Single Sample Soft Shadows using Depth Maps

Stefan Brabec Hans-Peter Seidel

Max-Planck-Institut für Informatik

Abstract

In this paper we propose a new method for rendering soft shadows at interactive frame rates. Although the algorithm only uses information obtained from a single light source sample, it is capable of producing subjectively realistic penumbra regions. We do not claim that the proposed method is physically correct but rather that it is aesthetically correct. Since the algorithm operates on sampled representations of the scene, the shadow computation does not directly depend on the scene complexity. Having only a single depth and object ID map representing the pixels seen by the light source, we can approximate penumbrae by searching the neighborhood of pixels warped from the camera view for relevant blocker information.

We explain the basic technique in detail, showing how simple observations can yield satisfying results. We also address sampling issues relevant to the quality of the computed shadows, as well as speed-up techniques that are able to bring the performance up to interactive frame rates.

1 Introduction

One of the most problematic tasks in computer graphics is the accurate and efficient computation of soft shadows caused by extended light sources. Although there have been enormous efforts in this specific area, only a small subset of algorithms are really appropriate for interactive rendering applications.

In this paper we will present a way of computing soft shadows using only sampled images taken from the view of a point light source. This soft shadow algorithm can be seen as an extension of the classical shadow map algorithm for calculating hard shadows. Instead of computing only a binary value (shadowed or lit) for each pixel seen by the camera, our algorithm processes the neighborhood of the corresponding depth map entry to gather information about what the shadow might look like in the case of an area light source.

Even though the input data contains no information about the characteristics of an area light, the resulting shadows are yet of very good quality and give the impression of a physically plausible computation. Using only a minimal amount of input data and a very compact algorithm, we can achieve extremely high computation speed. This way we can also utilize graphics hardware and specialized processor instruction sets.

2 Previous Work

Since a vast number of hard and soft shadow methods exist for general and very specific situations, we will only briefly discuss some methods here, focusing on those suitable for interactive and real-time applications, as well as on algorithms which are related to our method. As a good starting point we recommend Woo's survey on shadow algorithms [21].

In the field of hardware accelerated, interactive rendering, shadow algorithms are mainly categorized by the space in which the calculation takes place. One of the fundamental shadow algorithms, Crow's shadow volumes [5], processes the geometry of the scene. By extending occluder polygons to form semi-infinite volumes, so called shadow volumes, shadowed pixels can be determined by simply testing if the pixel lies in at least one shadow volume. A hardware-accelerated implementation of Crow's shadow algorithm was later proposed by Heidmann [10]. McCool [15] presented an algorithm that reduces the often problematic geometry complexity of Crow's method by reconstructing shadow volumes from a sampled depth map. Complexity issues were also addressed by Chrysanthou and Slater [4]. They propose the use of BSP trees for efficient shadow volume calculations in dynamic scenes. Brotman and Badler [3] came up with a soft shadow version of Crow's algorithm where they generated shadow volumes for a number of light source samples and computed the overlap using a depth buffer algorithm. Discontinuity Meshing, e.g. [14], is another exact way for computing soft shadows in object-space. Here surfaces are subdivided in order to determine areas where the visible part of the area light source is constant.

William's shadow map algorithm [20] is the fundamental idea of most methods working on sampled representations of the scene. The depths of visible pixels are computed for the view of the light source and stored away in a so called depth or shadow map. In the final rendering pass, pixels seen by the camera are transformed to the light source coordinate system and tested against the precomputed depth values. A hardware-based shadow map

technique was presented by Segal et al. [18].

William's original work suffered from sampling artifacts during the generation of the shadow map as well as when performing the shadow test. Reeves et al. [17] proposed a filtering method called percentage closer filtering which solved these problems and generates smooth, antialiased shadow edges. Reeves' approach is also often used to approximate penumbra regions by varying the filter kernel with respect to the projected footprint. This is somewhat similar to our approach but in general requires a very high resolution depth map in order to obtain soft shadows with reasonable quality.

Brabec et al. [2] showed how Reeves' filtering scheme can be efficiently mapped to hardware. Hourcade and Nicolas [12] also addressed the shadow map sampling problems and came up with a method using object identifiers (priority information) and prefiltering.

To compute soft shadow textures for receiver polygons, Herf and Heckbert [9] combined a number of hard shadow images using an accumulation buffer [7]. Although this method uses graphics hardware, it still requires a large number of light source samples to achieve smooth penumbra regions.

An approximative approach to soft shadowing was presented by Soler and Sillion [19] using convolution of blocker images. On modern hardware this method can utilize specialized DSP features to convolve images, leading to interactive rendering times. The main drawback of the method is the clustering of geometry, as the number of clusters is directly related to the amount of texture memory and convolution operations.

Heidrich et al. [11] showed that soft shadows for linear light sources can be computed using only a minimal number of light source samples. Depth maps are generated for each sample point and processed using an edge detection step. The resulting discontinuities are then triangulated and warped to a so called visibility map, in which a percentage visibility value is stored. Although the method works very well for linear lights, it can not directly be applied to the case of area light sources.

Keating and Max [13] used multi-layered depth images (MDIs) to approximate penumbra regions. This method is related to our algorithm because MDIs are obtained from only a single light source sample. However, in contrast to this multi-layer approach, our algorithm operates just on a single depth map taken from the view of the light source sample.

Agrawala et al. [1] efficiently adopted image-based methods to compute soft shadows. Although their coherence-based ray tracing method does not perform at interactive rates, they also presented an approach using layered attenuation maps, which can be used in interactive applications.

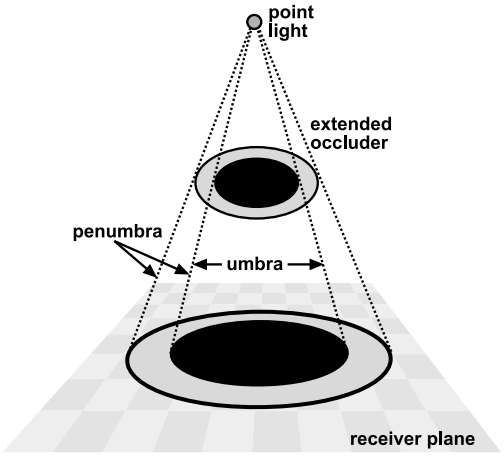

Figure 1: Computing penumbrae for a point light source.

A fast soft shadow method, especially suited for technical illustrations, was proposed by Gooch et al. [6]. Here the authors project the same shadow mask multiple times onto a series of stacked planes and translate and accumulate the results onto the receiver plane.

Haines [8] proposed a method for approximating soft shadows by first generating a hard shadow image on the receiver plane and then compute penumbra regions using distance information obtained from the occluder's silhouette edges. This paper is related to our work since it is also based on the work of Parker et al. [16], which will be explained in detail in Section 3.1. Drawbacks of Haines' method are that receivers need to be planar and that penumbra regions are only generated for regions outside the initial hard shadow.

3 Soft Shadow Generation using Depth Maps

3.1 Single Sample Soft Shadows

Parker et al. [16] showed how soft penumbra regions can be generated by defining an extended hull for each possible occluder object. By treating the *inner* object as opaque and having the opacity of the *outer* object fall off towards the outer boundary one can dim the contribution of a point light source according to the relative distances of light, occluder and receiver. Figure 1 illustrates this scheme.

In order to avoid light leaks occurring for adjacent objects the size of the inner object needs to be at least as large as the original occluder geometry. Since this causes relatively large umbra regions, which would not occur in a physically correct shadow computation, the approximation still produces reasonably looking shadows as long as the occluder objects aren't too small compared to the simulated light source area. Parker et al. implemented

this scheme using standard ray tracing. In this case, it is a comparatively easy task to compute the extended hulls for primitives like spheres and triangles, and ray intersection directly calculates the distances to the outer and inner boundaries, which are used to compute a corresponding attenuation factor.

Although it was shown that the algorithm only introduces about 20% of computation overhead (compared to normal ray tracing), it is still far from being suitable for interactive rendering. Especially when it comes to more complex scenes, too much computation is spent on extending the geometric primitives and computing attenuation factors that later will be discarded.

In the following sections we will show that this method can be adopted to work on sampled data (depth maps) in a much more efficient manner, while still achieving good shadow quality.

3.2 A Sampling Based Approach

Just like the traditional shadow map algorithm presented in [20], we start with the computation of two depth images, one taken from the view of the point light source and one taken from the camera. To compute hard shadows we simply have to compare the transformed z value of each frontmost pixel (as seen by the camera) to the corresponding entry in the light source depth map, according to the following algorithm:

```
foreach(x, y) {
  P = (x, y, depth_camera[x, y])
  P' =warp_to_light(P)
  if(depth_light[P'_x, P'_y] < P'_z)
    pixel is blocked
  else
    pixel is lit
}
```

To modify this method to add an *outside* penumbra region, we have to extend the `else` branch of the shadow test to determine if the pixel is really lit or lies in a penumbra region. According to the ray tracing scheme explained in the previous section, we have to trace back the ray from the surface point towards the light source and see if any outer object is intersected. If we consider the light source depth map as a collection of *virtual layers*, where each layer is a binary mask describing which pixels between the light and the layer got blocked by an occluder inbetween (hard shadow test result), we can simulate the intensity fall-off caused by an area light source by choosing the nearest layer to P'_z that is still in front, and compute the distance between (P'_x, P'_y) and the nearest blocked pixel in that specific layer. This is in a sense similar to Parker's method since finding the minimum distance corresponds to intersecting the outer hull and computing the distance to the inner boundary. The main difference is of course that we use a sampled representation containing all possible occluders rather than the exact geometry of only one occluder.

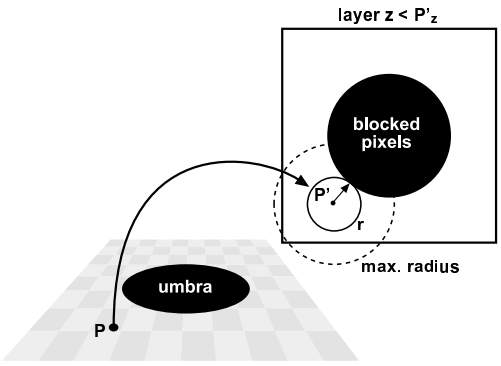

Figure 2: Projecting and searching for the nearest blocker pixel.

Figure 2 illustrates the search scheme using a very simple setup consisting of the umbra generated by an ellipsoid as an occluder and a ground plane as the receiver polygon. For a given point P which does not lie inside the umbra, we first warp P to the view of the light source (P'). Since the original point P was somewhere near the umbra, we find the transformed point P' in the neighborhood of the blocker image which causes the umbra. To calculate an attenuation factor for P, we start searching the neighborhood of P' till we either find a blocked pixel or a certain maximal search radius r_{max} is exceeded. The attenuation factor f is now simply the minimal distance r (or $r = r_{max}$ if no blocking pixel is found) divided by the maximal radius r_{max}. So $f = \frac{r}{r_{max}}$ rises up from 0 (no illumination) to 1 (full illumination) as the distance to the blocker increases.

In other words, we can now generate smooth shadow penumbra regions of a given maximal extent r_{max}. To simulate the behavior of a *real* area light source, we now have to define which properties affect the size of the penumbra and how these can be realized with our search scheme. As widely known, the following two distances mainly define the extend of the penumbra[1]:

- the distance between occluder and receiver, and
- the distance between receiver and light source.

For our search scheme the distance between receiver and light source can be integrated by varying r_{max} according to the distance between a given surface point P and the light source position. Assuming a fixed occluder,

[1] Apart from other properties like orientation of receiver and light source etc.

a receiver far away from the light source will get a larger penumbra whereas a receiver near to the light source will have a smaller r_{max} assigned.

Taking into account the distance between occluder and receiver is a little bit tricky: Since finding an appropriate occluder is the stop criterion for our search routine, we do not know in advance what this distance will be. What we do know is that the occluder has to be inside the region determined by the maximal extend, which is computed using the distance between receiver and light source.

In other words, the final r_{max} may be less the initial search radius. For our search routine this means that we first search the maximal extend since an occluder pixel is found and then re-scale the initial search radius by a factor computed using the distance between the surface point P and the found occluder pixel and use this r_{max} as the denominator for computing f (attenuation factor).

Assuming that the position of the point light in light source space is located at $(0, 0, 0)$ and that the light direction is along the z axis, we set the inital search radius

$$r_{max} = r_{scale} * |P'_z| + r_{bias} \quad ,$$

where r_{scale} and r_{bias} are user defined constants describing the area light effect[2]. Since shadow maps are usually generated for the very limited cut-off angle of spotlights, the difference of using P'_z instead of computing an euclidean distance is negligible. We can now rewrite the hard shadow algorithm to produce soft shadows by simply adding this search function:

```
foreach(x, y) {
  P = (x, y, depth_camera[x, y])
  P' = warp_to_light(P)
  if(depth_light[P'_x, P'_y] < P'_z)
    pixel is blocked
  else
    f = search(P')
    modulate pixel by f
}
search(P') {
  r = 0
  r_max = r_scale * |P'_z| + r_bias
  while(r < r_max) {
    if ∃(s,t) : ||(P'_x, P'_y) − (s,t)|| = r ∧
        depth_light[s,t] < P'_z {
      r_max* = r_shrink * (P'_z − depth_light[s,t])
      return clamp_{0,1}(r / r_max)
    }
    else
      increase r
  }
  return 1.0
}
```

[2] r_{bias} can be used to force a certain penumbra width even for receivers very near to the light source.

In the first loop we iterate over all frontmost pixels as seen by the camera performing the hard shadow test. For each lit pixel we start a search routine where we search in the light source depth map in order to find a suitable blocker pixel at a minimal distance to the transformed surface point. If a blocker pixel is found we then re-scale the inital r_{max} by a factor computed using the distance between the surface point and the occluder pixel. An user-defined scaling factor r_{shrink} is used to give additional control on the effect of this distance.

As can be seen in the pseudo code the described *virtual layers* are implicitly selected by processing only those pixels in the depth map where a blocker lies in front of the potential receiver (depth_light$[s,t] < P'_z$).

Up to now we have restricted ourselves to a very simple setup where the receiver was parallel to the light source image plane. This has the effect that P'_z remains constant during the soft shadow search, or in other words, the search takes place in a constant virtual layer. This is no longer the case if we consider an arbitrary receiver as depicted in Figure 3.

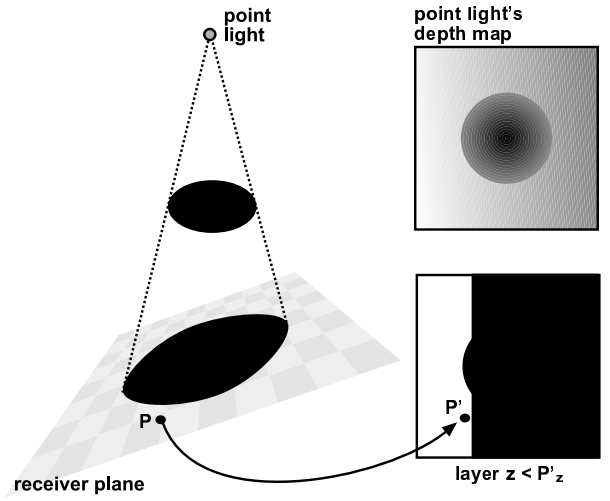

Figure 3: Wrong self shadowing due to layer crossing.

If we performed a search on the constant layer $z < P'_z$ we would immediately end up in the receiver's own shadow since the receiver plane may cross several of the virtual layers. This can be seen in the virtual layer image in Figure 3 where about two thirds of the layer contain blocked pixels belonging to the receiver polygon.

To solve this problem, we either have to divide the scene into disjunct occluders and receivers[3], which would make the algorithm only suitable for very special situations, or we need to supply more information to the

[3] Which is e.g. suitable for games where a character moves in a static environment.

search routine. To define an additional search criterion, which gives the answer to the question "does this blocker pixel belong to me?", we follow Hourcade's [12] approach and assign object IDs. These IDs are identification numbers grouping logical, spatially related, objects of the scene.

It must be pointed out that all triangles belonging to a certain object in the scene must be assigned the same object ID, otherwise self shadowing artifacts would occur if the search exceeded the projected area of the triangle belonging to P. Of course there are situations where also the ID approach fails, e.g. if distinct objects are nearly adjacent, but for most real-time applications there should exist a reasonable grouping of objects.

3.3 Handling of Hard Shadow Regions

Up to now we have concentrated on the computation of the *outer* part of the hard shadow region and simply assumed that the hard shadow region is not lit at all. In the case of an area light source, which we would like to simulate, this is of course an indefensible assumption. What we would like to obtain is of course a penumbra region which also smoothes this *inner* region. This can be easily achieved if we apply the same search technique for pixels that are initially blocked by an occluder. Instead of searching for the nearest blocker pixel within a given search radius we now have to search for the nearest pixel that is lit by the light source.

To combine this with the *outer* penumbra result we assume that *outer* and *inner* regions meet at an attenuation value of 0.5 (or some user defined constant). The final algorithm (including the object ID test) that produces penumbra regions can then be implemented according to the following pseudo code:

```
foreach(x, y) {
  P = (x, y, depth_camera[x, y])
  P' = warp_to_light(P)
  P'_ID = id_camera[x, y]
  inner = depth_light[P'_x, P'_y] < P'_z
  f = search(P', inner)
  modulate pixel by f
}
search(P', inner) {
  r = 0
  r_max = r_scale * |P'_z| + r_bias
  while(r < r_max) {
    if inner
      if ∃(s,t) : ||(P'_x, P'_y) − (s,t)|| = r  ∧
           depth_light[s,t] >= P'_z  ∧
           id_light[s,t] == P'_ID
        r_max* = r_shrink * (depth_light[s,t] − P'_z)
        return 0.5 * clamp_{0,1}(r / r_max)
    else
      if ∃(s,t) : ||(P'_x, P'_y) − (s,t)|| = r  ∧
           depth_light[s,t] < P'_z  ∧
           id_light[s,t] ≠ P'_ID
        r_max* = r_shrink * (P'_z − depth_light[s,t])
        return 1.0 − 0.5 * clamp_{0,1}(r / r_max)
    increase r
  }
  return inner ? 0.0 : 1.0
}
```

3.4 Discussion

The presented algorithm is capable of producing perceptually pleasing, rather than physically correct soft shadows using a total of four sampled images of the scene (two object ID maps, two depth maps). The behavior (extent) of the area light can be controlled by user defined constants. Using unique object IDs to group primitives into logical groups, soft shadows are computed for every occluder/receiver combination not sharing the same object ID.

4 Implementation

4.1 Generating the Input Data

Since our algorithm relies on sampled input data, graphics hardware can be used to generate the input data needed for the shadow computation. In a first step we render the scene as seen by the light source and encode object IDs as color values. For very complex scenes we either use all available color channels (RGBA) or restrict ourselves to one channel (alpha) and assign object IDs modulo 2^n (n bits precision in the alpha channel). This gives us the depth map (z-buffer) and the object IDs of the frontmost pixels according to the light source view, which we transfer back to the host memory. We then repeat the same procedure for the camera view. If only the alpha channel is used for encoding the object IDs, we can combine this rendering pass with the rendering of the final scene (without shadows).

In cases where 8 bits are enough we could also use a special depth/stencil format available on newer NVIDIA GeForce cards. With this mode we simply encode IDs as stencil values and obtain a packed ID/depth map (8 bits stencil, 24 bit depth) using only one frame buffer read. Another benefit of this format is that memory accesses to id/depth pairs are more cache friendly.

4.2 Shadow Computation

The actual shadow computation takes place at the host CPU. According to the pseudo code in Section 3.3, we iterate over all pixels seen by the camera and warp them to the light source coordinate system. Next we start searching for either the nearest blocker pixel (*outer* penumbra

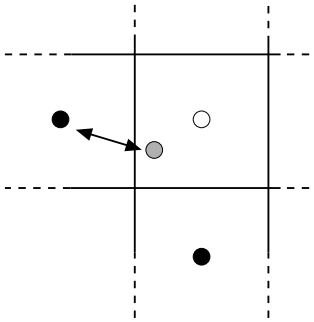

Figure 4: Computing distances at subpixel accuracy.

region) or the nearest pixel that is lit (*inner* penumbra region).

Since memory accesses (and the resulting cache misses) are the main bottleneck, we do not search circularly around the warped pixel but search linearly using an axis aligned bounding box. Doing so we are actually computing more than needed but this way we can utilize SIMD (single instruction, multiple data) features of the CPU, e.g. MMX, 3DNow, or SSE, which allows us to compute several r in parallel. If an appropriate blocking pixel is found (object ID test, minimal distance), we store the resulting attenuation factor for the given camera space pixel. If the search fails, a value of 1.0 or 0.0 is assigned (full illumination, hard shadow).

At the end, the contribution of the point light source is modulated by the attenuation map using alpha blending.

4.3 Improvements

Subpixel Accuracy

When warping pixels from camera to light there are two ways to initialize the search routine. One would be to simply round (P'_x, P'_y) to the nearest integer position and compute distances using only integer operations. While this should give the maximum performance, the quality of the computed penumbrae would suffer from quantization artifacts. Consider the case where pixels representing a large area in camera screen space are warped to the same pixel in the light source depth map. Since all pixels will find the same blocker pixel at the same distance, a constant attenuation factor will be computed for the whole area. This can be avoided by not rounding to the nearest integer but performing the distance calculation at floating point precision. As depicted in Figure 4, we compute the distance of the warped pixel (grey) to the next blocker pixels, which lie at integer position. Quantization artifacts can be further reduced if we also slightly jitter the integer position of the blocker pixels. In practice we observed that the latter is only needed for very low resolution depth maps.

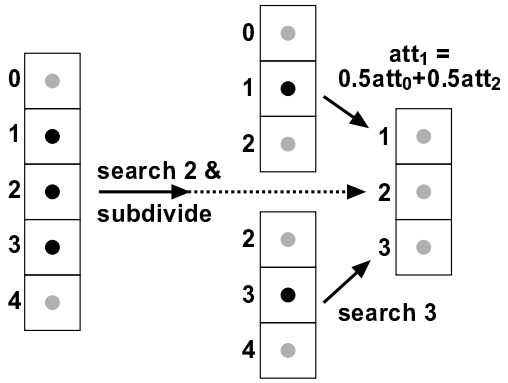

Figure 5: Subdivision and interpolation.

Adaptive Sampling

Up to now we only briefly discussed the cost of searching the depth map. Consider a scene where only 5% of the frontmost pixels are in hard shadow. To compute accurate penumbra regions we would need to perform neighborhood searches for 95% of the pixels in the worst case[4]. So for all completely lit pixels we have searched the largest region (r_{max}) without finding any blocker pixel. Even with a highly optimized search routine and depth maps of moderate size it would be very difficult to reach interactive frame rates.

Instead we propose an interpolation scheme that efficiently reduces the number of exhaustive searches needed for accurate shadowing. The interpolation starts by iterating over the columns of the camera depth map. In each iteration step, we take groups of 5 pixels and do the hard shadow test for all of them. Additionally, we also store the corresponding object IDs of the blockers, or, in the case of lit pixels, the object ID of the receiver pixel. Next, we perform a soft shadow search for the two border pixels in this group. As a criterion for the inner pixels we check if

- the object IDs are equal and
- the hard shadow test results are equal.

If this is true, we assume that there will be no dramatic shadow change within the pixel group and simply linearly interpolate the attenuation factors of the border pixels across the middle pixels. If the group test fails we refine by doing the soft shadow search for the middle pixel and subdivide the group into two three pixel groups for which we repeat the group test, interpolation and subdivision.

Figure 5 shows an example of such an interpolation step. Let us assume that the object ID of pixel 3 differs from the rest. In the first phase we perform hard shadow

[4]Worst case occurs when all pixels are in the view of the light source.

tests for all pixels and soft shadow searches for the two border ones. Since the interpolation criterion fails (IDs not equal), the pixel group is refined by a soft shadow search for pixel 2 and subdivided into two groups. Pixel group $(0, 1, 2)$ fulfills the criterion and an interpolated attenuation factor is assigned to pixel 1, whereas for pixel group $(2, 3, 4)$ we need to compute the attenuation by search. As we will later also need object IDs for interpolated pixels, we simply use the object ID of one interpolant in that case. We repeat this for all pixel groups in this and every 4th column, leaving a gap of three pixel in horizontal direction.

Having linearly interpolated over the columns we now process all rows in the same manner and fill up the horizontal gaps. This bi-linear interpolation mechanism is capable of reducing the number of expensive searches. In the best case, the searching is only done for one pixel in a 16 pixel block. Since this case normally occurs very often (e.g. in fully illuminated areas), we can achieve a great speed-up using the interpolation. On the other hand, quality loss is negligible or non-existent because of the very conservative refinement.

The size of the pixel group used for interpolation should depend on the image size. In experiments we observed that blocks of 4×4 pixels are a good speed/quality tradeoff when rendering images of moderate size (512×512, 800×600 pixels), whereas larger block sizes may introduce artifacts due to the non-perspectively correct interpolation.

5 Results

We have implemented our soft shadow technique on an Intel Pentium 4 1.7GHz computer equipped with an NVIDIA GeForce3 graphics card. Since the generation of depth and ID maps is done using graphics hardware, we get an additional overhead due to the two frame buffer reads needed to transfer the sampled images back to host memory.

Figure 7 shows the results of our soft shadow algorithm for a very simple scene consisting of one torus (occluder) and a receiver plane. We rendered the same scene three times varying only the position and orientation of the occluder.

All images in Figure 7 were rendered using an image resolution of 512×512 pixels and a light depth/ID map resolution of 256×256 pixels. By default, we always use the full image resolution when computing the depth and ID map for the camera view. Frame rates for this scene are about $10 - 15fps$. Computing only the hard shadows (shadow test done on the host CPU) the scene can be rendered at about $30fps$.

In the left image r_{max} was set to 20 (20 pixel search radius) for the inner and outer penumbra. The receiver plane does not reach this maximum (due to the distance between receiver and light source). The average search radius for pixels on the receiver plane is about 16 pixels. The effect of increasing or decreasing r_{max} for this scene is plotted in Figure 6. It must be pointed out that the distance between occluder and receiver does not affect the inital search radius. Therefore the cost of computing soft shadows for the three images in Figure 7 is nearly constant.

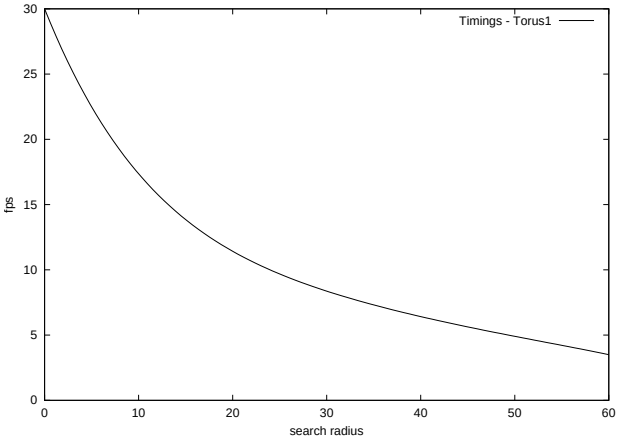

Figure 6: Frame rates for the torus test scene (Figure 8)

In the left image artifacts can be seen (ring), where the inner and outer penumbra meet. This is because the attenuation factors for inner and outer regions are computed in a slightly different way (see Section 3.3). Theoretically this transition should be smooth.

Figure 8 (left) shows a more crowded example scene with objects placed at various heights. It can be seen that objects very near to the floor plane cast very sharp shadows, whereas the shadows from the three tori are much smoother. The other two images in Figure 8 show the scene with hard shadows and hard shadows with outer penumbra. Since our soft shadow algorithm is based on the shadow map technique, we are independent of the scene geometry, which means we can generate soft shadows for arbitrary geometry. There is no distinction between receiver and occluder objects (apart from the missing self shadowing due to the ID test).

Figure 9 shows two more complex scenes where we used our soft shadow algorithm for penumbra generation. In order to assign reasonable object IDs we simply group polygons using the tree structure obtained when parsing the scene file. This way all polygons sharing the same transformation and material node are assigned the same object ID. Both images were taken using a low-

resolution light depth/ID map of 256×256 pixels and an image resolution of 512×512 pixel. In the right image we choose a very large cutoff angle for the spotlight which would normally generate very coarse hard shadows. Here the subpixel accuracy explained in Section 4.3 efficiently smoothes the shadows. Both images can be rendered at interactive frame rates ($\approx 15fps$).

Note that all the timings strongly vary with the size of the penumbra, so changing the light position or altering r_{max} may speed up or slow down the computation, depending on the number of searches that have to be performed. When examining the shape of the penumbrae, one can observe that they do not perfectly correspond to the occluder shape. This is due to the circular nature of the search routine, which rounds off corners when searching for the minimal distance.

6 Conclusions and Future Work

In this paper we have shown how good-looking, soft penumbra regions can be generated using only information obtained from a single light source sample. Although the method is a very crude approximation it gives a dramatic change in image quality, while still being computationally efficient. We showed how the time consuming depth map search can be avoided for many regions by interpolating attenuation factors across blocks of pixels. Since the algorithm works on sampled representations of the scene, computation time depends mostly on the shadow sizes and image resolutions and not on geometric complexity, which makes the method suitable for general situations.

In it is current state the algorithm still relies on a number of user parameters (r_{max}, r_{shrink}, etc.) which where introduced *ad-hoc*. As future work we would like to to hide these parameters and compute them based on one intuitive parameter (e.g. the radius of a spherical light source, defined in the scene's coordinate system). This way it would also be possible to compare our method to more accurate algorithms.

With real time frame rates as a future goal, another focus will be on more sophisticated search algorithms that work on hierarchical and/or tiled depth maps as well as investigating methods of pre-computed or cached distance information. Further speed improvements could also be achieved by using graphics hardware, e.g. interleaved frame buffer reads, as well as on the host CPU by using special processor instructions sets.

Another research direction will be the quality of shadows. Up to now we simply used a linear intensity falloff, which of course is not correct. Assuming a diffuse spherical light and an occluder with a straight edge (similar to Parker's original algorithm), a better approximation would be a sinusoid as the attenuation function.

Finally, we have only slightly addressed aliasing issues that occur when working on sampled data. Our algorithm can work on very low-resolution image data since the search technique efficiently smoothes blocky hard shadows. However, we expect an additional improvement of quality by using filtering schemes that also take into account the stamp size of the warped pixel or work on super-sampled depth maps.

Acknowledgements

We would like to thank Prof. Wolfgang Heidrich of the University of British Columbia, Canada, and the anonymous reviewers for valuable discussions and comments on this topic.

References

[1] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image-based methods for rendering soft shadows. *Proceedings of SIGGRAPH 2000*, pages 375–384, July 2000. ISBN 1-58113-208-5.

[2] Stefan Brabec and Hans-Peter Seidel. Hardware-accelerated rendering of antialiased shadows with shadow maps. *To appear in: Computer Graphics International 2001*, 2001.

[3] L. S. Brotman and N. I. Badler. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics and Applications*, 4(10):71–81, October 1984.

[4] Yiorgos Chrysanthou and Mel Slater. Shadow volume bsp trees for computation of shadows in dynamic scenes. *1995 Symposium on Interactive 3D Graphics*, pages 45–50, April 1995. ISBN 0-89791-736-7.

[5] Franklin C. Crow. Shadow algorithms for computer graphics. In *Computer Graphics (SIGGRAPH '77 Proceedings)*, pages 242–248, July 1977.

[6] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter S. Shirley, and Rich Riesenfeld. Interactive technical illustration. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 31–38. ACM SIGGRAPH, April 1999. ISBN 1-58113-082-1.

[7] Paul E. Haeberli and Kurt Akeley. The accumulation buffer: Hardware support for high-quality rendering. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 309–318, August 1990.

[8] E. Haines. Soft planar shadows using plateaus. *Journal of Graphic Tools*, 6(1):19–27, 2001.

[9] Paul Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, Carnegie Mellon University, January 1997.

[10] T. Heidmann. Real shadows real time. *IRIS Universe*, 18:28–31, November 1991.

[11] Wolfgang Heidrich, Stefan Brabec, and Hans-Peter Seidel. Soft shadow maps for linear lights. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 269–280, June 2000. ISBN 3-211-83535-0.

[12] J. C. Hourcade and A. Nicolas. Algorithms for antialiased cast shadows. *Computers & Graphics*, 9(3):259–265, 1985.

[13] Brett Keating and Nelson Max. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Rendering Techniques '99 (Proc. of Eurographics Rendering Workshop)*, pages 197–212, June 1999.

[14] Daniel Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics & Applications*, 12(6):25–39, November 1992.

[15] Michael D. McCool. Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics*, 19(1):1–26, January 2000.

[16] Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical Report UUCS-98-019, Computer Science Department, University of Utah, 1998. Available from http://www.cs.utah.edu/vissim/bibliography/.

[17] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 283–291, July 1987.

[18] Marc Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadow and lighting effects using texture mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 249–252, July 1992.

[19] Cyril Soler and François X. Sillion. Fast calculation of soft shadow textures using convolution. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 321–332, July 1998.

[20] Lance Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, pages 270–274, August 1978.

[21] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics & Applications*, 10(6):13–32, November 1990.

Figure 7: A simple test scene showing the effect of varying distance between receiver and occluder.

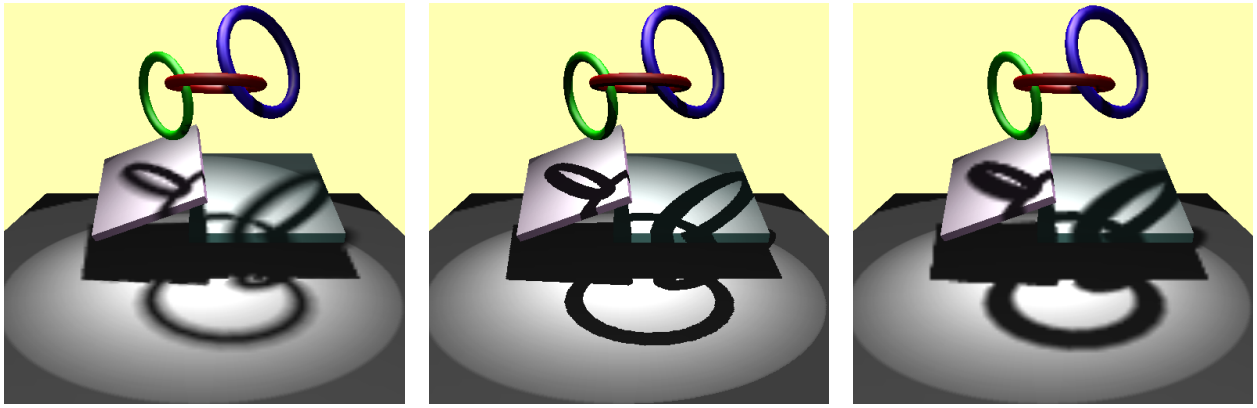

Figure 8: A more crowded scene. Left: soft shadows, middle: hard shadows, right: hard shadows with outer penumbra.

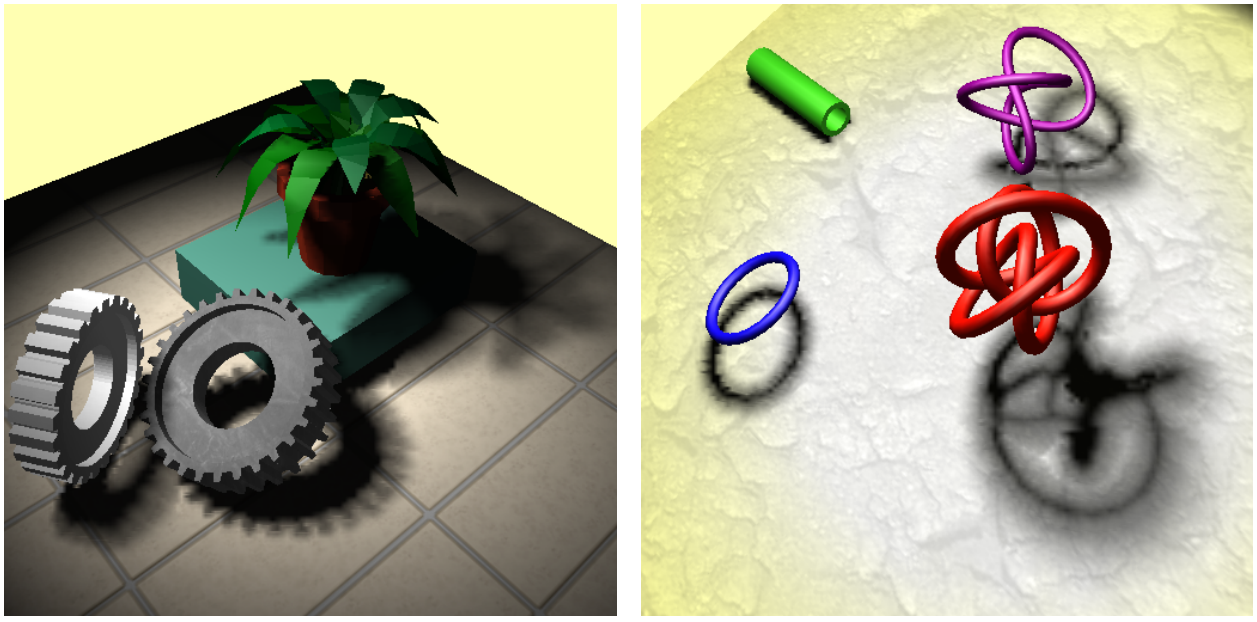

Figure 9: Two more complex scenes rendered with our soft shadow algorithm.