EUROGRAPHICS 2003 / P. Brunet and D. Fellner
(Guest Editors)

Volume 22 (2003), Number 3

Real-Time Caustics

M. Wand and W. Straßer

WSI/GRIS, University of Tübingen

Abstract

We present a new algorithm to render caustics. The algorithm discretizes the specular surfaces into sample points. Each of the sample points is treated as a pinhole camera that projects an image of the incoming light onto the diffuse receiver surfaces. Anti-aliasing is performed by considering the local surface curvature at the sample points to filter the projected images. The algorithm can be implemented using programmable texture mapping hardware. It allows to render caustics in fully dynamic scenes in real-time on current PC hardware.

Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture / Image Generation – Display Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

A real-time simulation of the interaction of light with complex, dynamically changing scenery is still one of the major challenges in computer graphics. In this paper, we look at a special global illumination problem, rendering of caustics. Caustics occur if light is reflected (or refracted) at one or more specular surfaces, focused into ray bundles of a certain structure, and then received as patterns of light on a diffuse surface. Caustics are a subtle effect but for many scenes these subtleties are very important to obtain realistic images. Consider for example a pool of water that reflects the environmental light onto its surrounding or a set of metallic objects like spoons and knifes focusing the incoming light onto the surface of a table.

Currently, the simulation of specular-to-diffuse light transports is often done by some variant of the photon-tracing algorithm[2,3,4,9,12,20,28]: Starting from the light sources, photons are shot into the scene and their interaction with specular objects is tracked until they meet a diffuse receiver. On the receiver surfaces, statistical density estimation is used to approximate the local incident flux density. Photon tracing is a very powerful paradigm that allows a physically correct simulation of a large class of light transport problems. However, due to the stochastic nature of the algorithm, it needs a considerable amount of sample photons to achieve a good image quality. Each of the photon traces needs multiple potentially expensive ray intersection queries. After that, a statistical density estimation step has to be performed that is often even more expensive. Thus, the technique is usually not very efficient. Although the implementation techniques for raytracing queries have made impressive advances in the last few years[29], raytracing based algorithms still need a considerable amount of computational power (such as a cluster of several high end CPUs) to calculate global illumination solutions in real time[30].

In this paper, we propose a rasterization based technique that implements caustics rendering using texture mapping: We discretize the specular surfaces into a set of sample points that are treated as reflecting pinhole cameras. Each of these "pinholes" projects an image of the incoming light onto the receiving objects. The images are blurred by an anisotropic filter kernel that is derived from the local surface curvature. To allow an efficient implementation, the algorithm considers only cases with a single specular interaction on each light pass and assumes that the light sources are far away from the specular reflectors. The spatial relationship between the reflectors and the receivers can be arbitrary. An advantage of the rasterization based strategy is that the algorithm does not need involved data structures and no preprocessing. Thus, it is fairly simple to implement robustly and, more important, it can be applied to fully dynamic scenes. The light emitters, specular reflectors and receivers can be changed arbitrarily without additional costs. The coherent, rasterization based strategy also allows for an efficient implementation using modern pro-

grammable graphics hardware: Our implementation uses current consumer PC hardware according to the DirectX 9 specifications. It is able to render caustics in scenes with fully dynamic receivers and reflectors and complex, dynamically changing lighting environments in real-time.

In the following, we will first survey related work in section 2. Then we will describe the new algorithm more in detail in section 3 and present results in section 4. The paper concludes with some ideas for future work in section 5.

2. Related Work

The standard technique for rendering caustics is photon tracing. The algorithm was introduced by Arvo[2] in 1986. To calculate caustics, it shoots a set of (random) photons from the light sources into the scene and tracks their path until they meet a diffuse surface. This is usually more efficient for rendering caustics than raytracing from the view point[15]. On the receiving surfaces, a density estimation technique is needed to recover continuous irradiance values from the discrete photon hits. Heckbert[9] proposes a technique that accumulates photon statistics in adaptive texture maps. Chen et al.[3] use a map corresponding to the viewport pixels and an adaptive reconstruction filter based on the radius of the k nearest neighbors. Jensen[12,13] constructs a photon map, i.e. a spatial hierarchy that stores the results of the photon tracing and allows efficient neighborhood queries. For caustics, the irradiance at a surface point is computed by estimating the surface area covered by its k nearest neighbors. The advantage of the nearest neighbor techniques is a constant noise ratio independent of the local photon density. Ma et al. propose a variant of the algorithm based on hash tables that allows for an efficient hardware implemenation[21].

Collins[4] uses extended ray cones to enhance the accuracy of the photon tracing: The method tries to estimate the footprint of the reflected rays using finite differencing. A more general framework for ray footprint estimation is presented by Ighey[10]. Our antialiasing scheme is based on similar ideas. Watt[33] proposes a backward beam tracing technique for polyhedral scenes. Another possibility of improving the photon tracing technique is bidirectional path tracing[20,28]: Two paths from the viewpoint and the light source are shot simultaneously and connected via shadow rays. Granier et al.[8] and Dmitriev et al.[6] describe global illumination techniques based on photon tracing that allow for efficient dynamic updates.

Today, it is possible to perform raytracing based global illumination algorithms on commodity hardware in interactive settings[29,30]. However, a cluster of several PCs is still needed to obtain fluid frame rates. Our rasterization based technique for rendering caustics is less flexible than the raytracing based techniques but it delivers fluid frame rates on a single PC with a modern graphics accelerator board.

Sloan et al.[26] describe an approach for static scenes with dynamically changing lighting conditions: The dependency between surface irradiance and global lighting is precomputed and stored in a spherical harmonics representation. The method can render low frequency caustics with correct shadowing under changing lighting conditions in real-time. Our technique neglects shadowing but it is able to handle fully dynamic scenes without preprocessing and without the restriction to low frequency effects.

Besides general purpose techniques for global illumination and caustic rendering, there are also some techniques for special cases such as underwater scenes[11,14,23]. The technique of Trendall et al.[19] calculates underwater caustics on a planar receiver using programmable graphics hardware.

Our algorithm was inspired by the “Instant Radiosity” algorithm by Keller[17]: It computes diffuse radiosity solutions by accumulating multiple images with shadows from single point light sources.

Our technique is also related to point based rendering: The techniques for sampling the reflector surfaces are taken from point based rendering literature[24,25,27,31,32]. We use the “differential point sample” representation by Kalaiah et al.[16] to estimate the texture footprints for the illumination environment maps.

In the results section, we show examples of caustics from natural light environment maps. The idea of using high dynamic range photographs for lighting was introduced by Debevec[5]. The environment maps in our example scenes are taken from his web site.

3. The Algorithm

The idea of our algorithm is based on a well-known observation: If you light a reflective disco ball in a room with a light bulb you obtain several small light spots on the walls that just turn out to be blurred images of the light source. The reason for this effect is that the facets of the disco ball act as a set of (reflecting) pinhole cameras that project an image of the light source onto the receiving surfaces. This observation leads to the following strategy for rendering caustics:

We discretize the surface of the specular reflector into small regions (analogous to the facets of the disco ball) and treat each as a pinhole camera that projects the incoming light onto the surrounding objects. This sampling ap-

proach is prone to aliasing artifacts: If you mirror a point light source at a set of surface points on a smooth surface you obtain a set of bright points on the receiver instead of a continuous caustic. To avoid these undersampling artifacts, we must examine the local curvature at the sample points on the reflector to determine an appropriate resampling filter for the projected images. Taking surface curvature into account, we will obtain different results for a disco ball with first-order discontinuities on the surface and a smooth sphere.

3.1. Texture Mapping

The idea of projecting images of the light sources onto the receivers using reflective or refractive pinhole cameras can be mapped easily to programmable texture mapping hardware (see Figure 1). The first step is to represent the incoming light as an environment map. If all light sources are far away from the specular object, it is sufficient to use one global environment map for all sample points at once. If the light sources become closer to the reflector, it might be necessary to compute multiple environment maps, which increases the computational efforts. In our implementation, we use only one map for all sample points, which is sufficient for most cases. We use a cube environment map that can be generated easily on-the-fly from the scene geometry using z-buffer rendering.

Now we assume that we are given a set of sample points on the specular object that act as pinhole cameras. Thus, the next step is to compute the caustics by adding together multiple images reprojected from the sample points. To do this, we render the receiving surfaces using a z-buffer renderer and a pixel shader program that calculates each projection. It takes the vector d from the current raster position on the receiver p_d to the sample point p_s and reflects it at the normal n_s of the point sample. The resulting reflection vector r is then used as direction vector to do the cube map texture lookup (Figure 1). The resulting color value has to be scaled to account for correct light attenuation. The point samples have to be treated as infinitesimal area reflectors. Each point sample represents a small piece of surface area that reflects light onto the receiver. This means, we must multiply the texture value by the cosines of the incident angles at the reflector and the receiver and divide by the distance:

$$\frac{\cos\alpha_s \cdot \cos\alpha_d}{dist^2} = \frac{\langle n_s, -d_n \rangle \cdot \langle n_d, d_n \rangle}{\langle d, d \rangle} \quad (d_n = d/\|d\|)$$

The scalar products are clamped to the range [1,0] to perform backface culling. To avoid singularities, we add a small positive constant to the denominator. This constant should be in the range of the diameter of the surface region that is represented by one sample point squared. This accounts for the uncertainty of the position (and thus of the distance d) due to the discrete sampling. Additionally, we have to normalize the light contribution by dividing the total surface area by the number of sample points.

The computation of the lookup vector, the texture lookup, and the computation of the attenuation term is implemented as a pixel shader. We employ DirectX9 2.0/2.x pixel shaders that allow floating point computations in the pixel pipeline and programmable ("dependent") texture lookups[22]. Due to the limitation of the instruction count, it is only possible to consider a fixed amount of reflector samples in one rendering pass. Using pixel shaders version 2.0 that allow 64 arithmetic and 32 texture operations per shader, up to three samples can be handled by the graphics hardware in a single pass. For hardware according to the extended version 2.x standard, more samples can be considered (the maximum instruction count is here 1024 and the instruction set includes loops). For additional rendering passes, we draw the receiver geometry again and accumulate the additional contribution in the frame buffer using additive blending.

3.2. Sampling

To perform the algorithm outlined above, we need sample points on the reflector surface that serve as pinhole cameras. We use a uniform sampling, i.e. we try to place sample points uniformly over the surface. This is done because the light contribution of all sample points should be similar. This avoids adding images with low contribution to the final result and thus leads to a better convergence. However, the contribution of the sample points also depends on the orientation of the two surfaces considered and their distance. Thus, a more elaborated sampling scheme should also consider the normal vectors for sample placement and use a spatial hierarchy to assign sample points to receivers adaptively. These optimizations have not been implemented yet but will be subject of future work.

To obtain sample sets with uniform density on the reflector surfaces, we employ well known techniques from point sample rendering. Four alternatives have been examined (see e.g. Wand et al.[32] for a more detailed analysis of the strategies):

1) Random[27,31]: The sample points are chosen at random with uniform probability across the surface of the reflector. This strategy is very easy to implement but leads to relatively large noise artifacts in the solution.

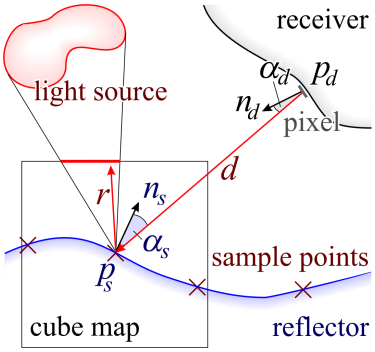


Figure 1: Rendering caustics using texture mapping

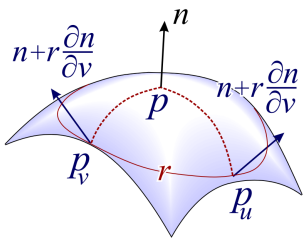


Figure 2: Differential point sample

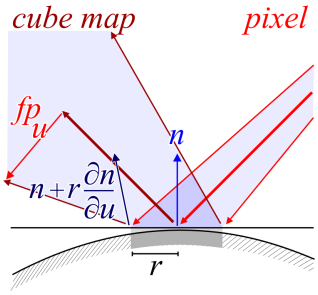


Figure 3: Footprint estimation

2) Grid-stratification[31]: We use a high density random sample set and fix a three dimensional grid of cubes in space. In each grid cell, we chose only one of the sample points and delete all others. The method is comparable to jittered sampling[7] in stochastic raytracing but we delete rather than explicitly place points using the grid. The stratification leads to a more uniform sample distribution and thus improves the image quality. However, the sample selection is biased: Surfaces parallel to the grid obtain more sample points than surfaces in a diagonal direction.

3) Quantized-grid stratification[31]: The same as strategy 2, but all sample points are quantized to the grid cells. The noise is reduced further but now it is possible to obtain aliasing artifacts. Additionally, the strategy can lead to considerable displacements of the points for small sample sets. Although the strategy is efficient for point based rendering, it is unfavorable for our application.

4) Neighborhood-based stratification[32]: Starting from a random sample set with high density, we remove superfluous points in a greedy manner. If a point is still covered by a sphere around another point with fixed diameter, it is regarded as superfluous and can be removed (similar to Poisson disc sampling[7] used for stochastic raytracing). This strategy leads to a more uniform distribution than the grid based techniques, avoids structured aliasing and has no bias. Therefore, we use it in all example scenes in the following sections.

For dynamic scenes, the sampling has to be performed at every frame. The initial random sampling can be done even with complex reflector models in reasonable time. For a complex stratification technique like method (4), the running time of the stratification step might be an issue. However, it depends only on the number of sample points, not on the model complexity. As we only chose a small number of sample points (due to the costs of the light source reprojection and blending), this is usually no performance problem.

3.3. Anti-Aliasing

Up to now, the algorithm still has an important drawback: In many cases, one can still see that the caustic is an overlay of multiple sample images and not continuous. This is an aliasing artifact that is caused by the finite sampling of the reflector surface. In order to avoid this effect, we must apply an appropriate resampling filter for the texture lookups. Theoretically, we would have to consider the reflection vectors at all surface points in the region around each sample point to calculate the exact texture footprint (i.e. the region in the texture that is mapped to the current pixel). In order to implement the filtering using graphics hardware, we must make an approximation:

Modern programmable graphics hardware according to the pixel shader 2.x standard offers programmable anisotropic texture lookups: The lookup instruction ("`texldd`") takes three arguments: a position texture and two gradients fp_u and fp_v in u and v direction. The three vectors define a parallelogram in texture space and the hardware tries to approximate the integral over this region by choosing a suitable mipmap level and then performing a footprint assembly[18]. The technique can also be used for cube maps using three dimensional gradient vectors.

As the local curvature determines the broadening or focusing of the reflected ray bundle, we use a second order surface approximation around each sample point ("differential" point samples[16]) to determine the footprint parallelogram. To estimate the local curvature, we take a denser set of sample points in the region around the main sample point. We determine a tangent plane by computing the average normal and establish a local coordinate frame u,v in this plane. We project all sample points into the plane

and examine the normals at these points. Using linear regression, we fit a linear function $n(u,v) = n + u\cdot\partial n/\partial u + v\cdot\partial n/\partial v$ to the sample points, yielding a matrix $\nabla n = (\partial n/\partial u \,|\, \partial n/\partial v)$.

The footprint parallelogram is then estimated by finite differencing: We calculate the reflection vector at three points: the center point p and two points p_u, p_v on the tangential plane of the surface with a distance r to the center where r is half the sample spacing of the point set (Figure 2). The two distant points are chosen in orthogonal directions, according to the minimum and maximum curvature of the surface. The directions are computed by an eigenspace transformation of the matrix ∇n (yielding a modified coordinate frame u,v). The gradients fp_u and fp_v are then set to the differences of the reflection vectors at the distant points $p_u = p+u$, $p_v = p+v$ to the reflection vectors at the center point (Figure 3). To calculate the reflection vectors at p_u, p_v, the normals $n_u = n + r\cdot\partial n/\partial u$ and $n_v = n + r\cdot\partial n/\partial v$ are used.

4. Results

In this section, we describe the results obtained with a prototype implementation of our algorithm. The implementation uses DirectX 9 pixel shaders to perform the texture mapping operations. We tested two variants of the algorithm: The first uses pixel shader version 2.0 instructions and does not perform antialiasing. This variant runs in hardware on an ATI Radeon 9700Pro graphics board. The second variant uses pixel shaders version 2.x instructions to perform programmable anisotropic texture lookups as described in section 3.3. This variant cannot be executed in hardware on the Radeon 9700 board. The pixel shader 2.x instruction set is supported by nVidia GeForceFX graphics boards that have become available just recently. However, we were not able to use the "`texldd`" instruction (anisotropic filtering with explicit footprint specification) with cube environment maps on GeForceFX boards. Up to now, we were not able to determine if this is a driver problem or a hardware restriction. Thus, we ran the 2.x version of the algorithm in a software simulation (DirectX reference device, 16× anisotropic sampling). This software simulation is about a thousand times slower than the hardware implementation. Therefore, we were not able to demonstrate interactive applications with antialiasing[*]. We expect that the version with antialiasing will run with a speed roughly comparable to that of the unfiltered version once the corresponding hardware support becomes available.

[*] The accompanying video was also recorded without antialiasing.

The results will be presented in two subsections: The first examines the properties of the different sampling strategies. Here, we use a worst-case scene to make the differences visible. The second subsection describes results from interactive applications.

4.1. Evaluation

In order to evaluate the benefits of the different sampling strategies, we applied the algorithm to a scene with very disadvantageous lighting conditions: A metallic ring lying on a flat table is lit by a point light source from above. The point light source has a solid angle of only about 0.0008 sr. It is a white Gaussian spot of about 5 pixel diameter on a $256^2\times 6$ cube map, using only 1/8000 of the cube map area. In contrast to scenes with complex, extended lighting settings, the texture mapping algorithm is not very efficient for this scene as it mostly processes black texels. However, this "worst-case" scene is well-suited to examine the differences between the different sampling strategies.

Anti-Aliasing: Figure 4 shows the ring scene rendered with 100, 1,000, and 10,000 surface samples on the reflector. The first row was rendered in hardware without antialiasing (135msec, 1.6sec, and 19sec rendering time). The second row was rendered with antialiasing, using the pixel shader 2.x software emulation (4 min, 57 min, and 11h rendering time). The resolution was 512×512 pixel. In all cases, the rendering without antialiasing clearly reveals the discrete nature of the rendering algorithm. Using anisotropic filtering, we already obtain a plausible result for very few samples and the image quality of the 1,000 samples version may be acceptable for interactive applications. Even at the high sampling rate of 10,000 light source samples per fragment, the quality is still strongly improved by the anisotropic filtering.

Sampling Pattern: Figure 5 shows a comparison of the four sampling strategies described in section 3.2. The metallic ring lit by a point light source is again used as benchmark scene with a fixed number of 2,000 reflector sample points. In contrast to Figure 4, we disabled the attenuation term in the intensity calculation to enhance the contrast of the calculated caustic patterns. No antialiasing was performed.

The random sampling strategy leads to an uneven distribution of sample points and thus to a noisy solution (Figure 5a). Using grid based stratification reduces the noise level (Figure 5b and c). Quantization to the grid cells leads to a more uniform sampling pattern but it also introduces aliasing artifacts. It can be seen in the left part of the caustics in Figure 5c that the caustics are composed of discrete rings. Additionally, the area illuminated by the

caustic does not exactly match that of the three other solutions. This is due to the artificial displacements of the quantization step. The best results are obtained with neighborhood based point removal (Figure 5d). The technique produces less noise than the other three strategies and does not suffer from aliasing or displacement problems. Therefore, this technique was used in all other examples, too.

4.2. Applications

In this section, we apply the technique to typical application scenes*. All examples where rendered in real-time using a Radeon 9700Pro graphics board and a 2Ghz Pentium 4 System. Due to the problems with pixel shader 2.x conformant hardware, all scenes had to be rendered without anti-aliasing. We use lighting environments without too much high frequency detail to compensate for the lack of correct filtering.

Natural Light: Figure 6 shows objects that are lit with natural light. The lighting situation was modeled as an environment map created from high dynamic range images (Figure 6a). The images for the environment maps where taken from P. Debevec's web site[5]. Figure 6(b) and (c) show a metallic spoon and Figure 6(d) and (e) show a metallic ring reflecting caustics on the ground. All objects (reflectors and receivers) can be moved interactively with no additional costs (see also the accompanying video). The rendering time is about 6-20 fps (frames per second) for the test cases. It depends linearly on the projected area on the screen and on the number of sample points.

Self reflectance: Figure 6f and Figure 7b show a reflecting teacup. It focuses the light inside and casts caustics on itself. The same technique is also used in Figure 6e. In these cases, the reflector is also receiver (however, only one reflective bounce is evaluated for the caustics). For objects reflecting on themselves, it is important to add the (aforementioned) small constant to the denominator in the distance attenuation term to avoid singularities (section 3.1). Otherwise, these singularities become visible as small bright spots near the sample points. The rendering time is 6.4fps for the ring and 4.9fps for the cup. The cup scene is the slowest example scene because it needs more sample points than the others to achieve a realistic effect.

Dynamic Lighting: A special advantage of our algorithm is that it allows for dynamic scenes with arbitrary changes at every frame. Figure 7c shows a scene with dynamic lighting generated from geometry. The bright spheres in the upper left are the light sources. The light geometry is rendered into an environment map at every frame. As in all other examples, the reflector samples are recomputed at every frame. Thus, it is possible to modify the lighting situation, the reflectors and the receivers interactively at every frame. See the accompanying video for a real-time demonstration.

* Note that the intensity of the caustics has been exaggerated a bit in order to demonstrate the effect.

5. Conclusions and Future Work

In this paper, we presented a new technique for rendering caustics at high frame rates of up to 20 frames per second. Our technique is based on rasterization with texture mapping. This allows an efficient and fairly easy implementation on current programmable graphics hardware. There are some advantages over photon tracing based techniques: The algorithm can handle complex lighting situations with extended light sources at no additional costs. The algorithm does not need a density estimation step, which is often a robustness problem as well as an efficiency problem. It performs an adequate filtering operation based on an analysis of the mapping process rather than based on a statistical estimate. The algorithm can easily be applied to fully dynamic scenes as it does not need any preprocessing.

However, there are still some restrictions. Some of them could be diminished in future work: Firstly, the algorithm does not compute any visibility for the light sources with the exception of backface culling. If the reflectors are far away from the receivers, it is possible to incorporate correct shadowing on the receivers by using a cube map with depth values as shadow map. A second restriction is that the light sources are assumed to be far away from the reflector. If this is not the case, it could be necessary to compute multiple cube maps for different clusters of sample points, requiring additional computation time. The third restriction is that only one specular bounce can be handled. This seems to be an inherent restriction. Except from special cases, it seems not to be possible to handle multiple specular interactions on a light path with our technique.

There are also some possibilities to generalize the technique. It should be straightforward to apply the algorithm to scenes with a single refractive interaction in the light path, such as underwater scenes. The algorithm could also be combined with volume rendering (using textured slices[1]) to obtain volumetric caustics. Another opportunity for a generalization would be the use of more complex BRDFs. To employ a more general local lighting model for the receiving surfaces, one could replace the simple Lambertian cosine law by a more elaborated lighting model (e.g. Phong or a more general, lookup table based model). The BRDF of the reflector could be altered, too: A simple

modulation of the reflected intensity depending on the incident angle (e.g. for a Fresnel term) is straightforward. More general glossy reflections are not so easy. This could probably be achieved by perturbing the reflection directions in the pixel shader. However, this would introduce new aliasing problems that have to be addressed.

Concerning the performance of the algorithm, we believe that it can be improved by using more adaptivity in the sampling step: Firstly, the sampling strategy should considers the surface normals of the reflector. Secondly, clustering of sample points for receivers that are farther away could improve the running time for larger scenes.

References

1. Akeley, K.: RealityEngine Graphics. In: *Siggraph 93 Conference Proceedings*, 109-116, 1993.
2. Arvo, J.: Backward Ray Tracing. In: *Developments in Ray Tracing, SIGGRAPH `86 Course Notes*, 1986.
3. Chen, E., Rushmeier, H. E., Miller, G., Turner, D.: A Progressive Multi-Pass Method for Global Illumination. In: *SIGGRAPH 91 Conference Proceedings*, 164-174, 1991.
4. Collins, S.: Adaptive Splatting for Specular to Diffuse Light Transport, *Proceedings of the Fifth Eurographics Workshop on Rendering*, 119-135, 1992.
5. Debevec, P.E: Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography. In: *SIGGRAPH 98 Conference Proceedings*, 1998.

   High dynamic range images taken from: `http://www.debevec.org/Probes`
6. Dmitriev, K., Brabec, S., Myszkowski, K., Seidel, H.P.: Interactive Global Illumination using Selective Photon Tracing. In: *Rendering Techniques 2002*, 21-34, 2002.
7. Glassner, A. S.: *Principles of Digital Image Synthesis*. Morgen Kaufmann Publishers, 1995.
8. Granier, X., Drettakis, G.: Incremental Updates for Rapid Glossy Global Illumination. In: *Computer Graphics Forum (EUROGRAPHICS 2001), 20(3)*, 2001.
9. Heckbert, P.S.: Adaptive Radiosity Textures for Bidirectional Ray Tracing, *SIGGRAPH 90 Conference Proceedings*, 145-154, 1990.
10. Igehy, H.: Tracing Ray Differentials. In: *SIGGRAPH 99 Conference Proceedings*, 179-186, 1999.
11. Iwasaki, K., Dobashi, Y., Nishita, T.: An Efficient Method for Rendering Underwater Optical Effects Using Graphics Hardware. In: *Computer Graphics Forum, 21(4)*, 2002.
12. Jensen, H.W.: Global Illumination using Photon Maps. In: *Rendering Techniques '96*, 21-30, Springer, 1996.
13. Jensen, H.W.: Rendering Caustics on Non-Lambertian Surfaces. In: *Proceedings of Graphics Interface '96*, 116-121, 1996.
14. Jensen, L.S., Golias, R.: Deep-Water Animation and Rendering. In: *Gamasutra*, September 2001. `http://www.gamasutra.com`
15. Kajiya, J.T.: The rendering equation. In: *SIGGRAPH 86 Conference Proceedings*, 143-150, 1986.
16. Kalaiah, A., Varshney, A.: Differential Point Rendering. In: *Rendering Techniques 2001*, Springer, 2001.
17. Keller, A.: Instant Radiosity. In: *SIGGRAPH 97 Conference Proceedings*, 49-56, 1997.
18. Schilling, A.G., Knittel, G., Straßer, W.: Texram: A Smart Memory for Texturing. In: *IEEE Computer Graphics & Applications*, 32-41, 1996.
19. Trendall, C., Stewart, A.J.: General calculations using graphics hardware, with application to interactive caustics. In: *Rendering Techniques 2000*, 287-298, Springer, 2000.
20. Lafortune, E. P., Willems, Y. D.: Bidirectional Path Tracing. *Proceedings of CompuGraphics*, 95-104, 1993.
21. Ma, V.C.H, McCool, M.D.: Low Latency Photon Mapping Using Block Hashing. In: *Graphics Hardware 2002*, pp. 89-98.
22. Microsoft DirectX9 Software Development Kit. `http://www.microsoft.com/windows/directx`

23. Nishita, T., Nakamae, E.: Method of Displaying Optical Effects within Water using Accumulation Buffer. In: *SIGGRAPH 94 Conference Proceedings,* 1994.

24. Pfister, H., Zwicker, M., van Baar, J., Gross, M.: Surfels: Surface Elements as Rendering Primitives. In: *SIGGRAPH 2000 Proceedings*, 335-342, 2000.

25. Rusinkiewicz, S., Levoy, M.: Qsplat: A Multiresolution Point Rendering System for Large Meshes. In: *SIGGRAPH 2000 Proceedings*, 343-352, 2000.

26. Sloan, P.P., Kautz, J., Snyder, J.: Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. In: *SIGGRAPH 2002 Conference Proceedings*, 2002.

27. Stamminger, M., Drettakis, G.: Interactive Sampling and Rendering for Complex and Procedural Geometry. In: *Rendering Techniques 2001*.

28. Veach, E., Guibas, L.: Bidirectional Estimators for Light Transport. In: *Rendering Techniques '92*, 147-162, 1992.

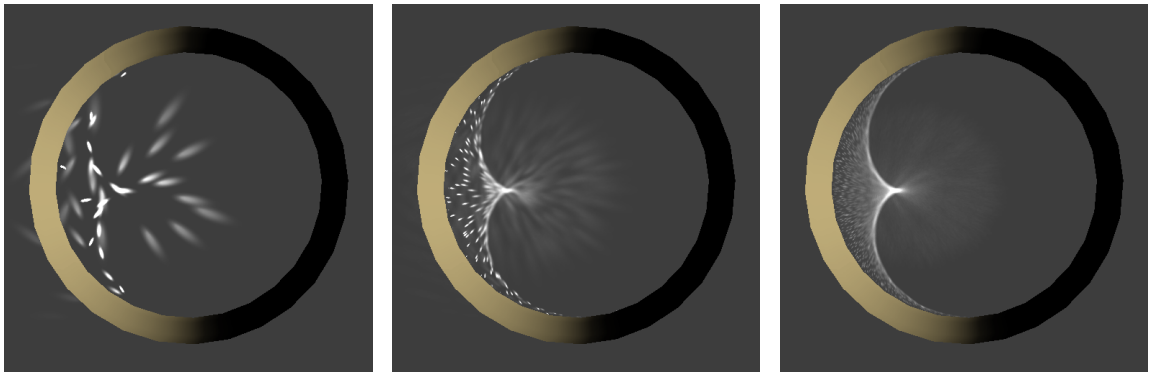
29. Wald, I., Slusallek, P., Benthin, C., Wagner, M.: Interactive Rendering With Coherent Raytracing. In: *Computer Graphics Forum, 20(3)*, 153-164, 2001.

30. Wald, I., Kollig, T., Benthin, C., Keller, A., Slusallek, P.: Interactive Global Illumination using Fast Ray Tracing. In: *Rendering Techniques 2002*, Springer, 2002.

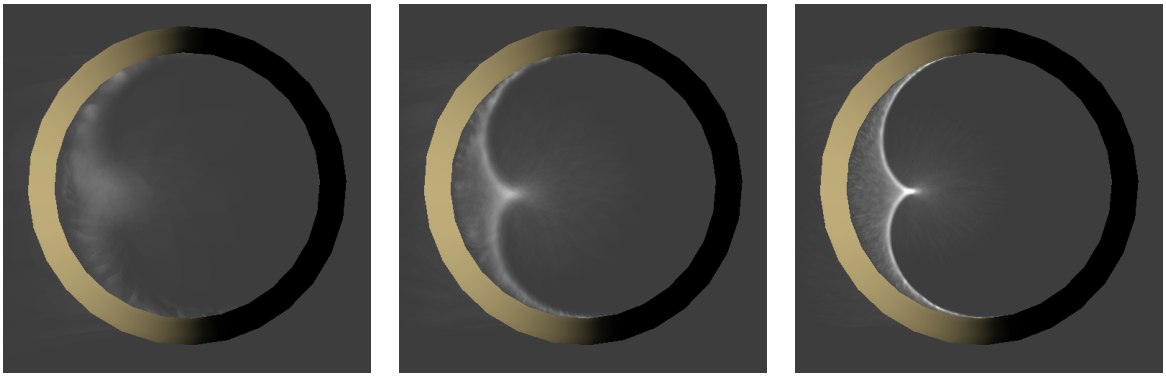
31. Wand, M., Fischer, M. Peter, I., Meyer auf der Heide, F., Straßer, W.: The Randomized z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes. In: *SIGGRAPH 2001 Conference Proceedings*, 361-370, 2001.

32. Wand, M., Straßer, W.: Multi-Resolution Rendering of Complex Animated Scenes. In: *Computer Graphics Forum, 21(3)*, 483-491, 2002.

33. Watt, M.: Light-water interaction using backward beam tracing, *SIGGRAPH 90 Conference Proceedings*, 377-385, 1990.

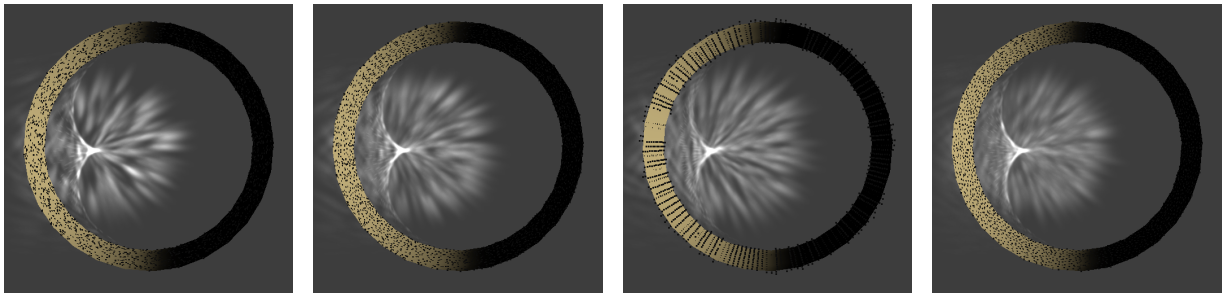


(a) 100 sample points, no filtering **(b) 1,000 sample points, no filtering** **(c) 10,000 sample points, no filtering**



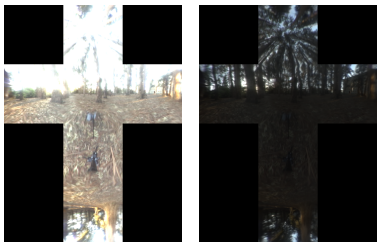
(d) 100 sample points, anisotropic texture lookup **(e) 1,000 sample points, anisotropic texture lookup** **(f) 10,000 sample points, anisotropic texture lookup**

Figure 4: *A worst case scene.* This scene shows a ring illuminated by a point light source of only about 0.0008 sr solid angle (1/8000 of the cube map area). The first row shows conventional texture mapping, the second performs anti-aliasing using anisotropic texture lookups. The sampling artifacts are strongly reduced by the anisotropic filtering strategy.

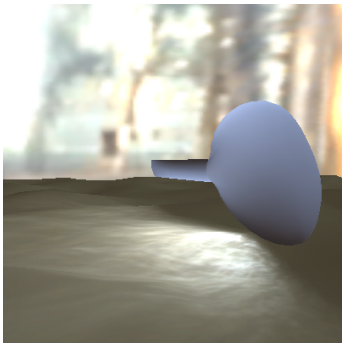


(a) random sampling (b) grid stratification (c) quantized grid stratification (d) neighbor-removal stratification

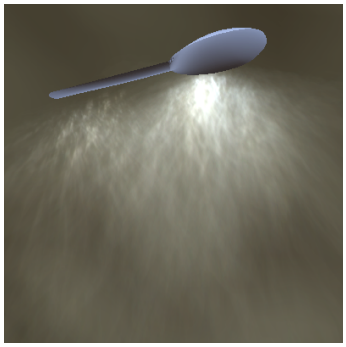
Figure 5: *A comparison of different sampling strategies.* The scene is the same as in Figure 4 but without filtering and with light attenuation by distance disabled to enhance the patterns of the reflected light. 2000 sample points are used. The black dots show the sample points. Strategy (d) yields the best results; thus it is used in all other example images in this paper.



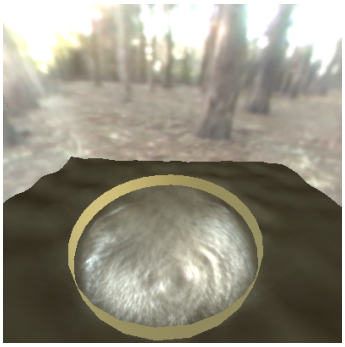
(a) High dynamic range environment maps (forest map taken from [5]). Left: exposure used for the background. Right: exposure used for the caustics.



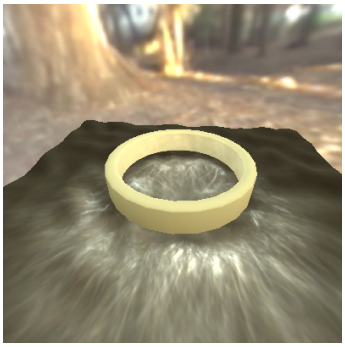
(b) A metallic spoon reflecting the incoming light on the ground. 60 samples / 19 fps.



(c) A metallic spoon. View from above. 60 samples / 15 fps.



(d) A metallic ring reflecting the incoming light on the ground (similar to Figure 4, only the inner side is reflective). 103 sample points / 13 fps.

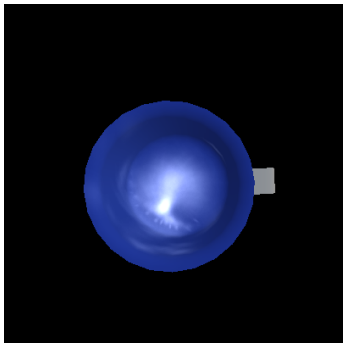


(e) A solid metallic ring, two sided reflections and self reflections. 154 sample points / 6.4 fps.

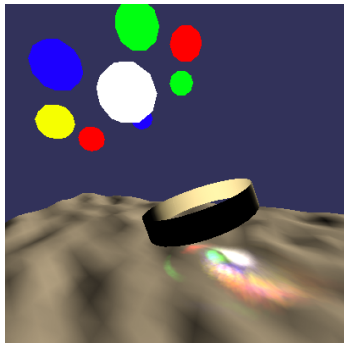
(f) A reflecting cup with caustics from natural light inside (beach environment map taken from [5]). 354 sample points / 4.9 fps.

Figure 6: Caustics from natural light. All scenes where rendered at a resolution of 400×400 Pixel on a Radeon 9700 Pro graphics board.

(a) test scene: a reflective cup (diffuse shading).



(b) Caustic in the cup. Lit from above by a point light source. 354 sample points / 4.9 fps.



(c) Fully dynamic scene. The light source and the reflector are moved interactively. 200 sample points / 9.6 fps.

Figure 7: Some more examples. All scenes where rendered at a resolution of 400×400 Pixel on a Radeon 9700 Pro graphics board.