Self-Shadowing of Cynamic Rcenes with Dnvironment L aps using the GPU

Martin Knuth
Fraunhofer IGD
Fraunhoferstr. 5
64283 Darmstadt, Germany
mknuth@igd.fhg.de

Arnulph Fuhrmann
Fraunhofer IGD
Fraunhoferstr. 5
64283 Darmstadt, Germany
afuhr@igd.fhg.de

ABSTRACT

In this paper we present a method for illuminating a dynamic scene with a high dynamic range environment map with real-time or interactive frame rates, taking into account self shadowing. Current techniques require static geometry (pre-computed light transport), are limited to few and small area lights or are limited in the frequency of the shadows. We facilitate importance sampling of the environment map and GPU based shadow calculation in an efficient way. The shadows are calculated per pixel, so no highly tessellated models are necessary in opposition to other techniques. Our method provides a novel and highly efficient way for using shadow maps as data structure for visibility computations done entirely on the GPU. We achieve real-time frame rates for moderate sized models on current graphics hardware. Since we evaluate the light transport of the scene per frame, complex dynamically animated models can be rendered efficiently.

Keywords Shadow Algorithms, Environment Mapping, GPU Programming

1. INTRODUCTION

Shadows reveal information about spatial object relation within a scene. Hence, using shadows in computer graphics allows a better immersion into a scene. Enhancing the quality and dynamics of the shadows will result in a more efficient comprehension of the image. For that task we present a system for rendering shadows caused by an environment map. The system evaluates the self shadowing of the scene at interactive or real time frame rates on current GPUs. The shadows are evaluated per pixel and are not limited to low frequencies. Furthermore, objects are allowed to be non-manifold. All this is done for fully dynamic scenes without prior knowledge of the animation.

WSCG 2005 conference proceedings,
ISBN 80-903100-7-9
WSCG'2005, January 31-February 4, 2005, Plzen, Czech Republic. Copyright UNION Agency - Science Press

Existing systems used for rendering such scenes lit by an environment map are limited to vertex lighting, do not allow shadows or are not interactive.

The presented method makes use of importance sampling to create a light setup, which approximates the environment map. The light visibility is determined with shadow buffers. Since the rendering is entirely done on the GPU, no time expensive read backs of data towards the CPU is needed.

In section 2 we refer to existing work related to our approach. Then, we explain our algorithm and the ideas behind it. Additionally some issues are presented which have to be considered when implementing the algorithm on a GPU. In section 4 an implementation of the system is shown and discussed, followed by a summary and a look into the future.

2. RELATED WORK

There exists a lot of literature addressing the problems of shadowing 3D scenes in real time. In [Has03] several methods for generating soft shadows are compared. Unfortunately, they are either limited to small area sources or too slow to be useful for our approach. Rendering real time shadows of dynamic geometry on today's graphics hardware is mainly done by only two approaches: shadow volumes [Cro77] and shadow

maps [Wil78]. Both methods are capable of rendering shadows of directional- and point lights, creating hard shadow boundaries. Shadow maps are fast to compute and need less fill rate than shadow volumes. On the other hand, care has to be taken of sampling artefacts. An approach for minimizing shadow buffer artifacts can be found in [Ree87].

Direct illumination of a scene with an environment map is given by a group of environment mapping methods (like [Gre86, Ram01, Hei99]). They do not support shadows, which is a big disadvantage.

In [Deb98] Paul Debevec presents image based lighting done with high dynamic range (HDR) environment maps, to bring together synthetic and real scenes under natural lighting conditions.

Ray tracing [Wal03b, Wal03a, Pur02] is another approach for visibility determination. It has reached interactive and real time frame rates by using a PC-cluster, the GPU or dedicated ray tracing hardware. Unfortunately, there is still development needed until this hardware is out of prototype state. The software solution needs the power of an efficient cluster system. Ray tracing on the GPU at last needs fast read back towards the CPU, which is a bottleneck.

Global illumination at interactive frame rates can be done, utilising a real time raytracer. A direct approach towards global illumination on the GPU is presented in [Coo04], achieving interactive frame rates for small scenes. Another method to achieve fast global illumination computations was presented by Keller et. al in [Kel97]. The method accumulates images of the scene illuminated by single shadowcasting lights.

Other methods make use of occlusion or precomputed lighting information to allow the use of an environment map as light source. The information is mostly stored per vertex. Nevertheless a lot of additional information has to be stored. Since directions have to be mapped to data, structures like spherical harmonics are used. Since these methods are based on pre computations they normally can not be used for animated geometry. This flaw is faced by Kautz et al. in [Kau04]. They presented a method for speeding up the pre-calculation to interactive frame rates on small models. In opposition to our method they need a model hierarchy which can have high preprocessing cost - if animated. Due to its Spherical Harmonics/vertex based character, the method processes only low frequency shadows.

Many methods use directional- or point-lights as an approximation of the environment map. Our algorithm belongs to this kind of methods. For the pre-calculation of ambient occlusion NVIDIA [Pha04]presented an algorithm, which uses accumulation of shadow maps in a preprocessing step to light the scene. As a disadvantage their method needs several seconds of preprocessing time for calculating the occlusion, with a reasonable visual quality.

Another approach for calculating ambient occlusion is presented in [Sat04]. It takes into account the colour of the environment map. Their method is mainly different in three points to our approach: Occlusion is evaluated per vertex, lights are generated with spherical distribution and at last occlusion data is read back from GPU.

Since the approximation of an environment map with directional lights is a difficult task, several methods addressing this problem exist. The easiest way is to sample the environment map homogeneously. But since most environment maps have varying areas of interest, it is advantageous to take the importance of these areas into account [Aga03, Kol03, Ost04, Sze04]. This is done by analysing the image to figure out more important areas of the image to place lights in.

3. OUR METHOD

Importance sampling

If we directly used the environment map for lighting a surface point, we would have to solve the problem of integrating over a hemisphere to get the amount of incoming radiance. Since the used environment maps are discrete, we could use a sum over n texels of the environment map for that task, but this would be still too much work to do.

Hence, we reduce the visibility problem to k directional lights, which are computed from the environment map. This is done by using structured importance sampling [Aga03]. This algorithm creates a distribution of the lights on the environment map according to the importance of the respective region. The importance of a region is determined by its extend and its light intensity. Roughly speaking, the method creates many lights in bright areas and only a few in darker ones, as can be seen in Figure 1. For more details on the importance metric and a derivation of it we refer the interested reader to the original paper [Aga03].

The computed point lights accumulate the radiance of their surrounding region. This method works considerably well, so a teapot scene inside Galileo's Tomb using only 300 lights rendered by Agarwal et al. shows no significant difference to a reference image computed with 100,000 samples using standard Monte Carlo sampling.

In difference to [Sat04] we use importance sampling in our system, since the rotation of lights is decou-

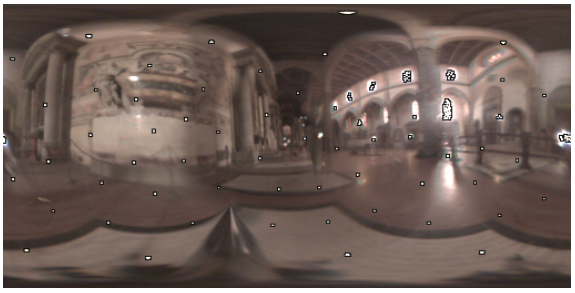

Figure 1: Environment map with 128 importance sampled lights shown as white dots.

pled from the rotation of the geometry. Rotation of the environment is followed by equal rotation of the lights. Although importance sampling of an environment map as described in [Aga03] takes some time, it either can be pre-computed and stored or done once at start up.

Scene Rendering

Conventional real-time algorithms render a shadowed scene light after light in several passes. So, a shadow map is calculated for each light and used directly. Since all triangles have to be rasterized in each pass, this approach generates a lot of redundant calculations, which slows down the rendering. Our idea is to get rid of most of these redundant calculations, by doing something similar to ray tracing: Take a pixel and calculate all lighting for it, then take the next one. Calculations for a given pixel which are independent of the light position need to be done only once. The algorithm looks like this:

Calculate k lights from environment map.
for all Frames **do**
  Calculate k shadow maps.
  for all pixel **do**
    Compute visibility of k lights using the shadow maps.
  end for
end for

Algorithm 1: The shadowing algorithm.

Since rendering the scene taking all lights into account at once, a lot of redundant calculations are prevented. The rendering of the scene into the frame buffer becomes a single pass operation. But the storing and handling of the k shadow maps raises problems addressed in the next section.

Shadow Map Management

Normally, a shadow map is used for one light at a time only. Since shadow maps are usually represented by textures, this causes a lot of state changes. To minimize these state changes we render several shadow maps into one texture to fulfill the requirements set by our algorithm (See Figure 2).

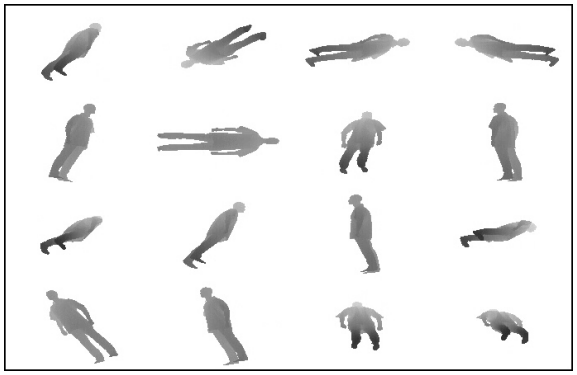

Figure 2: Texture containing several shadow maps, packed side by side.

This approach concentrates first completely on shadow map generation for all light sources and then on rendering the actual scene. Shadow map generation and scene rendering are completely separated.

Further Reduction of the Number of Lights

The structured importance sampling reduces the number of lights necessary to approximate the lighting of an environment map. But, there are still too many lights needed to emulate soft shadows caused by lights on the environment map. Reducing the number of lights further will result in clearly distinguishable shadows with hard boundaries, due to under sampling (See Figure 3).

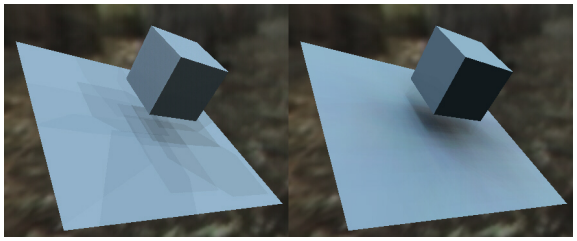

Figure 3: An example for under sampling the environment map: On the left 32 lights are used and the single shadows are clearly distinguishable. On the right figure 2048 light sources have been used.

These hard boundaries arise from sharp shadow edges of the individual light sources. More reduction needs a method to avoid these artefacts. The shadow maps are stored as plain depth information within a texture. So we can use simple 2D image manipulation functions to solve the problem by using a softening filter function.

This allows a blending of shadow boundaries. By this, a quality similar to images rendered with much more lights is achieved (See Figure 4).

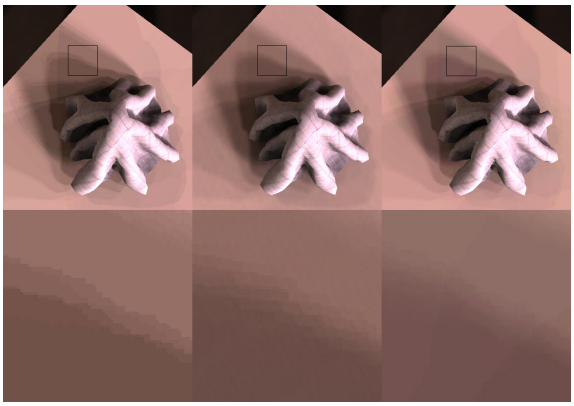

Figure 4: Using smoothing to avoid shadow artifacts: The left image is illuminated by 64 lights. Single shadows are well visible. The center image is illuminated by the same number of light sources, but with smoothed shadows. As reference, the right image is illuminated with 192 lights.

In [Aga03] jittering is used to reduce these artefacts. This is done by randomly choosing a light direction pointing inside the stratum of the light. This takes into account the distance of the occluder to the surface point: With increasing distance of the occluder the shadow gets more and more blurry.

In our approach a shadow map contains the visibility for a constant light direction. As a consequence we can only jitter the position within the shadow map. A shadow boundary will be equally thick regardless of the distance to the occluder. So our method cannot be interpreted as a quick alternative to soft shadow algorithms. Taken alone it just blurs a shadow boundary. The soft shadow effect is caused mainly by the large number of lights used.

GPU-based visibility

Current GPUs behave like a dataflow machine. This has severe consequence when designing algorithms for GPUs. Changing the GPU state for example will stall the pipeline and if this happens often the overall performance decreases considerably. In our system we take care of this by clearly dividing shadow map calculation and scene rendering.

Unfortunately, it is necessary to split Algorithm 1 into several parts, since the GPU has a limited program length and not all lights can be computed in one pass. In order to be able to map our shadowing algorithm onto a programmable GPU, we modified Algorithm 1 into a multi-pass algorithm. The lights are packed into clusters of size c. The size depends on the maximum number of lights supported by the fragment program of the GPU. The modified algorithm is shown in Algorithm 2 and can be implemented on current GPUs.

```
Calculate k lights from environment map.
for all Frames do
  for all Light clusters do
    Calculate c shadow maps.
    for all pixel do
      Compute visibility of c lights using the
      shadow maps.
    end for
  end for
end for
```

Algorithm 2: The modified shadowing algorithm.

Also, all data and intermediate data used should be stored within the GPU memory to prevent wait states. So, intermediate data should simply reside on the GPU. By using a GPU which is able to render into a texture the shadow maps fulfill this criteria. The geometry data can be stored inside the GPU memory for one frame of animation, since we are using a multipass operation this speeds up the algorithm.

4. IMPLEMENTATION AND DISCUSSION

We implemented our algorithm on a Radeon 9700 using OpenGL. The workload was divided between CPU and GPU. The CPU handles constants and the control flow of the algorithm. The vertex processor computes parameters which then can be interpolated over a triangle. The fragment shader does the per pixel work.

In order to map the algorithm efficiently to graphics hardware we used several extensions:

- GL_ARB_vertex_program for vertex program support.
- GL_ARB_fragment_program for fragment program support.
- GL_ARB_vertex_buffer_object for geometry storage inside GPU memory.
- WGL_ARB_pbuffer to be able to render to texture.

We have implemented shadow maps via the PBuffer extension of OpenGL. So using shadow map information is simply a texture lookup. As described above, we are trying to put as many shadow maps in the PBuffer as possible. Unfortunately, the PBuffer has a

maximum resolution which limits the number of containable shadow maps. But since textures are usually coloured there is another way to put more shadow maps inside one PBuffer. Every colour channel is used separately. This multiplies the capacity by four without decreasing the shadow map resolution (See figure 5). The shadow buffer information is interpreted in-

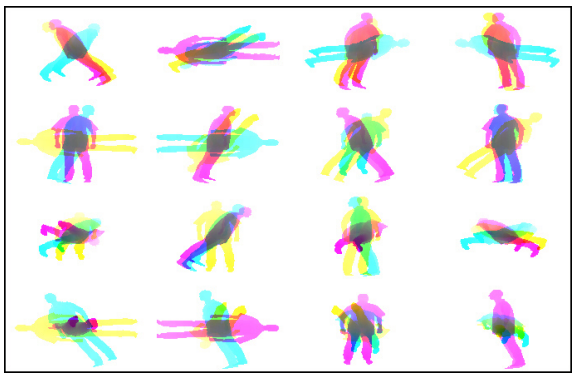

Figure 5: RGBA-texture as shadow buffer (alpha channel not shown): Additionally several buffers were packed side by side.

side the fragment program. Additional filtering (simple smoothing or percentage closer filtering ([Ree87])) is done here, too.

Storing the geometry data inside the GPU is mandatory, since we have a multi-pass algorithm. So the bus between CPU and GPU is free for control operations and is not a bottleneck.

Vertex/Fragment Load Balancing

Load balancing is done by changing the number of lights calculated simultaneously. By calculating one light per pass, most work of the vertex program is done by transforming vertex coordinates. The fragment program has less work to do by handling one light. Nevertheless, it is more often called due to more passes are needed. Calculating several lights per pass increases the load inside the fragment program. Since fewer passes are needed the vertex program has to do less work.

Results

In order to analyse the behaviour of the load balancing we implemented shaders for calculating shadows of one, four and eight lights simultaneously inside the fragment program. The shaders have shown a boost of frame rates as more lights were rendered per frame, since the number of passes decreases. All methods for reducing fill rate and vertex count have shown direct consequences towards higher frame rates. The implementation also has shown that careful detection of bottlenecks and several exploitations of redundancies created a system, able to reach real time frame rates for moderate polygon count models. In practical use, bottlenecks tend to wander between fragment program and vertex program, dependent on the amount of objects covering the screen.

The evaluation of the shadows within the fragment program allows user defined filtering functions. Observing the visual results of our system, shadow smoothing is not always necessary to be convincing. It depends on the environment map and the number of lights used.

5. CONCLUSIONS

We presented a method for self-shadowing of dynamic scenes with environment maps using the GPU. Our algorithm allows the creation of interactive systems, which are capable of rendering scenes taking into account self shadowing caused by an environment map. We evaluate the lighting condition of the geometry on the fly by using current graphics hardware and their shadow mapping features. Our algorithm achieves interactive frame rates for large dynamic models, without prior knowledge of the animation. The implementation is flexible enough to allow an easy load balancing between vertex and fragment program, by controlling the number of lights rendered per pass. In order to raise the visual quality of the shadows we use smoothing and percentage closer filtering.

The implementation of the algorithm has shown it's ability to achieve real-time frame rates for models with moderate polygon count with plausible looking self shadowing of the scene, realistically illuminated by the HDR environment map.

6. FUTURE WORK

One direction for future research would be to consider more information about the light source. The directional lights created by importance sampling describe actually areas of the environment map and not just a singular point. If we took the shape and size of the light source into account during the visibility computations, it would be possible to reduce the number of light sources needed for realistic images even further.

The rendered images would reach a next grade towards photo realism if inter-reflections are taken into account. Since this requires visibility calculation between faces of the geometry, inter-reflections are not handled by our scheme yet. It will take several generations of GPUs and further algorithmic improvements until this vision will be reality.

7. ACKNOWLEDGEMENTS

The high dynamic range environment maps used in this paper were made by Paul Debevec [Deb98]. Textiles shown were created with the prepositioning and cloth simulation methods described in [Fuh03b, Fuh03a, Gro03].

8. REFERENCES

[Aga03] Agarwal, S., Ramamoorthi, R., Belongie, S., and Jensen, H. W. (2003). Structured importance sampling of environment maps. *ACM Trans. Graph.*, 22(3):605–612.

[Coo04] Coombe, G., Harris, M. J., and Lastra, A. (2004). Radiosity on graphics hardware. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 161–168. Canadian Human-Computer Communications Society.

[Cro77] Crow, F. (1977). Shadow algorithms for computer graphics. *j-COMPGRAPHICS*, 11(2):242–248.

[Deb98] Debevec, P. (1998). Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 189–198. ACM Press.

[Fuh03a] Fuhrmann, A., Gross, C., and Luckas, V. (2003a). Interactive animation of cloth including self collision detection. *Journal of WSCG*, 11(1):141–148.

[Fuh03b] Fuhrmann, A., Gross, C., Luckas, V., and Weber, A. (2003b). Interaction-free dressing of virtual humans. *Computers & Graphics*, 27(1):71–82.

[Gre86] Greene, N. (1986). Environment mapping and other applications of world projections. *IEEE Comput. Graph. Appl.*, 6(11):21–29.

[Gro03] Gross, C., Fuhrmann, A., and Luckas, V. (2003). Automatic pre-positioning of virtual clothing. In *Proceedings of the Spring Conference on Computer Graphics*, pages 113–122.

[Has03] Hasenfratz, J.-M., Lapierre, M., Holzschuch, N., and Sillion, F. (2003). A survey of real-time soft shadows algorithms. In *Eurographics*. Eurographics, Eurographics. State-of-the-Art Report.

[Hei99] Heidrich, W. and Seidel, H.-P. (1999). Realistic, hardware-accelerated shading and lighting. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 171–178. ACM Press/Addison-Wesley Publishing Co.

[Kau04] Kautz, J., Lehtinen, J., and Aila, T. (2004). Hemispherical rasterization for self-shadowing of dynamic objects. In *Proceedings Eurographics Symposium on Rendering 2004*.

[Kel97] Keller, A. (1997). Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co.

[Kol03] Kollig, T. and Keller, A. (2003). Efficient illumination by high dynamic range images. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 45–50. Eurographics Association.

[Ost04] Ostromoukhov, V., Donohue, C., and Jodoin, P.-M. (2004). Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics*, 23(3):488–495. Proc. SIGGRAPH 2004.

[Pha04] Pharr, M. (2004). Ambient occlusion. *Game Developers Conference (GDC) 2004*.

[Pur02] Purcell, T. J., Buck, I., Mark, W. R., and Hanrahan, P. (2002). Ray tracing on programmable graphics hardware. *ACM Trans. Graph.*, 21(3):703–712.

[Ram01] Ramamoorthi, R. and Hanrahan, P. (2001). An efficient representation for irradiance environment maps. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 497–500. ACM Press.

[Ree87] Reeves, W. T., Salesin, D. H., and Cook, R. L. (1987). Rendering antialiased shadows with depth maps. *SIGGRAPH Comput. Graph.*, 21(4):283–291.

[Sat04] Sattler, M., Sarlette, R., Zachmann, G., and Klein, R. (2004). Hardware-accelerated ambient occlusion computation. In Girod, B., Magnor, M., and Seidel, H.-P., editors, *Vision, Modeling, and Visualization 2004*, pages 331–338. Akademische Verlagsgesellschaft Aka GmbH, Berlin.

[Sze04] Szecsi, L., Sbert, M., and Szirmay-Kalos, L. (2004). Combined correlated and importance sampling in direct light source computation and environment mapping. *Computer Graphics Forum (Eurographics 04)*, 23(3).

[Wal03a] Wald, I., Benthin, C., and Slusallek, P. (2003a). Interactive global illumination in complex and highly occluded environments. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 74–81. Eurographics Association.

[Wal03b] Wald, I., Purcell, T. J., Schmittler, J., Benthin, C., and Slusallek, P. (2003b). Realtime ray tracing and its use for interactive global illumination. In *Eurographics State of the Art Reports*.

[Wil78] Williams, L. (1978). 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.

Figure 6: The left image was rendered with standard OpenGL. The center image was illuminated by an irradiance map. The right image was rendered with our algorithm taking self-shadowing into account. The resolution was 421x711 pixel and 128 smoothed shadows were used. We achieved four frames per second. The model consists of 107K triangles.

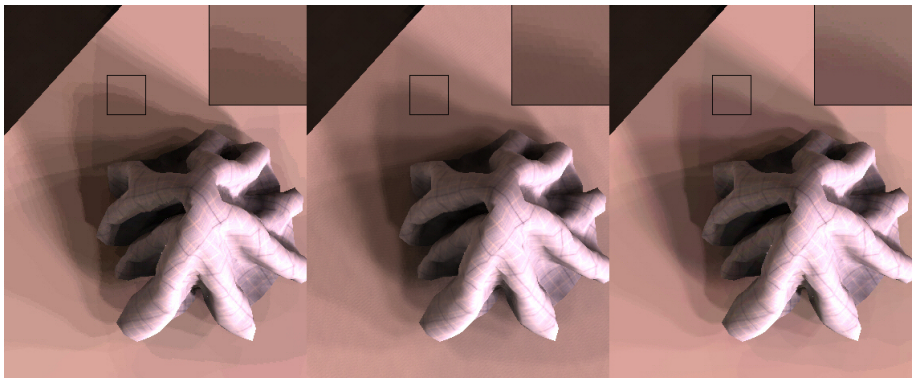

Figure 7: The left image was rendered with 64 light sources and 8 lights per pass at 21 FPS. The middle image was rendered with 64 light sources, shadow smoothing and 4 lights per pass at 8 FPS. The right image was rendered with 192 light sources and 8 lights per pass at 6 FPS.

Figure 8: Some sample frames taken from a real-time animation and rendering of cloth.