Caustics Mapping: An Image-space Technique for Real-time Caustics

Musawir Shah, Sumanta Pattanaik

***Abstract*— In this paper, we present a simple and practical technique for real-time rendering of caustics from reflective and refractive objects. Our algorithm, conceptually similar to shadow mapping, consists of two main parts: creation of a caustic map texture, and utilization of the map to render caustics onto non-shiny surfaces. Our approach avoids performing any expensive geometric tests, such as ray-object intersection, and involves no pre-computation; both of which are common features in previous work. The algorithm is well suited for the standard rasterization pipeline and runs entirely on the graphics hardware.**

I. Introduction

CAUSTICS are complex patterns of shimmering light that can be seen on surfaces in presence of reflective or refractive objects, for example those formed on the floor of a swimming pool in sunlight. Caustics occur when light rays from a source, such as the sun, get refracted, or reflected, and converge at a single point on a non-shiny surface. This creates the non-uniform distribution of bright and dark areas. Caustics are a highly desirable physical phenomenon in computer graphics due to their immersive visual appeal. Some very attractive results have been produced using off-line high quality rendering systems; however, real-time caustics remain open to more practical solutions. Deviating from the conventional geometry-space paradigm, which involves path tracing in a 3D scene, intersection testing, etc, we explore an image-space approach to real-time rendering of caustics. Our algorithm has the simplistic nature of shadow mapping, yet produces impressive results comparable to those created using off-line rendering. We support fully dynamic geometry, lighting, and viewing direction since there is no pre-computation involved. Furthermore, our technique does not pose any restrictions on rendering of other phenomena, such as shadows, which is the case in some previous work [3]. Our algorithm runs entirely on the graphics hardware with no computation performed on the CPU. This is an important criterion in certain applications, such as games, in which the CPU is already extensively scheduled for various tasks other than graphics.

{mali,sumant}@cs.ucf.edu, School of Computer Science, University of Central Florida

Fig. 1. Image rendered using the caustics mapping algorithm. This result was obtained using double surface refraction (both for the appearance of the bunny as well as for the caustics) at the rate of 31 fps.

The remainder of this paper is organized as follows: a short survey of related work is presented in Section 2. Our rendering algorithm is then explained in Section 3, followed by Section 4 discussing results and limitations. We conclude with a summary of the ideas presented in the paper and provide directions for future research in Section 5.

II. Previous Work

Although caustics rendering, in general, has been subjected to a fair amount of research, a practical real-time caustics rendering does not exist for everyday applications. In this section, we look at some of the earlier work in offline caustics rendering and recent attempts to achieve caustics at interactive frame-rates.

Fig. 2. Photograph of caustics from a spherical glass paper weight using a desk lamp to emulate directional spotlighting. The caustics are formed on rough paper placed underneath the refractive object.

For a fair amount of computational cost, accurate and extremely beautiful caustics can be produced. Introductory work using backward ray-tracing was proposed by Arvo [1], which was then pursued and extended by a number of researchers. In this method, light rays are traced backwards from the light source into the scene as opposed to conventional ray tracing in which the rays emerge from the eye. Photon mapping, a more flexible framework, was proposed by Jensen [9] which handles caustics in a natural manner on arbitrary geometry and can also support volumetric caustics in participating media [10]. Variants and optimized versions of path tracing algorithms have been presented which utilize CPU clusters [4] and graphics hardware [12]; but the computational cost in time and resources are limiting factors in practical application of these techniques to real-time systems. Wyman et al. [18] rendered caustics at interactive frame-rates using a large shared-memory machine by pre-computing local irradiance in a scene and then sampling caustic information to render nearby surfaces. Such pre-computation steps in algorithms restrict their functionality to domains for which the pre-computation was performed and are unable to support fully dynamic scenes. Our algorithm is also based on the backward ray-tracing idea, however it does not require any pre-computation.

Wand and Straβer [15] developed an interactive caustics rendering technique by explicitly sampling points on the caustics-forming object. The receiver geometry is rendered by considering the caustic intensity contribution from each of the sample points. The authors presented results using specular caustics-forming objects, but refractive caustics can also be achieved with this technique. However, the explicit sampling hinders the scalability of the algorithm since the amount of computation done is directly proportional to the number of sample points used in rendering caustics.

Perhaps the most prominent caustics are those formed in the presence of water. Therefore, the problem of rendering underwater caustics specifically has received significant attention. In early work, Stam [13] pre-computed underwater caustics textures and mapped them onto objects in the scene. Although this technique is extremely fast, the caustics produced are not correct given the shape of the water surface and the receiver geometry. Trendall and Stewart [14] showed refractive caustics to demonstrate the use of graphics hardware for performing general purpose computations. Their intent was to perform numerical integration which they used to calculate caustic intensities on a flat receiver surface. Their method cannot support arbitrary receiver geometry and also cannot be easily extended to handle shadows.

Beam tracing has been employed to produce more physically accurate underwater optical effects, caustics in particular [5], [16]. In this technique, a light beam through a polygon of the water surface mesh is traced to the surface of a receiver object, hence projecting the polygon onto the receiver. The energy incident on the water surface polygon is used to compute the caustic intensity at the receiver, taking into account the areas of the surface and projected polygons. The intensity contributions from all the participating polygons are accumulated for the final rendering.

Nishita and Nakamae [11] present a model based on beam-tracing for rendering underwater caustics including volumetric caustics. Their idea was implemented on graphics hardware by Iwasaki et al. in [7]. In a more recent publication Iwasaki et al. [8] adopt a volume rendering technique in which a volume texture is constructed for receiver objects using a number of image slices containing the projected caustic beams. The case of warped volumes which can occur in beam tracing has not been addressed in either of the above. Ernst et al. [3] manage this scenario and also present a caustic intensity interpolation scheme to reduce aliasing resulting in smoother caustics. However, their algorithm is unable to obtain shadows since it does not account for visibility. In contrast, our algorithm is able to handle shadows and in general does not impose any restrictions on rendering other phenomenon.

III. Rendering Caustics

Our rendering algorithm consists of two main phases: (i) construction of a caustic map texture, and (ii) application of the caustics map to diffuse surfaces called receivers. We will first give a brief explanation of how caustics are formed, and then relate it to our method of construction of the caustic-map.

A. Caustic formation

Caustics are formed when multiple rays of light converge at a single point. This occurs in the presence of refractive or reflective objects which cause the light rays to deviate from their initial path of propagation and converge at a common region. Therefore, to obtain caustics accurately, one must trace light rays from their source and follow their paths through refractive and off reflective surfaces. The photons eventually get deposited on nearby diffuse surfaces, called receivers, thus forming caustics as seen Fig. 3. Our algorithm closely emulates this physical behavior, and is capable of obtaining both refractive and reflective caustics. For reflective caustics we support a single specular bounce. In the case of refractive caustics, in addition to single surface refraction, our method also supports double surface refraction employing the image-space technique recently proposed by Wyman [17]. The algorithm supports point and directional lighting. Area lights can be accommodated by sampling a number of point lights.

B. Caustics Mapping Algorithm

Without loss of generality, we will describe rendering of caustics through refractive objects using our algorithm. A general overview of the algorithm, which closely resembles shadow mapping, is presented next along with elaborative discussion on certain parts.

Following is a stepwise breakdown of the main caustics mapping algorithm.

- **Obtain 3D positions of the receiver geometry:** The receiver geometry is rendered to a *positions texture* from the light's view. 3D world coordinates are outputted for each pixel instead of color. This positions texture is used for ray-intersection estimation in the next step.
- **Create caustic-map texture:** The caustic-map texture is created by splatting points onto the receiver geometry from each vertex of the refractive object along the refracted light direction. The intersection point of the refracted ray and the receiver geometry is estimated using the positions texture. This step is explained in detail in the following section.
- **Construct shadow map:** Although optional, conventional shadow mapping can easily be integrated into the caustics mapping algorithm to render images with both caustics and shadows.

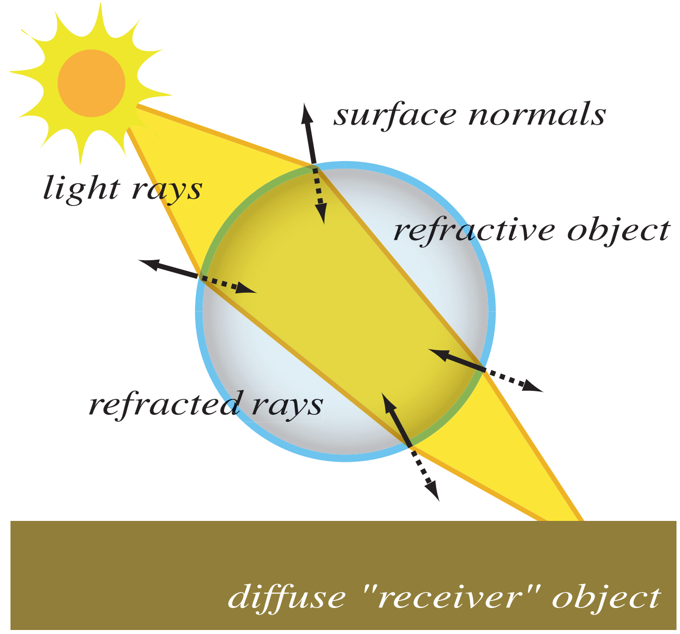

Fig. 3. Diagram showing how multiple light rays can refract through an object and converge at the same point on a diffuse surface.

- **Render final scene with caustics:** The 3D scene is rendered to the frame buffer from the camera's view. Each pixel of the receiver surface is projected into the light's view to compute texture coordinates for indexing the caustic-map texture. The caustic color from the texture is assigned to the pixel and augmented with diffuse shading and shadowing.

The steps mentioned above are performed every frame in separate render passes, and thus impose no constraints on the dynamics of the scene. Furthermore, no computation is performed on the CPU; neither as a per-frame operation nor as a pre-computation.

C. Creating the Caustic-Map

Rendering of the caustics map texture lies at the heart of our algorithm. It consists of three main steps:

- Refraction of light at each vertex
- Estimation of the intersection point of the refracted ray with the receiver geometry
- Estimation of caustic intensity at the intersection point

The caustic-map texture is created by rendering the caustics-forming object from the light's view. The vertices are displaced along the refracted light direction and then splatted onto the receiver geometry by rendering point primitives instead of triangles. The algorithm is implemented in a vertex shader program and proceeds as follows:

for each *vertex v* **do**
T = *Refract(LightDirectionVector)*
P = *EstimateIntersection(v.position + normalize(T), ReceiverGeometry)*
v.position = *P*
return *v*

Notice that multiple vertices can end up at the same position on the receiver geometry. In the pixel shader, the light intensity contribution from each vertex is accumulated using additive alpha blending. Details regarding the intensity and final caustic color computation are given in Sub-section 3 D.

The intersection point of the refracted ray with the receiver geometry must be computed in order to output the final position of the vertex being processed. Normally, this requires expensive ray-geometry intersection testing which is not feasible for real-time applications. We present a novel image-space algorithm for estimating the intersection point which, to our knowledge, has not been used before.

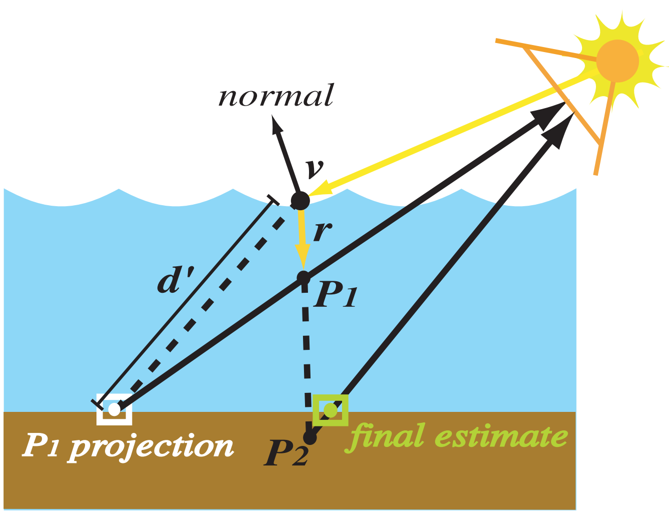

Fig. 4. Diagram of the intersection estimation algorithm. The solid-lined arrows correspond to the position texture lookups.

The intersection estimation algorithm utilizes the positions texture rendered in the first pass of the main caustics mapping algorithm. A schematic illustration of the procedure is shown in figure 4. Let *v* be the position of the current vertex and $\overrightarrow{r}$ the normalized refracted light vector. Points along the refracted ray are thus defined as:

$$P = v + d * \overrightarrow{r} \quad (1)$$

where *d* is the distance from the vertex *v*. Estimating the point of intersection amounts to estimating the value of *d*, the distance between *v* and the receiver geometry along $\overrightarrow{r}$. An initial value of 1 is assigned to *d* and a new position, P_1, is computed:

$$P_1 = v + 1 * \overrightarrow{r} \quad (2)$$

P_1 is then projected into the light's view space and used to look up the positions texture. The distance, *d'*, between *v* and the looked up position is used as an estimate value for *d* in Equation 1 to obtain a new point, P_2. Finally, P_2 is projected in to the light's view space and the positions texture is looked up once more to obtain the estimated intersection point.

The intersection estimation algorithm is essentially an iterative process of which a single iteration has been discussed above. For example, in the next iteration the distance between the estimated intersection point in the last iteration and *v* would be plugged into Equation 1 as a new estimate value for *d*. We have found empirically that the estimate of the intersection point improves with each iteration as it tends to converge at the true intersection point. The magnitude of the error and the number of iterations to convergence depends on the topology distribution of the scene. For a fairly uniform topology, the error is negligible after only 5 iterations. For the purpose of rendering caustics, we observed that only a single iteration is sufficient. Furthermore, the small error in the estimation is well sustained by our caustics mapping algorithm and thus is suitable for the purpose.

D. *Caustic Intensities*

The intensity of the caustics formed on the diffuse receiver geometry depends on the amount of light that gets accumulated at any particular point on the receiver's surface. Since the caustics map texture is created by refracting or reflecting the light rays at each vertex, the intensity of the caustics will depend on the number of vertices which make up the caustic-forming object. If the algorithm is employed using a naive accumulation scheme, the caustics will appear to be bright or dark depending on the number of vertices of the object. This undesirable phenomenon occurs due to the incorrect assumption that the light flux contribution for each vertex is the same. We combat this problem by computing a weighting coefficient for each vertex which is then multiplied with the incident light.

In order to compute the coefficient for a vertex, the total flux contribution for that vertex must be determined by observing the flux through the surrounding triangles in the mesh of the caustic-forming object. For physically accurate computation, the ratio of the projected area of the triangles to the entire projected surface area of the object must be calculated. The flux through an individual triangle can then be distributed equally amongst its constituent vertices. Therefore, the total flux contribution for each vertex in the mesh is computed as:

$$\Phi_i = \sum_{j \in S_i} \frac{PA(\Delta_j)}{3} \frac{1}{V} \tag{3}$$

where V is the total projected surface area of the object, $PA(\Delta_j)$ is the projected area of triangle j, and S_i is the set of triangles that fan around vertex v_i.

Although Φ_i is the correct weighting coefficient for the intensity of the caustics formed through v_i, calculating Φ_i is not feasible for real-time applications due to the expensive projected area computation. In order to maintain the practical nature of our algorithm, we approximate the flux computation as:

$$\widetilde{\Phi_i} = \sum_{j \in S_i} \frac{Area(\Delta_j)}{3} \frac{1}{\widetilde{V}} \tag{4}$$

$$\widetilde{V} = \frac{1}{2} \sum_{\forall k} Area(\Delta_k) \tag{5}$$

where $Area(\Delta_k)$ is the actual area of the triangle k. $\widetilde{V}$ is an approximation of V, and is computed as half of the total surface area. Note that for non-closed 3D meshes, such as a water surface mesh, the total surface area is used for V. With this simplification, the coefficient $\widetilde{\Phi_i}$ can be computed and stored as a vertex attribute in the mesh data structure. Since the coefficients for the vertices are fixed, we consider them to be part of the 3D mesh data structure and not a pre-computation step in our algorithm. For animated meshes, the total surface area and the area of the individual triangles are likely to change during run-time. In such a case, the initial key-frame configuration of the mesh is used to compute the coefficients. This approximation is capable of handling subtle deformations, such as those used in animating water.

The final color of the caustics formed through a refractive object is computed by taking into account the absorption coefficient of the object's material. This enables us to achieve colored caustics.

$$I = I_o e^{-K_a d} \tag{6}$$

where I_o is the incident light intensity, K_a is the absorption coefficient, and d is the distance that light travels through the refractive object. This distance is easily determined by rasterizing the back-faces of the object in an initial render pass to obtain positions of the hind points. In fact, this data is already available if double surface refraction is used [17].

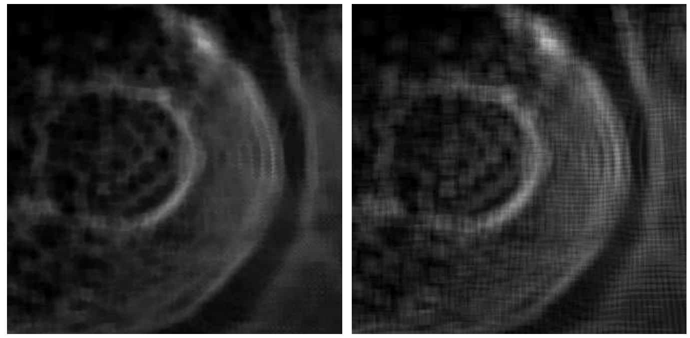

Fig. 5. Caustic-map of a water ripple. Comparison between filtered (left) and unfiltered (right) caustic-map texture is shown.

E. Implementation Issues

The caustics mapping algorithm suffers from issues similar to shadow mapping and other image-space techniques. There are two main points of concern which must be addressed: aliasing, and view frustum limitation. The former issue is inherent in all image-space algorithms. This problem is further magnified due to the usage of point primitives rather than triangles for rendering of the caustic-map. The gaps between the point splats give a non-continous appearance to the caustics. However, if the 3D mesh of the caustics-forming object is tessellated enough, the gaps are significantly reduced. Low pass filtering of the caustic-map texture further smoothes the caustics and improves their appearance.

The second issue pertaining to the algorithm is the view frustum limitation during rasterization of the caustic-map. This problem is exactly like that of shadow mapping with point lights. In such a case an environment shadow map is created, usually with six 2D textures for each face of a cube map rather than a single texture. Similarly, in the caustics mapping algorithm if the caustics are formed outside the light's view frustum, they will not be captured on the caustic-map texture. This happens more frequently with reflective caustics

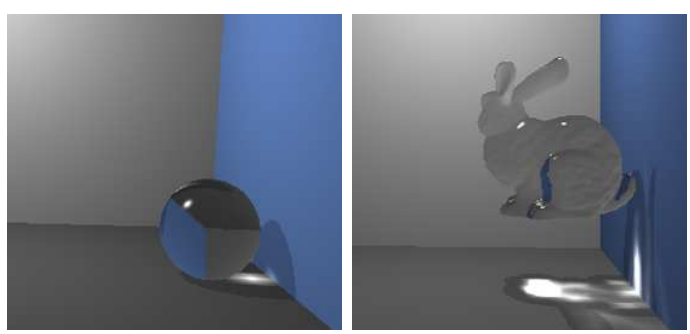

Fig. 6. Caustics rendered using our algorithm for simple objects such us spheres (113fps) as well as complex ones such as the Stanford bunny (101fps)

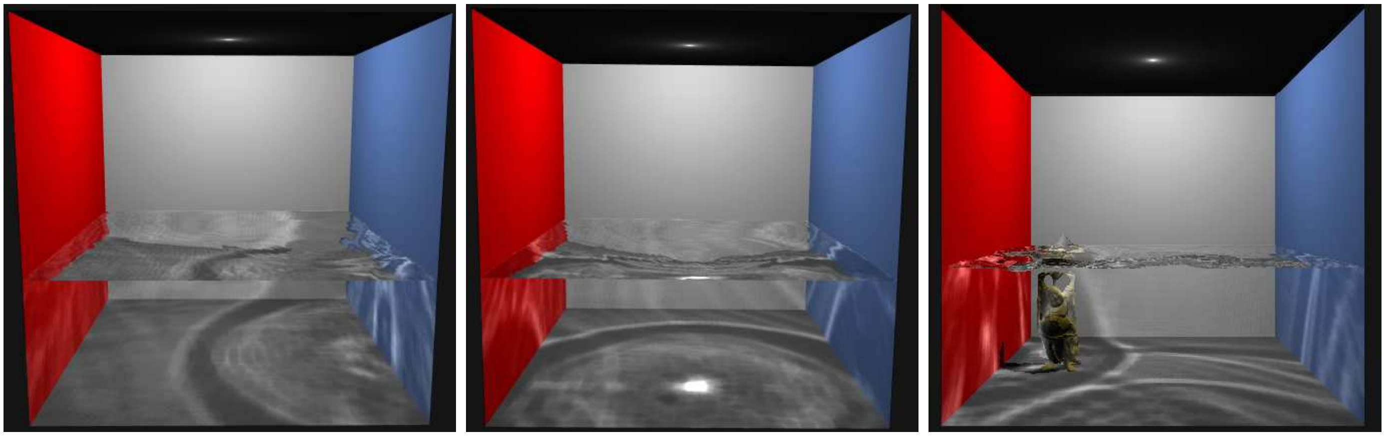

Fig. 7. Frames from a water animation demo with caustics mapping. Random ”plops” are introduced in the water surface mesh at regular time intervals. Notice the caustics pattern and the corresponding shape of the water surface. The last image shows caustics on the Happy Buddha to demonstrate that non-planar receiver geometry is also supported.

than refractive ones. Using an environment caustic map solves this problem at an overhead cost of rendering extra textures. In our implementation we employed cube maps, however the dual paraboloid mapping technique proposed by Heidrich [6], which has been applied to shadow mapping for omnidirectional light sources by Brabec et al. [2], can also be utilized.

IV. Results and Limitations

The caustics mapping algorithm was developed and implemented on a 2.4GHz Intel Xeon PC with 512MB of physical memory. However, since the algorithm performs no computation on the CPU, a lower-end processor will be sufficient. We employed a GeForce 6800 graphics card and pixel shader model 3.0 because the algorithm requires texture access in the vertex shader for intersection estimation in the caustic-map generation step. The algorithm was implemented using Microsoft DirectX 9.0 SDK.

The major advantage of our algorithm is the speed at which it renders the caustics, making it very practical for utilization in games and other real-time applications. We conducted a number of tests and produced results to demonstrate the feature-set of our algorithm. Caustics from both refractive and reflective objects were generated. For refractive objects, two main categories were established: single-surface refraction, and double-surface refraction. Single-surface refraction is suitable for underwater caustics since there is only a single refraction event as light enters through the water surface and hits the floor. The result of our underwater caustics can be seen in Figure 7. It was rendered at a resolution of 640x480 at the rate of 60 frames per second. The mesh used for the water surface consisted of 100x100 vertices.

For solid refractive objects, using double-surface refraction is more physically correct. Most of the previous work done in real-time caustics rendering is limited to single-surface refraction and cannot be easily extended to include the double-surface interaction. Our algorithm effortlessly accommodates Wyman’s [17] image-space double-surface refraction. We rendered various simple and complex objects and observed the caustics that they produced. The results with caustics from a sphere, and the Stanford bunny are shown in Figure 6. Notice that the frame-rate even with double-surface refraction is quite high. The shadows in these images were obtained using conventional shadow mapping. Since we deal with point lights, a shadow cube map was employed. The resolution of both the shadow map and the caustics map was 768x768x6 pixels for rendering final images of resolution 1024x768 pixels.

Caustics from reflective objects using our method are shown in Figure 8. Our caustics mapping algorithm performs better with refractive caustics than reflective ones since the error in the intersection estimation during the caustic-map generation step tends to be greater in the latter case. This is due to the fact that refraction causes small deviations in the path of the light ray, whereas reflection causes the ray to change its direction completely. We would like to improve upon our intersection estimation algorithm in future work to better handle the reflection case.

Another limitation of our algorithm is that it does not easily extend to volumetric caustics like some techniques proposed in previous work [3]. This is mainly due to the fact that the caustic-mapping algorithm operates in image-space. One way of achieving volumetric caustics using our algorithm is to use a number of planes perpendicular to the light ray as the caustic receivers. However this method spawns further issues relating to sampling and volumetric rendering which need to be addressed.

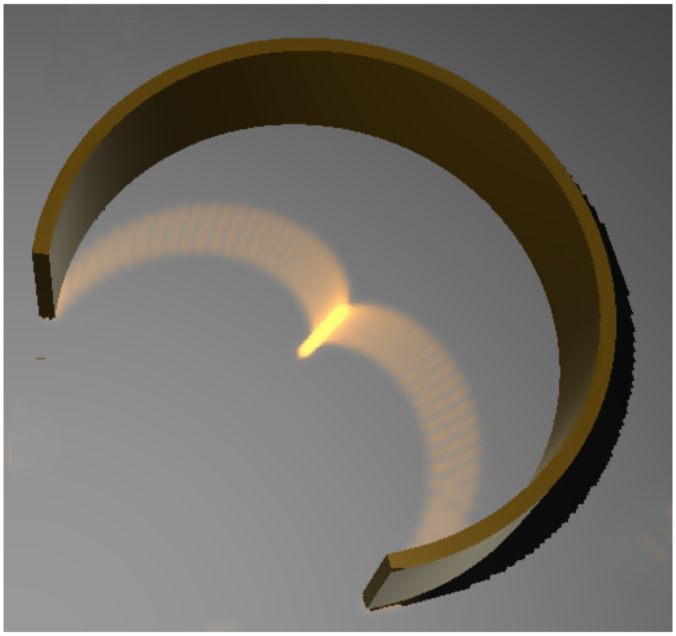

Fig. 8. Reflective caustics from a brass ring. The desired cardioid shape that is observed in real life is obtained from the caustics-mapping algorithm.

V. Conclusion and Future Work

We have presented a practical real-time caustics rendering algorithm that runs entirely on the graphics hardware and requires no pre-computation. It emulates the light transport involved in caustics formation in the image-space; therefore it is both physically inspired and fast. The algorithm is conceptually similar to shadow mapping and integrates easily into virtually any rendering system.

We also presented a fast ray-scene intersection estimation technique which we plan to further explore and improve in future work. This technique is extremely fast and is well suited to algorithms, such as caustics mapping, which can tolerate a certain amount of error in the estimation.

The caustics mapping algorithm presents a number of options worth pursuing for further enhancement. For example, better techniques for filtering of the caustic-map might considerably improve the overall appearance of the caustics. Finally we would like to develop methods of applying the caustics mapping algorithm to participating media and achieve volumetric caustics.

Acknowledgment

The authors would like to thank Guillaume Francois, Kévin Boulanger, and Jaakko Konttinen for their insightful comments, and Eric Risser for assistance with illustrations used in this paper. This work was supported in part by ATI Reseach and Electronic Arts.

References

[1] James Arvo. Backward ray tracing. In *Developments in Ray Tracing, SIGGRAPH '86 Course Notes*, August 1986.

[2] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *Computer Graphics International*, 2002.

[3] Manfred Ernst, Tomas Akenine-Moller, and Henrik Wann Jensen. Interactive rendering of caustics using interpolated warped volumes. In *Graphics Interface*, May 2005.

[4] Johannes Guenther, Ingo Wald, and Philipp Slusallek. Realtime caustics using distributed photon mapping. In *Eurographics Symposium on Rendering*, pages 111–122, 2004.

[5] Paul S. Heckbert and Pat Hanrahan. Beam tracing polygonal objects. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 119–127, New York, NY, USA, 1984. ACM Press.

[6] Wolfgang Heidrich. View-independent environment maps. In *Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '98*, 1998.

[7] Kei Iwasaki, Yoshinori Dobashi, and Tomoyuki Nishita. An efficient method for rendering underwater optical effects using graphics hardware. *Computer Graphics Forum*, 21(4):701–711, 2002.

[8] Kei Iwasaki, Yoshinori Dobashi, and Tomoyuki Nishita. A fast rendering method for refractive and reflective caustics due to water surfaces. In *Eurographics*, pages 283–291, 2003.

[9] Henrik Wann Jensen. Global Illumination Using Photon Maps. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, pages 21–30, New York, NY, 1996. Springer-Verlag/Wien.

[10] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. In *Computer Graphics (SIGGRAPH 98)*, pages 311–320. ACM Press, 1998.

[11] Tomoyuki Nishita and Eihachiro Nakamae. Method of displaying optical effects within water using accumulation buffer. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 373–379, New York, NY, USA, 1994. ACM Press.

[12] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *Graphics Hardware*, pages 41–50. Eurographics Association, 2003.

[13] Jos Stam. Random caustics: natural textures and wave theory revisited. In *SIGGRAPH '96: ACM SIGGRAPH 96 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '96*, page 150, New York, NY, USA, 1996. ACM Press.

[14] C. Trendall and A. Stewart. General calculations using graphics hardware with applications to interactive caustics. In *Eurographics Workshop on Rendering*, pages 287–298, 2000.

[15] M. Wand and W. Strasser. Real-time caustics. *Computer Graphics Forum*, 22(3):611–611, 2003.

[16] Mark Watt. Light-water interaction using backward beam tracing. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 377–385, New York, NY, USA, 1990. ACM Press.

[17] Chris Wyman. An approximate image-space approach for interactive refraction. In *SIGGRAPH 2005*. ACM Press, 2005.

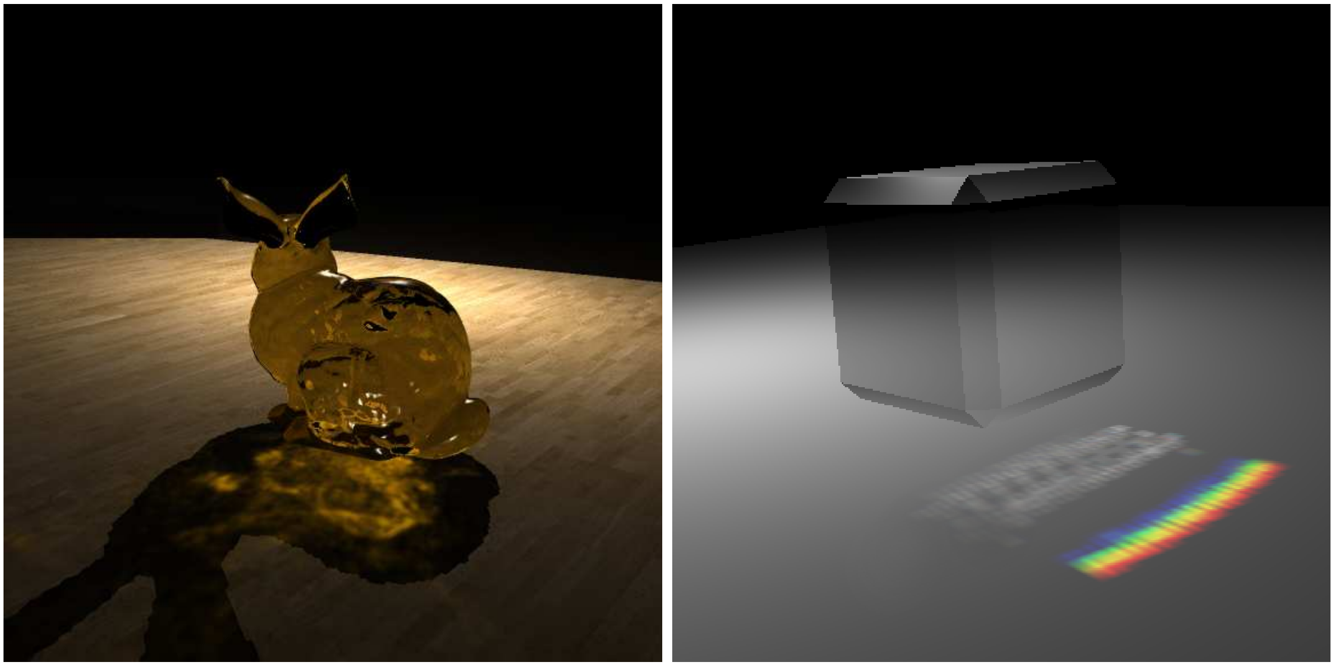

Fig. 9. (Left) Colored caustics from a refractive bunny with light attenuation in different color channels. (Right) Diffraction grating effect obtained using the caustic mapping algorithm with different refractive indices for each of the three RGB color channels.

[18] Chris Wyman, Charles D. Hansen, and Peter Shirley. Interactive caustics using local precomputed irradiance. In *Pacific Conference on Computer Graphics and Applications*, pages 143–151, 2004.