CGI 2008 Conference Proceedings

Tamás Umenhoffer · Gustavo Patow · László Szirmay-Kalos

Caustic Triangles on the GPU

Abstract This paper proposes a robust algorithm to compute caustics caused by multiple reflections and refractions on the GPU. The proposed algorithm solves two problems of previous methods, caustic light leaks and caustic undersampling. In order to eliminate caustic light leaks, terminal photon hits are stored and caustic patterns are reconstructed on the faces of a layered distance map attached to the caustic generator. To avoid undersampling, we propose caustic triangles to be drawn instead of splatting photon hits. Unlike splatting, caustic triangles adapt to the local density of the uneven photon distribution, always form continuous patterns, do not cause excessive blurring, and do not require manual user intervention to set the size of these splats.

Keywords Caustics · Global Illumination · GPU · Ray tracing · Real-time rendering

1 Introduction

Caustics show up as high frequency patterns on diffuse or glossy surfaces, formed by light paths originating at light sources and visiting mirrors or refracting surfaces. A caustic is the concentration of light, which can "burn". The name caustic, in fact, comes from the Latin "causticus" derived from Greek "kaustikos", which means "burning". These indirect effects have a significant impact on the final image [9,18,20].

Light paths starting at the light sources and visiting specular reflectors and refractors until they arrive at diffuse surfaces need to be simulated to create caustic effects. Theoretically such paths can be built starting the path at the light source and following the direction of the light (light or photon ray tracing), or starting at the receiver surfaces and going opposite to the normal light (visibility ray tracing). If light sources are small, then the probability that visibility ray tracing finds them is negligible, thus visibility ray tracing is inefficient to render general caustics.

In GPU based caustic algorithms, the photon tracing part requires the implementation of some ray tracing algorithm on the GPU. When a ray is traced the complete scene representation needs to be processed. Since the shader processors of GPUs may access just the currently processed varying item (vertex, primitive or fragment), global parameters, and textures, this requirement can be met if the scene geometry is stored in textures.

This paper is organized as follows. In Section 2 we survey the previous work on GPU based caustic effects and in Section 3 we review an existing ray tracing algorithm that we utilized during photon tracing and final rendering. Section 4 discusses the new caustic algorithm. Finally we present the results and the conclusions.

2 Previous work on GPU based caustic effects

General and effective caustic generation algorithms have two phases [1], where the first phase identifies the terminal hits of light paths using some kind of photon ray tracing, and the second projects caustic patterns onto the receiver surfaces.

In the first phase, putting the view plane between the light and the refractor (Figure 3), the scene is rendered from the point of view of the light source, and the terminal hits of caustic paths are determined.

The location of the terminal hits are stored in pixels of the render target called the *photon hit location image*. Note that photon hits can be computed by the fragment shader, or alternatively by the geometry shader of Shader Model 4 GPUs.

From discrete photon hits a continuous caustic pattern needs to be reconstructed and projected onto the caustic receiver surfaces. During reconstruction a photon hit should affect not only a surface point, but also a

Tamás Umenhoffer · László Szirmay-Kalos
TU Budapest, Hungary
E-mail: szirmay@iit.bme.hu

Gustavo Patow
University of Girona, Spain
E-mail: dagush@ima.udg.edu

surface neighborhood where the power of the photon is distributed. A neighborhood consists of points that are in the same direction from the caustic generator object and are all visible from the caustic generator. In order to eliminate light leaks, this neighborhood information should be preserved in the space where photon hits are stored and caustics are reconstructed.

There are several alternatives for spaces to represent the location of a photon hit, having different advantages and disadvantages:

3D grid [15]: Similarly to classical photon mapping photon hits can be stored independently of the surfaces in a regular or adaptive 3D grid. When the irradiance of a point is needed, hits that are close to the point are obtained. If the surface normal is also stored with the photon hits, and only those hits are taken into account which have similar normal vectors as the considered surface point, then light leaks can be minimized and photons arriving at back faces can be ignored. Unfortunately, GPUs are not good in managing adaptive data structures needed by this approach, so such approaches are not effective.

Texture space [5,17,3]: Considering that the reflected radiance caused by a photon hit is the product of the BRDF and the power of the photon, and the BRDF is most conveniently fetched according to the texture coordinates, one straightforward possibility to identify a photon hit is the texture coordinates of that surface point which is hit by the ray. A pixel of the photon hit location image stores the two texture coordinates of the hit position and the luminance of the power of the photon. Since the texture space neighborhood of a point visible from the caustic generator may also include occluded points, light leaks might occur.

Screen or image space [11,24]: A point in the scene can be identified by the pixel coordinates and the depth when rendered from the point of view of the camera. This screen space location can also be written into the photon hit location image. If photon hits are represented in image space, photons can be splat directly onto the image of the diffuse caustic receivers without additional transformations. However, the BRDF of the surface point cannot be easily looked up with this representation, and we should modulate the rendered color with the caustic light, which is only an approximation. This method is also prone to creating light leaks.

Ray space [7,4,10]: Instead of the hit point, the ray after the last specular reflection or refraction can also be stored. When caustic patterns are projected onto the receiver surfaces, the first hit of these rays need to be found to finalize the location of the hit, which is complicated. Thus these methods either ignore visibility [7,4] or do not apply filtering [10].

Shadow map space [16]: In the coordinate system of the shadow map, where the light source is in the origin, a point is identified by the direction in which it is visible from the light source. An advantage of this approach is that rendering from the light's point of view is needed by shadow mapping anyway. The drawbacks are the possibility of light leaks and that caustics coming from the outside of the light's frustum are omitted.

Photon hits form a discrete representation of a continuous phenomena. The distribution of these photon hits are very uneven, since the photon density follows the high frequency caustic patterns. Thus the reconstruction filter and the number of traced photons need to be carefully selected. If the number of traced photons is low, then undersampling effects occur. If this number is high, then the algorithm would be slow.

Most of the GPU based algorithms reconstructed the continuous caustic pattern from discrete hits with photon *splatting* [20,17,16,24,23]. However, photon splatting has drawbacks. The size of splats should be carefully selected in order to preserve high frequency details but to eliminate dot patterns (Figure 1). To find a uniformly good splat size is impossible due to the very uneven distribution of the photon hits. On the other hand, splatting does not take into account the orientation of the receiver surface, which results in unrealistically strong caustics when the surface is lit from grazing angles. The reconstruction problems of splatting can be reduced by adaptive size control [23] or by hierarchical methods [22], but these approaches either always work with the largest splat size or with multiple hierarchical levels, which reduces their rendering speed.

In addition to splatting, another possibility for reconstruction is to take three neighboring hits, assume that they form a caustic triangle or caustic beam, and this beam is intersected with the caustic surfaces. Evolving on an original idea from Nishita and Nakamae [13] for rendering underwater caustics based on the beam-tracing approach, Iwasaki, Dobashi and Nishita [7,8] developed methods for the fast rendering of refractive and reflective caustics due to water surfaces, which were subdivided into triangular meshes. At each water surface mesh vertex, the refracted vector is computed. They call the volume defined by sweeping the reflected or refracted vector at each vertex of a caustic generator triangle the illumination volume. Caustics patterns are displayed by drawing the intersection areas between all of the illumination volumes and the receiver surfaces, and by accumulating the intensities of light reaching the intersection area. Unfortunately, this method treated neither shadows nor the case of warped volumes which can occur in beam tracing Ernst et al. [4] solved some of these problems and also presented a caustic intensity interpolation scheme to reduce aliasing, which resulted in smoother caustics. However, this algorithm also ignored occlusions, so was unable to obtain shadows.

To attack the problems of previous GPU based methods, we propose a new algorithm that

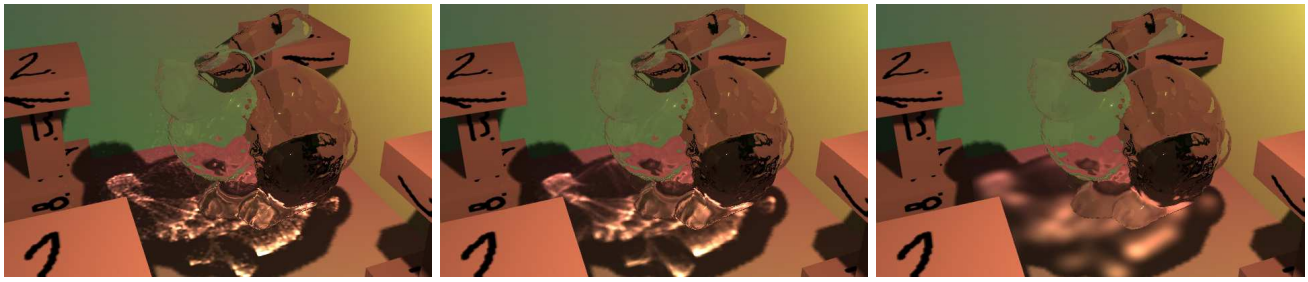


Fig. 1 The problems of splatting. The left image was rendered with too small splatting filter, the right one with too large. To obtain the middle image, the splatting size was manually optimized.

– stores photon hits and reconstructs caustic patterns in the cube map space of the caustic generator and takes into account visibility information when projecting reconstructed caustic patterns onto the surfaces in order to avoid light leaks,
– renders caustic triangles instead of splatting to reduce undersampling artifacts.

3 Ray tracing using layered distance maps

As mentioned, all caustic algorithms should involve a ray tracing method that identifies the terminal hits of the photon paths. In our caustic algorithm we use the method of ray tracing in *layered distance maps* [19]. Representing the geometry as layered distance maps has several advantages, including the seamless integration into rendering engines based on the concept of rasterization and the ease of incorporation of fast visibility algorithms developed for static [2] and dynamic scenes [21].

A single layer of these layered distance maps is a cube map, where a texel contains the material properties, the distance, and the normal vector of the point that is in the texel's direction. The material property is the reflected radiance for diffuse surfaces and a Fresnel factor at perpendicular illumination for specular reflectors. For refractors, the index of refraction is also stored as a material property. The distance is measured from the center of the cube map, which is called the *reference point*.

The computation of distance maps is very similar to that of classical environment maps. The only difference is that not only the color, but the distance and the normal vector are also calculated and stored in additional cube maps. The set of layers representing the scene could be obtained by *depth peeling* [12] in the general case. However, in our case correct depth order is not required between subsequent layers, so layers are organized to maintain the smoothness of the distance values instead of assigning the closest points to the same layer.

The distance map is a sampled representation of the scene geometry, which is searched during ray tracing. When a ray of origin $\mathbf{x}$ and direction $\mathbf{R}$ is traced, ray

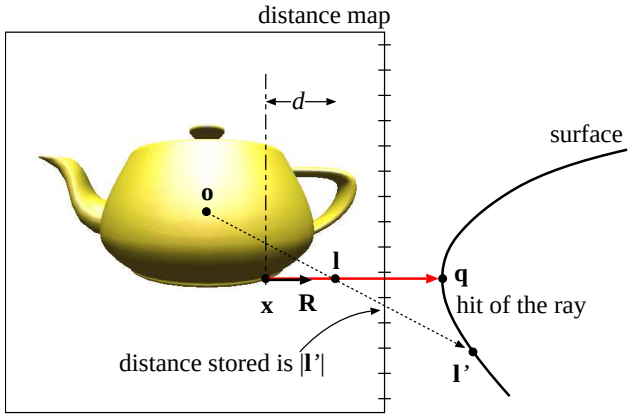


Fig. 2 Tracing a ray from $\mathbf{x}$ at direction $\mathbf{R}$

parameter d should be approximated, which expresses the hit point as $\mathbf{x} + \mathbf{R}d$. The accuracy of an arbitrary approximation d can be checked by reading the distance of the environment surface stored with the direction of $\mathbf{l} = \mathbf{x} + \mathbf{R}d$ in the cube map ($|\mathbf{l}'|$) and comparing it with the distance of approximating point $\mathbf{l}$ on the ray ($|\mathbf{l}|$). If $|\mathbf{l}| \approx |\mathbf{l}'|$, then we have found the intersection. If the point on the ray is in front of the surface, that is $|\mathbf{l}| < |\mathbf{l}'|$, the current approximation is an *undershooting*. On the other hand, the case when point $\mathbf{l}$ is behind the surface ($|\mathbf{l}| > |\mathbf{l}'|$) is called *overshooting*.

Ray parameter d can be found by an iterative process. The process starts with ray marching, i.e. by a linear search, then continues with a secant search. The linear search provides robustness and guarantees that no hit is missed, while the secant search is responsible for accuracy. Note that such two level searches are also used in displacement mapping [14] and in volume ray-casting [6].

An important observation we should make is that this algorithm maintains a cube map centered at the caustic generator object and represents the geometry as distance values stored in cube map texels. In the proposed caustic generation algorithm we exploit this structure also for caustic reconstruction and projection.

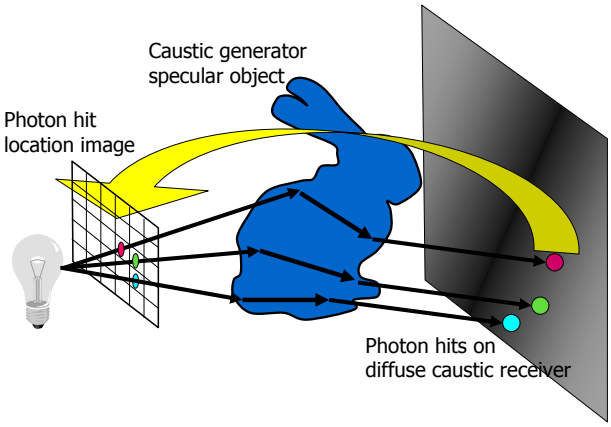


Fig. 3 Caustics generation renders into the photon hit location image where each pixel stores the location of the photon hit with respect to some coordinate system.

4 The new caustics generation algorithm

As concluded during the review of the previous work, a critical issue of all caustic algorithms is the selection of the coordinate system where photon hits are stored and the definition of the algorithm that reconstructs the continuous caustic pattern from these discrete hits.

In order to support artifact free filtering and reconstruction, in this space two points should be close if they are seen in similar directions from the point of view of the caustic generator object. Furthermore, we need a way to separate those points which can be hit by rays leaving the caustic generator objects from those points which cannot be hit due to occlusions.

Let us note that the coordinate system of the distance map used to trace rays leaving the caustic generator meets these requirements. Points that are represented in neighboring texels are seen in similar direction from the center of the cube map, and are potentially affected by neighboring photon paths. On the other hand, comparing the distance of the point from the center of the cube map to the stored distance of the cube map layers, we can decide whether a point is occluded or visible from the center of the caustic generator object.

A point is identified by the direction in which it is visible from the reference point, i.e. the texel of the cube map, and also by the distance from the reference point. An appropriate neighborhood for filtering is defined by those points that are projected onto neighboring texels taking the reference point as the center of projection, and having similar distances from the reference point as stored in the distance map. Note that this approach is very similar to classical shadow mapping and successfully eliminates light leaks.

The caustic generation algorithm has multiple passes. See Figure 4. Firstly, the Photon tracing pass builds the Photon hit Location Image, followed by the Caustic reconstruction pass which reconstructs the caustic patterns on the cube map face of the distance map and, finally, the Camera pass, which projects the caustic patterns onto the caustic receiver surfaces.

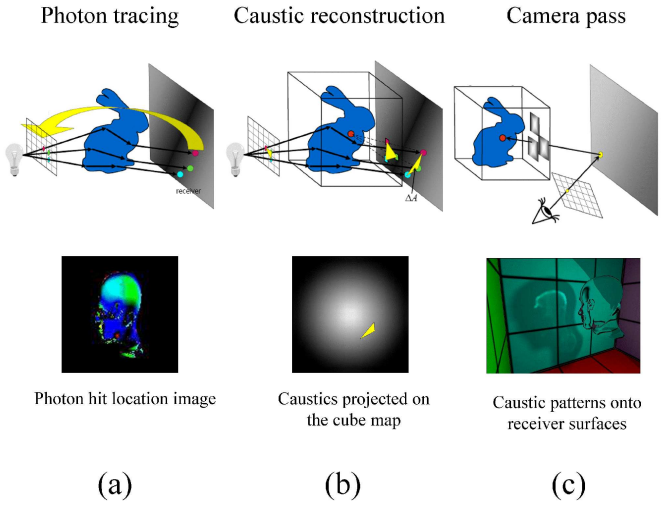


Fig. 4 Overview of the new caustics generation algorithm: Firstly, the Photon tracing pass (a), followed by the Caustic reconstruction pass (b) and finally, the Camera pass(c).

4.1 Photon tracing pass

The caustic generator algorithm first identifies those caustic generators that are visible from the light sources and may cast caustics onto caustic receivers that are visible from the camera. Layered distance maps are generated for the scene. Theoretically, one layered distance map is enough for the whole scene, but due to accuracy reasons we generate a separate distance map for each of these caustic generators if they are far from each other.

Then each caustics generator is rendered from the point of view of each light source (Figure 3). The shaders are set so that the vertex shader transforms the caustic generator to clipping space, and also to the coordinate system of the cube map by first applying the modeling transform, then translating to the reference point. When the fragment shader processes a point on the caustic generator, the light ray defined the light source position and the current pixel is traced, possibly multiple times, until a diffuse or glossy surface is found. The direction of this point with respect to the center of the cube map and the power of the photon are written into the processed fragment. The power is obtained by multiplying the power going through a pixel of the light's camera and the Fresnel factors along the path (or the complement of the Fresnel if refraction happens). To distinguish fragments where the caustic generator is not visible, the render target is initialized with negative power values.

4.2 Caustic reconstruction pass

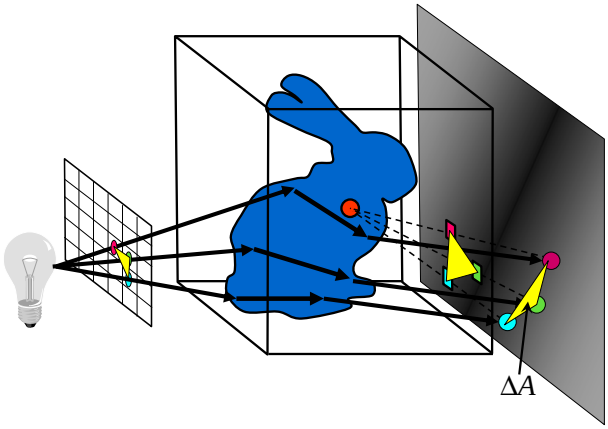


Fig. 5 Computation of the irradiance at the vertices of caustic triangles. The same caustic triangle is shown on the photon hit location image, on the light cube map face, and projected onto the caustic receiver.

We propose caustic patterns to be reconstructed on the cube map face of the distance map, and then projecting the caustic patterns onto the caustic receiver surfaces (Figure 5).

During reconstruction neighboring photon hits are found in adjacent texels of the photon hit location image. These texels contain all information (direction and distance) to determine the vertices of the caustic triangle. A caustic triangle is drawn onto the surface of the cube map, where the vertex locations are set as the directions stored in the photon hit location image. We send two times as many triangles as pixels the photon hit location image has (Figure 6) down the pipeline. We can avoid the expensive GPU to CPU transfer of this image, if vertices are sent with dummy coordinates, or with photon hit location image coordinates, and the vertex shader sets the world or cube map space coordinates according to the content of the photon hit location image. This operation requires at least Shader Model 3 GPUs that allow the vertex shader to access textures. If a pixel of the photon hit location image has negative power, i.e. it is invalid since the caustic generator is not visible in this pixel, then the corresponding vertex is placed outside the clipping region. This way we can exploit the clipping hardware to eliminate invalid triangles and vertices.

The irradiance at a vertex can be computed as the ratio of power $\Delta\Phi$ arriving at neighborhood of area ΔA, i.e. as $I = \Delta\Phi/\Delta A$. Area ΔA is approximated from the areas of projected caustic triangles adjacent to this vertex (Figure 5). Since we use Gouraud shading, i.e. linear interpolation between vertices, power distribution area ΔA is computed as the half of the total areas of triangles attached to this vertex.

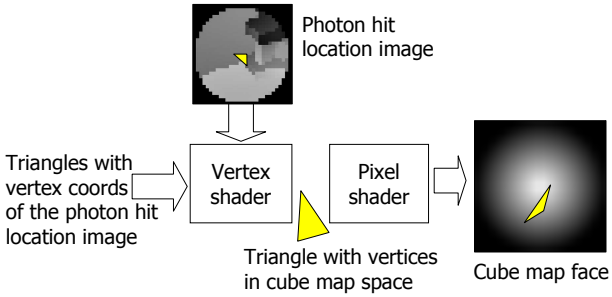


Fig. 6 Rendering to the light cube map.

The compositing operator should be set to "add" since the contributions of different photon hits should be added.

The resulting cube map with irradiance values in its texels is called the *light cube map*, which can be considered as a direction dependent point light source that is responsible for adding the caustic contribution.

4.3 Camera pass

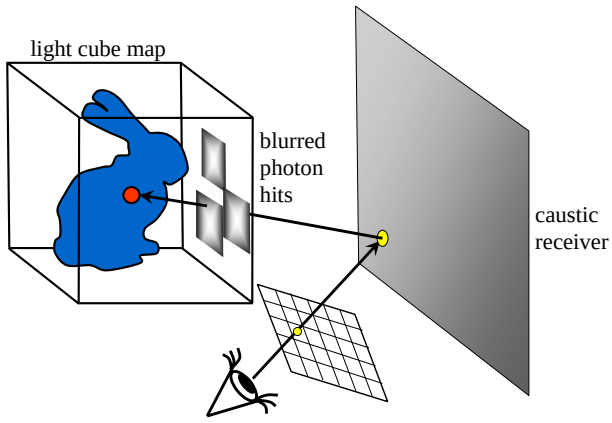


Fig. 7 Light projections in the final camera pass.

During the final camera pass, the light cube map acts as an additional light source (Figure 7). The texels of the cube map already have distance information, thus this cube map is also a shadow map. When a point is processed by the fragment shader during final gathering, the direction between the center of the light cube map and the shaded point is obtained. The light cube map is looked up in this direction. If the distance stored in the light cube map texel is similar to the distance to the shaded point, then the irradiance stored in the texel is multiplied by the BRDF of the shaded point and result is added to the reflection of other light sources. Note that not only the irradiance but also the direction of the

illumination is available during this computation thus glossy BRDFs can also be correctly processed.

5 Optimizations

In the presented method the area computation of a triangle is repeated if a vertex is reused in other triangles. So we can implement a separate pass to render just a single quad at the resolution of the photon hit location image. This pass computes the area of triangles adjacent to a single vertex.

This step can also be utilized to execute hierarchical reconstruction similarly to [22], but unlike in Wyman's approach not the photon hits but caustic triangles are merged together. The hierarchical reconstruction can address the oversampling problem, i.e. that caustic triangles may be very small in the focus (i.e. smaller than a texel of the light cube map or a pixel on the screen). It is worth combining small triangles into a single triangle adding up their power, using the geometry shader of Shader Model 4 GPUs. We consider the implementation of this mipmap based approach as an important future work.

Another important area of further study is the representation of high frequency caustics into the process: now, represented caustics are limited by the resolution of both the Photon Hit Location Image and the Cubemap surface. Although the usage of triangles built from the Photon Hit Location Image generates patterns without noticeable artifacts, small details could be lost. A possible workaround this problem is to employ an adaptive or hierarchical solution, as mentioned before. In our experiments, we have not seen any serious problem with respect to the dependency on the cubemap resolution (we used maps of $6 \times 512 \times 512$, see below).

6 Results

The discussed algorithm has been implemented in DirectX/HLSL environment and integrated into the Ogre3D game engine. Figure 9 shows the photon hit location image, the projected photon hits visualized by point primitives, and the result with caustic triangles. We used a 128×128 pixel resolution photon hit location image, and a Cubemap of $6 \times 512 \times 512$. Figure 8 shows a glass head in the "laboratory" scene. Since the caustic cube map can be looked up not only when the primary ray hits a surface but also when secondary reflective or refractive rays are traced, the proposed method can produce the reflections or refractions of caustics as well. Note that even with shadow, reflection, and refraction computation, the method runs at 60 FPS on an NV6800GT GPU.

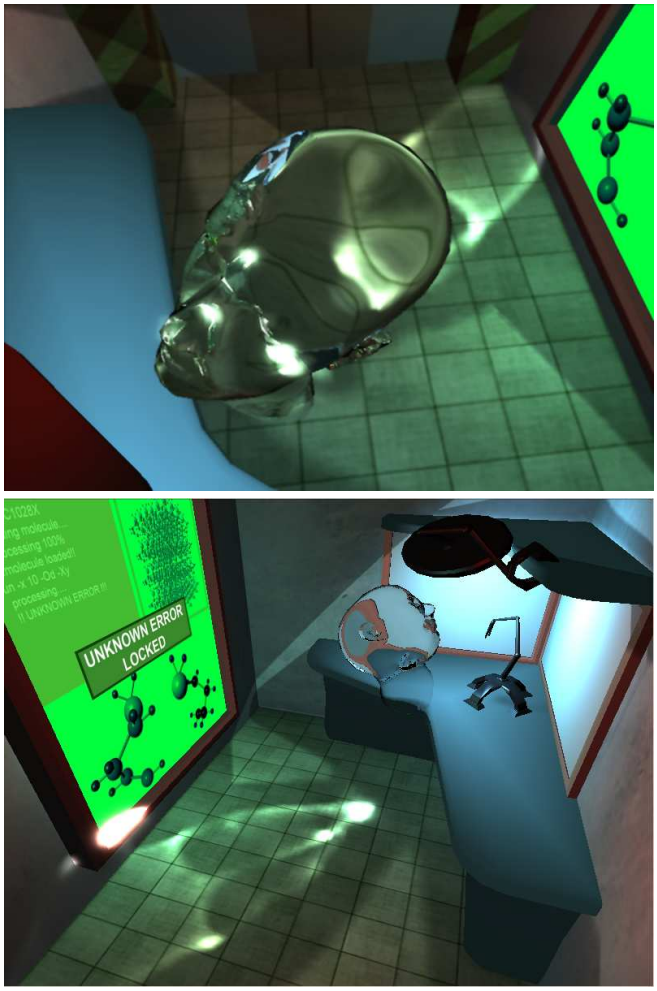
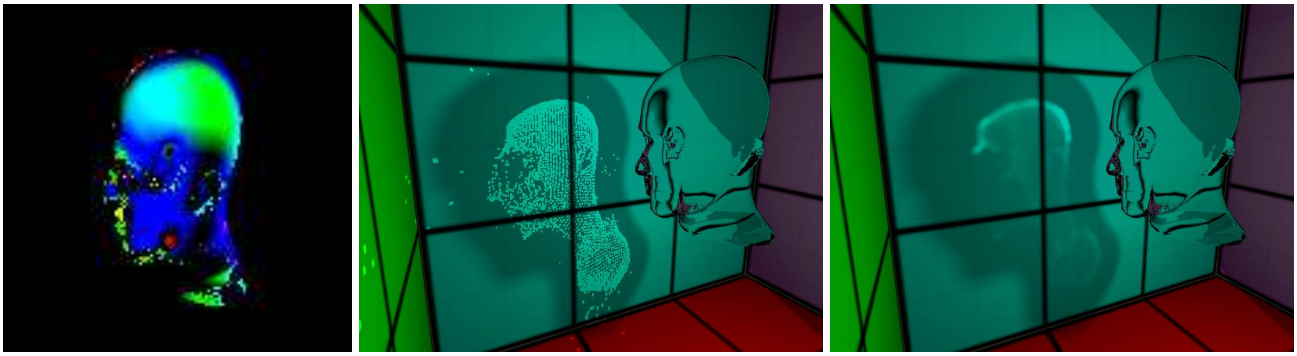


Fig. 8 Real-time caustics caused by a glass head and rendered by the proposed method rendered at 60 FPS on an NV6800GT GPU.

7 Conclusions

We proposed a robust algorithm to compute caustics by tracing rays in scenes represented by layered distance maps. The ray tracing algorithm is used in a multi-pass method where one pass traces light rays and the second eye rays. Unlike splatting-based methods our caustic algorithm does not exhibit problems depending on a splatting filter since caustic triangles provide continuous patterns without overblurring even at moderate resolutions. Also, unlike [4], the proposed approach can correctly take into account occlusions and warped volumes, which allows accurate shadow computations. On the other hand, the method presented here inherits some of the limitations of distance map based ray tracing: the step size of the linear search and the number of iterations of the secant search should be carefully chosen to guarantee that artifacts in the ray tracing stage are minimized.



(a) (b) (c)

Fig. 9 (a) The 128 × 128 resolution photon hit location image, (b) photon hits rendered as points, and (c) the caustics as a collection of transparent triangles rendered at 80 FPS on an NV6800GT GPU.

8 Acknowledgement

This work has been supported by the National Office for Research and Technology, by OTKA K-719922 (Hungary), by TIN2007-67120 from the Spanish Government and by the GameTools FP6 (IST-2-004363) project.

References

1. Arvo, J.: Backward ray tracing. In: SIGGRAPH '86 Developments in Ray Tracing (1986)
2. Bittner, J., Wonka, P., Wimmer, M.: Visibility preprocessing for urban scenes using line space subdivision. In: Pacific Graphics, pp. 276–284 (2001)
3. Czuczor, S., Szirmay-Kalos, L., Szécsi, L., Neumann, L.: Photon map gathering on the GPU. In: Eurographics short papers, pp. 117–120 (2005)
4. Ernst, M., Akenine-Moller, T., Wann Jensen, H.: Interactive rendering of caustics using interpolated warped volumes. In: Proceedings Graphics Interface, pp. 87–96 (2005)
5. Granier, X., Drettakis, G.: Incremental updates for rapid glossy global illumination. Computer Graphics Forum (Eurographics '01) **20**(3), 268–277 (2001)
6. Hadwiger, M., Sigg, C., Scharsach, H., Bühler, K., Gross, M.: Real-time ray-casting and advanced shading of discrete isosurfaces. Computer Graphics Forum (Eurographics '05) **22**(3), 303–312 (2005)
7. Iwasaki, K., Dobashi, Y., Nishita, T.: An efficient method for rendering underwater optical effects using graphics hardware. Computer Graphics Forum **21**(4), 701–711 (2002)
8. Iwasaki, K., Dobashi, Y., Nishita, T.: A fast rendering method for refractive and reflective caustics due to water surfaces. Computer Graphics Forum (Eurographics '03) **22**(3), 601–609 (2003)
9. Jensen, H.W.: Realistic Image Synthesis Using Photon Mapping. AK Peters (2001)
10. Krüger, J., Bürger, K., Westermann, R.: Interactive screen-space accurate photon tracing on GPUs. In: Eurographics Symposium on Rendering, pp. 319–329 (2006)
11. Larsen, B.D., Christensen, N.: Simulating photon mapping for real-time applications. In: Eurographics Symposium on Rendering, pp. 123–131 (2004)
12. Li, B., Wei, L.Y., Xu, Y.Q.: Multi-layer depth peeling via fragment sort. Tech. Rep. MSR-TR-2006-81, Microsoft Research (2006)
13. Nishita, T., Nakamae, E.: Method of displaying optical effects within water using accumulation buffer. In: SIGGRAPH'94 Proceedings (1994)
14. Policarpo, F., Oliveira, M.M., Comba, J.: Real-time relief mapping on arbitrary polygonal surfaces. In: ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games, pp. 155–162 (2005)
15. Purcell, T., Donner, T., Cammarano, M., Jensen, H.W., Hanrahan, P.: Photon mapping on programmable graphics hardware. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp. 41–50 (2003)
16. Shah, M., Pattanaik, S.: Caustics mapping: An image-space technique for real-time caustics. IEEE Transactions on Visualization and Computer Graphics (TVCG) (2006)
17. Szirmay-Kalos, L., Aszódi, B., Lazányi, I., Premecz, M.: Approximate ray-tracing on the GPU with distance impostors. Computer Graphics Forum (Eurographics '05) **24**(3), 695–704 (2005)
18. Trendall, C., Stewart, A.: General calculations using graphics hardware, with application to interactive caustics. In: Rendering Techniques 2000, pp. 287–298 (2000)
19. Umenhoffer, T., Patow, G., Szirmay-Kalos, L.: Robust multiple specular reflections and refractions. In: H. Nguyen (ed.) GPU Gems 3, pp. 387–407. Addison-Wesley (2007)
20. Wand, M., Strasser, W.: Real-time caustics. Computer Graphics Forum (Eurographics '03) **22**(3), 611–620 (2003)
21. Wimmer, M., Bittner, J.: Hardware occlusion queries made useful. In: GPU Gems 2, pp. 91–108. Addison-Wesley (2005)
22. Wyman, C.: Hierarchical caustic maps. Tech. Rep. Technical Report UICS-07-04, University of Utah (2007)
23. Wyman, C., Dachsbacher, C.: Improving image-space caustics via variable-sized splatting. Tech. Rep. Technical Report UICS-06-02, University of Utah (2006)
24. Wyman, C., Davis, S.: Interactive image-space techniques for approximating caustics. In: Proceedings of ACM Symposium on Interactive 3D Graphics and Games (2006)