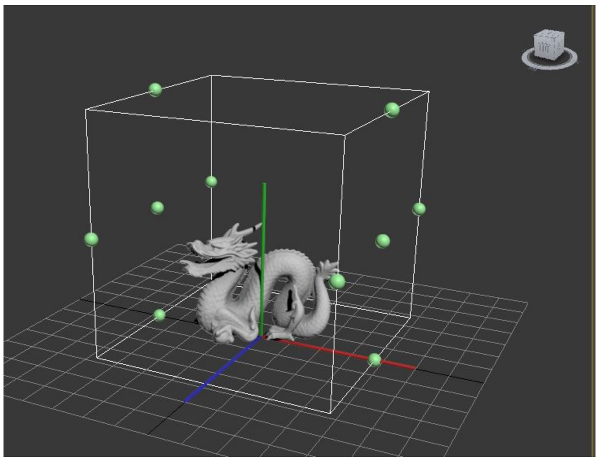

Fig. 2 Reference viewpoint placement for the intermediate shadow maps. The 10 spheres indicate the reference viewpoints with respect to the bounding box of the scene.

3.2 Intermediate Shadow Map Triangulation

We triangulate the intermediate shadow maps on the GPU, by processing neighborhoods of 2×2 intermediate shadow maps samples in parallel, as described in algorithm 2.

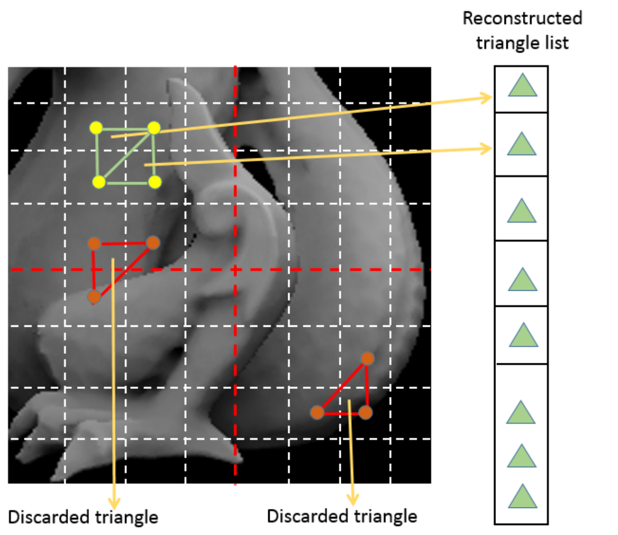

Fig. 3 Triangle mesh reconstruction from intermediate shadow maps. Four neighboring samples in the intermediate map are connected with two triangles. The green triangles are added to the mesh. The red triangles are discarded since they span a depth discontinuity or involve an invalid background sample.

Algorithm 2 Intermediate Shadow Map Triangulation

```
Input: $INSM_j$
Output: triangle mesh $TM_j$ obtained from $INSM_j$
1: for each $INSM_j$ sample $(u, v)$ do
2:     if IsConnected($(u, v), (u, v+1), (u+1, v+1)$) then
3:         $TM_j$ += $[(u, v), (u, v+1), (u+1, v+1)]$
4:     if IsConnected($(u+1, v+1), (u+1, v), (u, v)$) then
5:         $TM_j$ += $[(u+1, v+1), (u+1, v), (u, v)]$
6: return $TM_j$
```

Sample (u, v) is the top left sample of the 2×2 neighborhood. The four samples are connected with two triangles. A triangle is kept if its vertices pass the connectivity test. The triangle connectivity test IsConnected(a, b, c) checks connectivity for each of the three triangle edges (a, b), (b, c), and (c, a). For example, edge $((u, v), (u, v+1))$ passes the connectivity test if both of the two conditions below are met, where $z(u, v)$ is the depth value of sample (u, v), and ε is a threshold that depends on the scene.

$$|z(u, v-1) + z(u, v+1) - 2z(u, v)| < \varepsilon$$

$$|z(u, v) + z(u, v+2) - 2z(u, v+1)| < \varepsilon$$

Fig. 3 shows triangles that are kept and triangles that are discarded, including invalid triangles that involve null (i.e. background) samples.

4 RESULTS AND DISCUSSION

In this section, we discuss the quality of the shadows rendered by our method, we report frame rate measurements, and we discuss limitations.

We tested our algorithm with several scenes: *Dragon*(871ktris), *Church*(1,868ktris), *Carousel*(1,336ktris), *City*(1,117ktris), *Planes*(1,000ktris), *Grass*(1,198ktris), and *Sponza*(1,063ktris). All performance measurements reported in this paper were recorded on a 3.8 GHz Intel(R) Core(TM) i7-2600K CPU PC with 12 GB of RAM and an NVIDIA GeForce GTX 660 graphics card. We use NVIDIA's Optix ray tracer.

4.1 Quality

We compare the shadows rendered with our method to shadows rendered using Imperfect Shadow maps(ISM) and to ground truth shadows rendered with ray tracing. The error metric used is the average shadow intensity error per valid image pixel. Although it is a relaxed error measurement for visibility computation, since positive and negative errors from different lights may cancel out at a given pixel, the average intensity error is directly related to the quality of the output image.

Our method renders high quality shadows, comparable to shadows rendered by ray tracing, for complex scenes with 1,024 lights (see Figs. 1 and 4, as well as the accompanying video). In all our experiments, the default resolution of the intermediate shadow maps is 128×128, the default resolution of the shadow maps computed for each light is 512×512, the default number of intermediate shadow maps is 10, and the default number of light sources is 1,024. Fig. 5 visualizes the error for the six images shown in Figs. 1 and 4.

Our method typically underestimates blockers, by only considering what was captured in the intermediate shadow maps, and by eroding surface edges a half-pixel during reconstruction, which results in light leaking. Consequently, the small approximation errors in our images are typically due to pixels that are brighter than they should be. The occasional "too dark" approximation errors are due to incorrect depth discontinuity detection which generates superfluous blocker surfaces. The *Planes* and *Grass* scenes are the most challenging scenes for our method due to the high depth complexity, which is challenging for the small number of intermediate shadow maps we use, and the minute detail, which is challenging for the reconstruction of the blocker surfaces from the sample base representation brought by the intermediate shadow maps. Even for these challenging scenes, the approximation errors introduced by our method are small.

ISM uses points sampling to approximate the geometry of the scene, which is rendered by splatting and pull-push operations to generate a low resolution shadow map for every light source. In our experiments, we use about 12,000 point samples to render each of the 1,024 128×128 ISMs, which yields a frame rate comparable to that of our method. Table 1 shows that our method has a smaller average shadow intensity error for our scenes.

Table 1 Average pixel shadow value errors for our method and for the prior art Imperfect Shadow Maps method.

Scene	*Dragon*	*Church*	*Carousel*	*City*	*Plane*	*Grass*
Ours	0.1%	0.7%	1.1%	0.8%	3.4%	3.9%
ISM	3.9%	5.8%	4.7%	4.6%	10.0%	9.0%

Tables 2, 3, and 4 show the dependency of the approximation error on the resolution of the intermediate shadow maps, on the resolution of the approximate shadow maps computed for each light, and on the number of intermediate shadow maps, for the *Dragon* scene. As expected, the error decreases as the three parameters increase. For this scene, the values of the three parameters after which returns diminish are 128×128, 512×512, and 10.

Table 2 Approximation error as a function of intermediate shadow map resolution.

Resolution of INSM	64×64	128×128	256×256	512×512
Avg. pixel error	0.56%	0.12%	0.09%	0.08%

Table 3 Approximation error as a function of individual light shadow map resolution.

Resolution of ILSM	256×256	512×512	1024×1024	2048×2048
Avg. pixel error	0.18%	0.12%	0.10%	0.09%

Table 4 Approximation error as a function of the number of intermediate shadow maps.

Number of INSMs	6	8	10	12
Avg. pixel error	0.30%	0.16%	0.12%	0.11%

We have also tested our method with an inside looking out scene *Sponza*. We set 14 reference cameras for this scene. Fig. 6 gives a top view illustration of the reference camera placement. Fig.7 shows our results compared to ground truth, as well as shadow error images. The errors are 5.5% and 4.7%, and our method is 3 times faster than ray tracing.

4.2 Performance

We have compared the performance of our algorithm to that of NVIDIAs Optix ray tracer, and to conventional shadow mapping that renders a shadow map for each light from the original scene geometry. Table 5 provides the frame rendering times for all three methods. Our method is between 11.8× and 24.9× faster than ray tracing, and between 5.0× and 17.5× faster than conventional shadow mapping. The speedup comes from replacing the original scene geometry with the triangle meshes reconstructed from the intermediate shadow maps, when computing the individual light shadow maps. The numbers of triangles in these meshes is 119k, 43k, 135k, 69k, 112k, and 94k for the *Dragon*, *Planes*, *City*, *Grass*, *Carousel* and *Church* scenes, which is considerably less than the number of triangles in the scene models. For a scene with N triangles, for k intermediate shadow maps of resolution $w \times w$, and for n lights, the number of triangles rendered by our method is at most $kN + 2nkw^2$, where we counted 2 reconstructed triangles per intermediate shadow map sample. Conventional shadow mapping renders nN triangles, so our method scales much better with scene geometric and lighting complexity.

In our experiments, we tried to perform an equal quality comparison to ISM by increasing the number of scene point samples used in ISM. No matter how much we increased the number of point samples, ISM quality remained inferior to the quality of our method. Once the number of point samples increases above what can be handled by the GPU in a single pass, the additional rendering pass made performance slower than that of ray tracing.

Table 5 Frame rendering times in milliseconds for our method (INSM), for conventional shadow mapping (SM), and for ray tracing (RT).

Scene	INSM	SM	*SM/ INSM*	RT	*RT/ INSM*
Dragon	844	4,200	5.0×	10,000	11.8×
Church	744	13,000	17.5×	18,500	24.9×
Carousel	835	8300	9.9×	14300	17.1×
City	960	7300	7.6×	12,500	15.2×
Planes	493	4,100	8.3×	10,500	21.3×
Grass	680	9,400	13.8×	16,000	23.5×

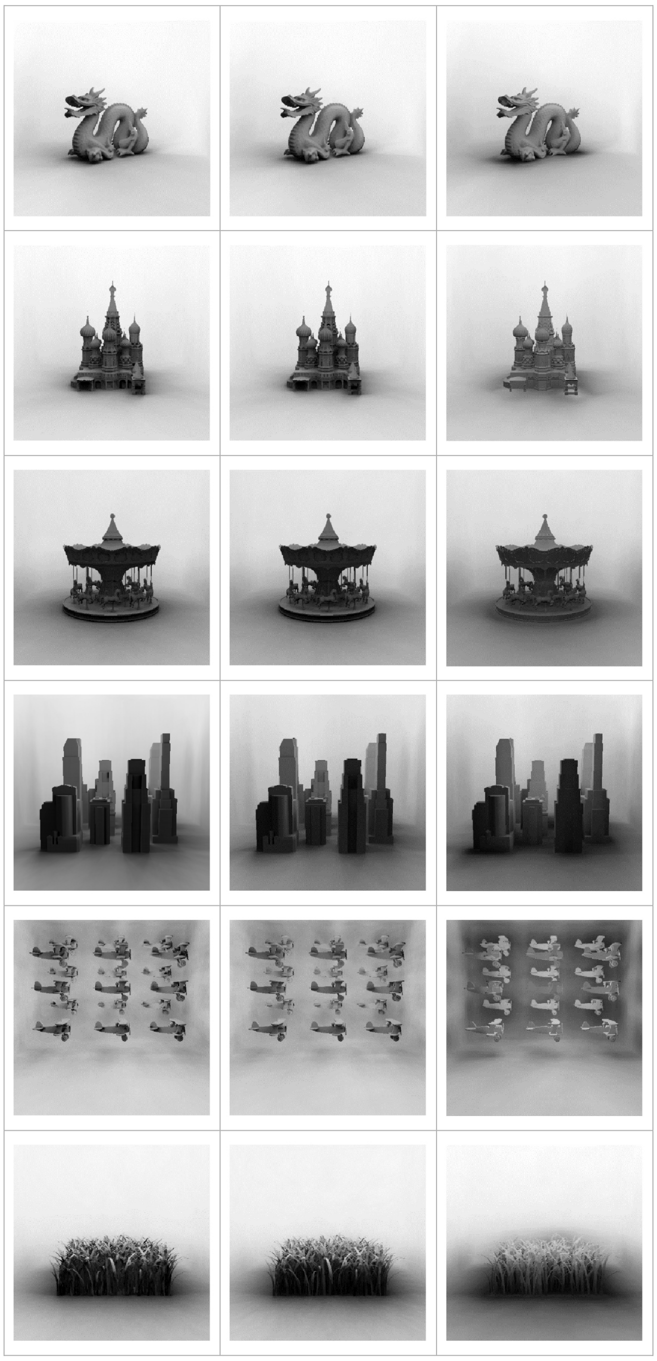

Fig. 4 Comparison between our method (left), ray tracing (middle), and imperfect shadow maps (right). The approximation introduced by ISM translates into noticeable shadow errors.

The graphs in Figs. 8, 9, and 10 confirm the quadratic dependence of performance on the linear resolution w of the intermediate shadow maps, and the linear dependence on the number of intermediate shadow maps and on the number of lights.

The graph in Fig. 11 confirms that the resolution of the shadow maps computed for individual lights does not affect performance much, which indicates that rendering the individual light shadow maps is geometry and not fill-rate bound.

4.3 Limitations

As discussed, our method approximates blocker geometry with intermediate shadow maps, which can result

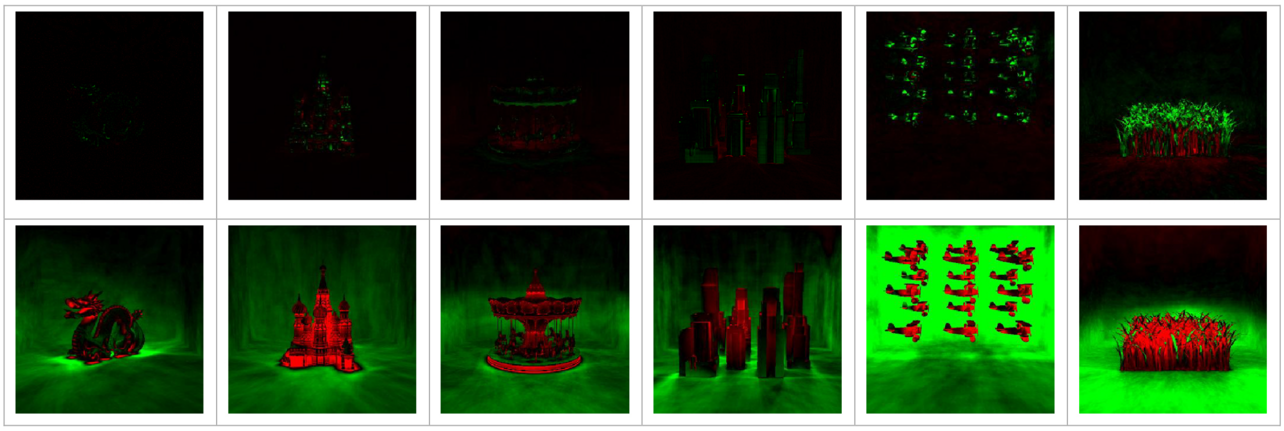

Fig. 5 Visualization of approximation errors of our method (top) and ISM (bottom)for the images shown in Fig. 4. The error is scaled by a factor of 5 for illustration purposes. Red/green highlights pixels that are too bright/dark.

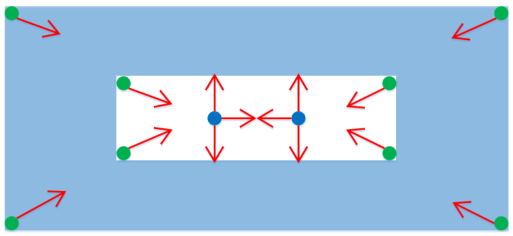

Fig. 6 Reference viewpoint placement for the intermediate shadow maps for the *Sponza* scene. The cameras shown by the blue dots are placed mid-height with a horizontal view direction, and the cameras shown by the green dots are ground-level and look at the center of the scene.

in light leaks when the blocker geometry is not sampled well enough. Insufficient sampling can be caused by high depth complexity, i.e. many occluding layers, or by high surface complexity, i.e. minute details. Our method provides a straightforward approach for mitigating these challenges: increasing the number of intermediate shadow maps, and increasing the resolution of intermediate shadow maps. Adequate values for these two essential parameters should be determined based on the scene and based on the application.

Another limitation of our approach is that, in order to surpass the performance of conventional shadow mapping, the scene has to be sufficiently complex such that the triangle meshes reconstructed from the intermediate shadow maps be less expensive than the original scene model, and the number of lights should be sufficiently large such that these per light gains accumulate to overtake the initial startup cost of rendering and triangulating k intermediate shadow maps.

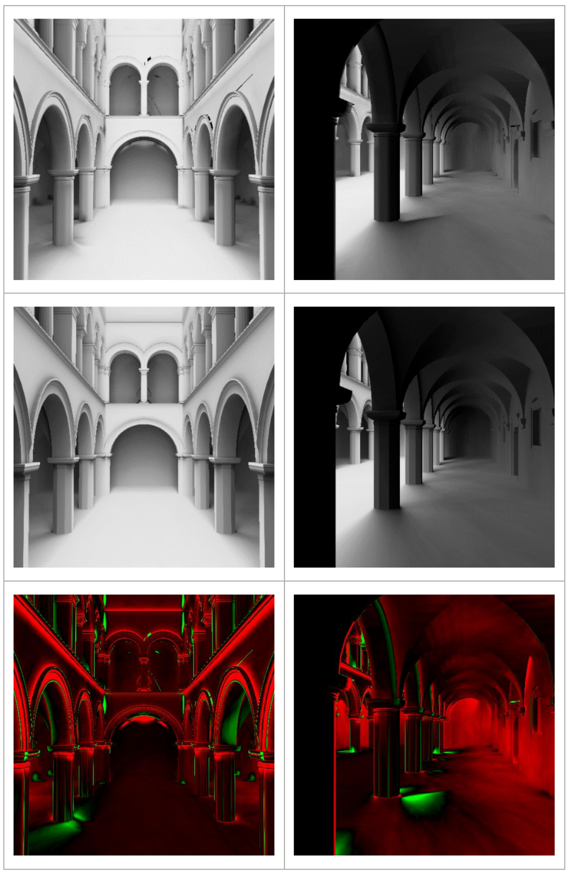

Fig. 7 *Sponza* inside-looking-out scene with 1,024 lights rendered with our method (top), and with ray tracing (middle). The approximation errors of our method are 5.5% (left) and 4.7% (right). We visualize the approximation errors scaled up by a factor of 5, with red/green highlighting pixels where the images are too bright/too dark.

5 Conclusions and Future Work

We have presented a general and efficient method for rendering shadows for many light sources. The method

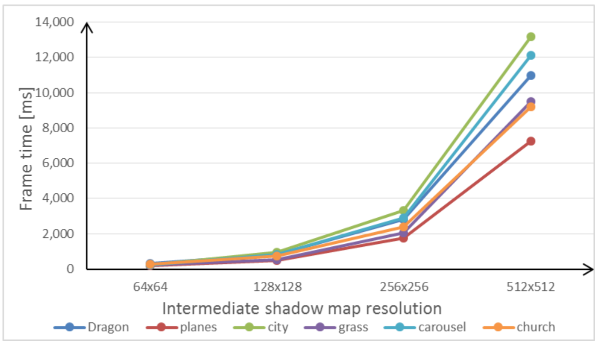

Fig. 8 Frame rendering time as a function of the linear resolution of the intermediate shadow maps.

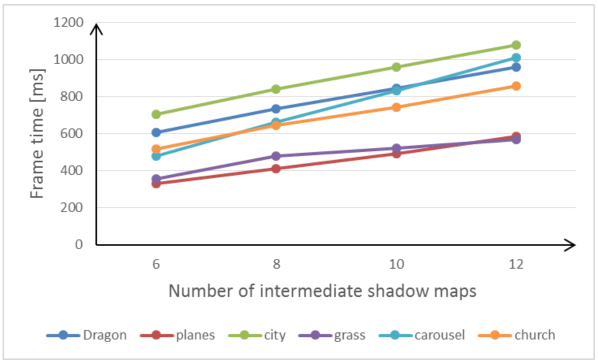

Fig. 9 Frame rendering time as a function of the number of intermediate shadow maps.

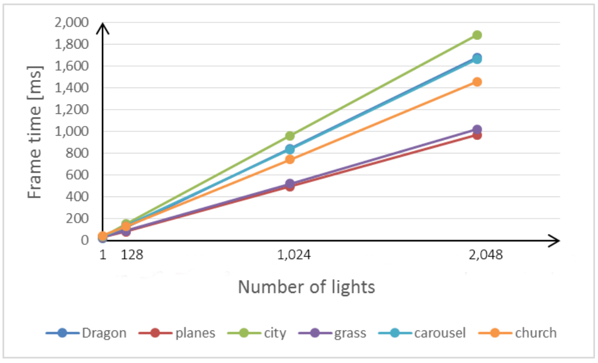

Fig. 10 Frame rendering time as a function of the number of lights in the scene.

handles robustly fully dynamic scenes with millions of triangles and a thousand light sources, and renders high quality soft shadows. Our method decreases the redundancy of conventional shadow mapping a large number of lights by only rendering the scene geometry a small number of times to generate intermediate shadow maps, which are then used to approximate visibility from the individual light sources. The intermediate shadow maps contain much of the visibility information needed for the many light sources. We extract this information carefully by reprojecting the intermediate shadow maps to the viewpoints of the individual lights. We do *not* approximate visibility by interpolation, since visibility is notoriously non-linear. We do *not* cluster lights, and we truly estimate visibility for each one of the many individual lights. Our lights are free to change from a uniform distribution to a clustered distribution or even to converge to a single point, and our method produces quality shadows, gradually changing from soft to harder and then to hard shadows, without temporal artifacts.

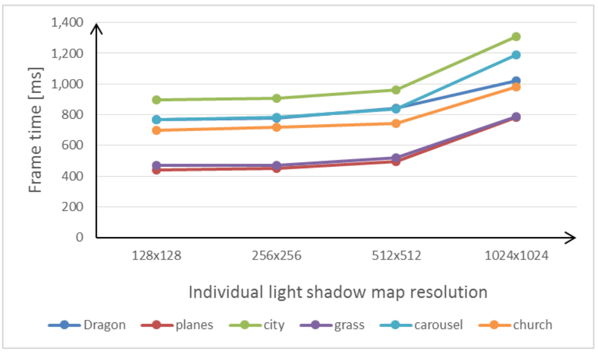

Fig. 11 Frame rendering time as a function of the resolution of the individual light shadow maps.

We compared our results to ground-truth obtained by ray tracing and to conventional shadow mapping over a variety of scenes, and we showed that our method brings a substantial performance gain at the cost of only small approximation errors.

Our method makes progress in the direction of substantially increasing the number of lights that are available to interactive graphics applications. An important direction of future work is to provide algorithmic support for lighting design by automatically placing and calibrating the individual light sources. Another direction of future work is to investigate the extension of our method to global illumination where surface samples become virtual point light sources from where secondary light rays originate.

6 Acknowledgments

References

1. 3ds max. `http://www.autodesk.com/products/3ds-max/overview`, 2016.
2. Simplygon. `https://www.simplygon.com/`, 2016.
3. Oskar Akerlund, Mattias Unger, and Rui Wang. Precomputed visibility cuts for interactive relighting with dynamic brdfs. In *Conference on Computer Graphics & Applications*, pages 161–170, 2007.
4. Ewen Cheslack-Postava, Rui Wang, Oskar Akerlund, and Fabio Pellacini. Fast, realistic lighting and material design using nonlinear cut approximation. *Acm Transactions on Graphics*, 27(5):32–39, 2008.

5. Carsten Dachsbacher, Jaroslav Křivánek, Miloš Hašan, Adam Arbree, Bruce Walter, and Jan Novák. Scalable realistic rendering with many-light methods. In *Computer Graphics Forum*, volume 33, pages 88–104. Wiley Online Library, 2014.
6. Tomáš Davidovič, Jaroslav Křivánek, Miloš Hašan, Philipp Slusallek, and Kavita Bala. Combining global and local virtual lights for detailed glossy illumination. In *ACM Transactions on Graphics (TOG)*, volume 29, page 143. ACM, 2010.
7. Zhao Dong, Thorsten Grosch, Tobias Ritschel, Jan Kautz, and Hans-Peter Seidel. Real-time indirect illumination with clustered visibility. In *VMV*, pages 187–196, 2009.
8. Rogerio Feris, Ramesh Raskar, Kar-Han Tan, and Matthew Turk. Specular reflection reduction with multi-flash imaging. In *Proceedings of the Computer Graphics and Image Processing, XVII Brazilian Symposium*, SIBGRAPI '04, pages 316–321, Washington, DC, USA, 2004. IEEE Computer Society.
9. Miloš Hašan, Fabio Pellacini, and Kavita Bala. Matrix row-column sampling for the many-light problem. In *ACM Transactions on Graphics (TOG)*, volume 26, page 26. ACM, 2007.
10. Matthias Hollander, Tobias Ritschel, Elmar Eisemann, and Tamy Boubekeur. Manylods: Parallel many-view level-of-detail selection for real-time global illumination. In *Computer Graphics Forum*, page 1233C1240, 2011.
11. Yuchi Huo, Rui Wang, Shihao Jin, Xinguo Liu, and Hujun Bao. A matrix sampling-and-recovery approach for many-lights rendering. *ACM Transactions on Graphics (TOG)*, 34(6):210, 2015.
12. Anders Wang Kristensen, Tomas Akenine-M?ller, and Henrik Wann Jensen. Precomputed local radiance transfer for real-time lighting design. *Acm Transactions on Graphics*, 24(3):1208–1215, 2005.
13. Greg Nichols, Rajeev Penmatsa, and Chris Wyman. Interactive, multiresolution image-space rendering for dynamic area lighting. In *Computer Graphics Forum*, volume 29, pages 1279–1288. Wiley Online Library, 2010.
14. Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief texture mapping. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 359–368, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.
15. Ola Olsson, Markus Billeter, Erik Sintorn, Viktor Kampe, and Ulf Assarsson. More efficient virtual shadow maps for many lights. 2015.
16. Ola Olsson, Erik Sintorn, Viktor Kämpe, Markus Billeter, and Ulf Assarsson. Efficient virtual shadow maps for many lights. In *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 87–96. ACM, 2014.
17. Eric Paquette, Pierre Poulin, and George Drettakis. A light hierarchy for fast rendering of scenes with many lights. In *Computer Graphics Forum*, volume 17, pages 63–74. Wiley Online Library, 1998.
18. T. Ritschel, T. Engelhardt, T. Grosch, H. P. Seidel, J. Kautz, and C. Dachsbacher. Micro-rendering for scalable, parallel final gathering. *Acm Transactions on Graphics*, 28(5):89–97, 2009.
19. T. Ritschel, T. Grosch, M. H. Kim, H. P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *Acm Transactions on Graphics*, 27(5):32–39, 2008.
20. Tobias Ritschel, Thorsten Grosch, Jan Kautz, and Stefan Eller. Interactive illumination with coherent shadow maps. *In Proc. EGSR (2007*, pages 61–72, 2007.
21. Tobias Ritschel, Thorsten Grosch, Jan Kautz, and Hans Peter Seidel. Interactive global illumination based on coherent surface shadow maps. In *Proceedings of Graphics Interface 2008*, 2008.
22. Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: A scalable approach to illumination. *Acm Transactions on Graphics*, 24(3):pgs. 1098–1107, 2005.
23. Bruce Walter, Pramook Khungurn, and Kavita Bala. Bidirectional lightcuts. *Acm Transactions on Graphics*, 31(4):13–15, 2012.
24. Rui Wang, Yuchi Huo, Yazhen Yuan, Kun Zhou, Wei Hua, and Hujun Bao. Gpu-based out-of-core many-lights rendering. *ACM Transactions on Graphics (TOG)*, 32(6):210, 2013.
25. Yu-Ting Wu and Yung-Yu Chuang. Visibilitycluster: Average directional visibility for many-light rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 19(9):1566–1578, 2013.
26. Tang Yang, Wu Hui-zhong, Xiao Fu, and Xiao Liang. Inverse image warping without searching. In *International Conference on Control, Automation, Robotics and Vision*, pages 386–390, 2004.