Fifth Hungarian Conference on Computer Graphics and Geometry, Budapest, 2010

Real-Time Volumetric Caustics with Projected Light Beams

Gábor Liktor, Carsten Dachsbacher

VISUS, University of Stuttgart

Abstract

We present a method for the visualization of volumetric caustics in single-scattering participating media. The caustics beams are generated from a projected grid in light's image space, making the solution independent from the geometric complexity of generators. The caustic volumes are extruded in the geometry shader and accumulated into the final volumetric effect.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Raytracing I.3.7 [Computer Graphics]: Color, shading, shadowing and texture

1. Introduction

Specular surfaces tend to change the distribution of reflected or refracted photons causing beautiful light phenomena in their environment. Caustics are high-frequency patterns appearing on the surface of diffuse objects as the curvature of the specular surface focuses or defocuses the light. Rendering caustics adds a lot of realism to the final appearance of the image. A glass of drink casts typical focused light to the table, and it is hard to make an underwater scene believable without the wavy light-patterns on the ground. In this scenario the specular objects are generally called caustics generators and the diffuse ones receivers. Being an important global illumination effect, many efforts have been made in recent computer graphics research for plausible real-time approximation.

While recent work mainly focused on the rendering of surface caustics, the efficient handling of volumetric phenomena requires slightly different considerations. If the specular surface is enclosed by participating media, the rays of focused photons become visible to the viewer. The effect can be experienced under the sea surface, since the sea water always contains microscopic particles attenuating the light. In the case of surface caustics, the main challenge is the adaptive sampling and filtering of the projected pattern to make it smooth and continuous on the screen[8 9]. Volumetric caustics are less sensitive to the accuracy of filtering, but require the integration of the inscattered light along the viewing ray for each pixel.

A common way of rendering volumetric caustics is to subdivide the caustics generator to small patches and extrude them along the directions of the specular photons, forming the bounding volumes of the caustic beams. One possible subdivision is the usage of the mesh geometry itself. By accumulating the contributions of the beams in each pixel, the volumetric photon distribution becomes visible. The appropriate rendering of such volumes was deeply analyzed in the work of Ernst et al.[1], which we used as main basis for our implementation.

Our contributions include the GPU-based implementation of the volumetric caustics algorithm and the application of the projected grid concept to the beam generation. Instead of using the mesh geometry, our method projects a vertex grid to the surface in light's image space and then performs the extrusion of the beams in the geometry shader. This way the complexity of the algorithm becomes independent of the complexity of the specular surface models.

The paper is organized as follows. The next section summarizes recent work on interactive rendering of caustics, and in particular volumetric effects. Section 3 describes our algorithm in detail. Section 4 highlights some important aspects of the implementation and section 5 presents the test results, followed by a discussion.

2. Previous Work

Most of the caustics rendering algorithms can be divided into two main stages. The first phase starts from the light source,

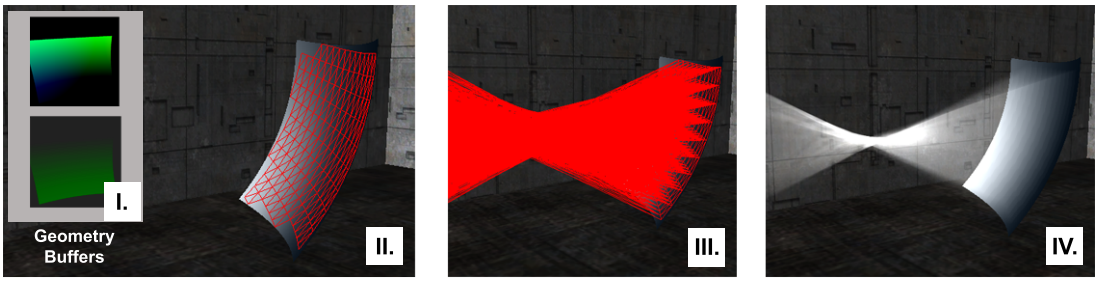

Figure 1: *The outline of the algorithm. In the first pass, the specular surface normals and positions are stored in the geometry buffers (I). A regular grid is projected onto the surface (II), from which the geometry shader extrudes the beam volumes (III). The volumetric caustic effect is reconstructed by ray-marching in the pixel-shader (IV).*

and identifies the terminal points of caustic light paths on the non-shiny surfaces. In the following step, the contributions of these paths are accumulated at the points visible from the camera.

Photorealistic off-line rendering algorithms generally rely on ray-tracing methods, producing extremely accurate caustics for the price of high computational costs. Photon mapping[2] is a popular approach, which stores caustics photon hits in dedicated photon maps. Later, the enhanced algorithm was able to capture volume caustics[3], and Purcell et al.[4] implemented it on the graphics hardware.

By the advent of GPU-based solutions, the interactive rendering of caustics became possible. Caustics mapping can be regarded as a simplification of the photon mapping algorithm. Since the k-neighbor gather operation does not suit to the current GPU architectures, the photon hits are stored in a 2D buffer. Possibilities of representing photon locations include texture space[5], image space[7 9], or the coordinate system of the caustics generators[6].

During the caustics reconstruction, most of the recent works applied photon-splatting. The common problem of splatting algorithms is to find the optimal photon distribution (importance sampling) and splatting size to achieve high quality results. Wyman and Dachsbacher[7] improved the quality by the adaptive variation of splat sizes. Recent work of Wyman et al. analyzed the problem of hierarchical sampling during the caustic map generation[8 9].

Other researchers proposed alternative reconstruction methods. Rendering continuous geometry instead of discrete photon hits can eliminate the discretization artifacts. Ernst et al.[1] applied beam tracing[10] to render interpolated caustic triangles to the receiver surfaces. Their algorithm significantly improved the quality of caustic triangle methods by taking into account the warped distortions of the beams. However, it does not handle occlusions, and requires CPU assistance due to the limitations of graphics hardware at that time. In a recent work, Umenhoffer et al.[6] render caustic triangles on the GPU using layered distance impostors[5] to render smooth caustics with occlusion information.

For volumetric caustics effects, beam tracing was used by Nisthita and Nakamae [11] to render underwater volume caustics. Iwasaki et al.[12] implemented the former algorithm on the GPU. For a moderate amount of caustic triangles, their results had blocky artifacts because of using constant shading. Ernst et al.[1] analyzed the radiance distribution along a warped triangular beam, and were able to decrease the resolution of the generator mesh by using interpolation inside the beams. The caustic interpolation was implemented in the pixel shader during the rasterization of the convex bounding prisms of the warped volumes.

Approximate global illumination algorithms often try to avoid expensive computations at invisible points of the scene. Screen space adaptive subdivision is one of the possible optimization alternatives. In an early application for the rendering of ocean surfaces[13] a regular screen space grid was projected to the heightmap of the water. Recently Müller et al.[14] used this concept to render arbitrary 3D fluids with screen space tessellation. The advantage of the method is that surfaces closer to the viewer get finer geometric details, while invisible surfaces are not tessellated at all. We have applied this idea to the problem of beam tracing in this work.

3. Algorithm

Our method for rendering volume caustics breaks down to two rendering passes (Figure 1). In the first pass the surfaces of the specular objects are rendered from the light source. The photon bounces of the surfaces are stored in a set of render targets. Using these as textures, the second pass generates the geometry of the beams. For each beam the scattered radiance towards the camera needs to be determined using the participating media rendering equation. The final caustics effect is the result of the accumulated rendering of each beam.

Since we do not have control over the rendering order of caustic beams, only single-scattering homogeneous participating media can be simulated. For real-time applications, this simplified model is generally satisfactory.

3.1. Beam Generation

The first step of the algorithm captures the caustics generator surfaces from the light source. Since the beam generation is independent of the mesh topology, surfaces of arbitrary number and complexity can be used as generators in the same pass. The surface positions, normals and material properties are stored in geometry buffers. Usually the choice of a proper coordinate system for these parameters is important for the following steps. As we will see later, most of the computations are performed in a special local space for each individual beam, therefore any initial representation is sufficient. Currently we store the world space positions and normals, but in a caustic shadows solution we can transform those into cube-map space like Umenhoffer et al.[6].

The next step projects a regular grid in light's image space to the specular surfaces. Having the previously rendered geometry buffers, this process is extremely simple. A pregenerated regular grid is sent to the pipeline, but during rendering the position and normal vertex attributes are replaced by the sampled values from the geometry buffers. Prior to rendering the buffers are cleaned to values indicating that the given pixel is invalid. This will allow the geometry shader to discard the primitives not covered by the generator surfaces.

The projected grid is then forwarded to the geometry processing stage. The geometry shader either discards the triangles which are not bases of caustics beams, or emits new primitives forming the convex boundaries of the caustic volumes.

If the vertices of the specular triangle are denoted by $\vec{v}_0$, $\vec{v}_1$, $\vec{v}_2$ and the caustic photon directions at these vertices $\vec{d}_0$, $\vec{d}_1$, $\vec{d}_2$, the geometry shader has to extrude a volume bounding the rays $\vec{v}_i + t_i\vec{d}_i$, $i \in \{0..2\}$. In our current solution t_i is a constant value, since we do not consider occlusions yet. Later on t_i should be determined by a depth-texture lookup.

The extrusion of the beams needs special considerations[1]. In the general case the geometry of the beams cannot be represented by a finite set of planar surfaces, since the adjacent $\vec{d}_i$ rays do not necessarily lie on the same plane, but form the sides of a bilinear patch instead. Figure 2 illustrates artifacts raised by the direct emission of the beams in the geometry shader. From the point of volume caustics, the main artifact is caused by the additive blending, where the same beam might render multiple times to the same pixel, resulting in light streaks. This artifact remains dominant on the screen even at high grid resolutions.

The task of the geometry shader is to emit a triangular mesh bounding the bilinear patches. During the extrusion a

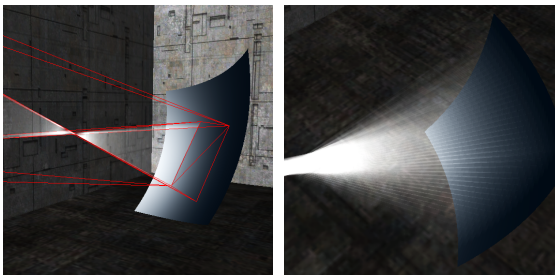

Figure 2: *The rendering artifacts when extruding along the photon directions. The non-planar surfaces of the bilinear patches are approximated with two triangles (left), which leads to light streaks when using additive blending (right). The contrast of the image was slightly increased to emphasize the effect.*

touching plane is found for each edge of the specular triangle. Assuming finite beam length, one can always find a plane that holds the entire caustics volume on its negative half space[1]. The extrusion ray for each vertex can be found as the intersection of touching planes for the adjacent edges.

The volumetric rendering equation is solved in the fragment shader, therefore the geometry processing stage must encode all the necessary parameters into the emitted vertex attributes for performing the sampling of the beams. These parameters are:

- The positions of the caustic vertices
- The directions of the caustic beam edges
- The area of the specular triangle
- The caustic radiance values at the vertices of the triangle

The main task of the fragment shader is to find the radiance at each sample point in the beam. This sampled radiance depends on the initial caustic radiance on the surface and the area ratio of the specular triangle and the parallel cross section of the beam at the sample point. We know that the radiant flux

$$\Phi(\Delta\omega, \Delta_c) = \int_{\Delta_c}\int_{\Delta\omega} L(\vec{p}, \vec{\omega}) \cos\theta \, d\vec{\omega} dA$$

is constant for every cross section of the beam (Δ_c is the caustic triangle, $\vec{p}$ is a representative point in the infinitesimal surface and θ is the angle between the surface normal and $\vec{\omega}$). If the cross section is parallel to the triangle, the radiance at each point can be computed using the area ratio. The question is how to get the area of the cross section of the warped volume?

The double area of the caustic triangle is defined by the length of the cross product of the edges:

$$A(\Delta_c) = |(\vec{v}_1 - \vec{v}_0) \times (\vec{v}_2 - \vec{v}_0)|$$

The calculation of the double area for each sample along the beam would be very expensive in the fragment shader. It is desirable to express the area as a function of distance along the beam. Ernst et al.[1] proposed a special coordinate system, which we will call beam space in the following, where the mentioned cross product simplifies to a one dimensional problem.

Since we are looking for cross sections parallel to the specular triangle, it is beneficial to use a basis where one axis is perpendicular to the triangle. Prior to further calculations, the beam parameters are transformed into a coordinate system where $\vec{v}_0$ is the origin, the x-axis is the $\vec{v}_0 - \vec{v}_1$ edge and the y-axis is the triangle normal. Normalizing these vectors and choosing their cross product as z-axis gives us an orthogonal basis in beam space. Another important simplification is to scale the $\vec{d}_i$ ray direction vectors so that their y-coordinate equals to one in beam space (Figure 3).

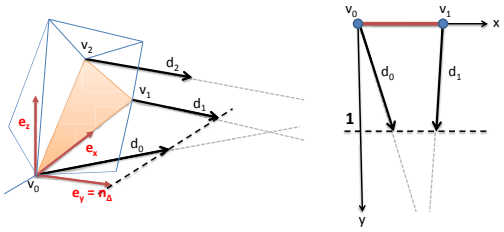

Figure 3: *The basis vectors in beam space. The ray-marching of the beams is performed in this coordinate system.*

Using this representation, we can rewrite the area function into the following form[1]:

$$\begin{aligned} A(y) &= |(\vec{v}'_1 + y\vec{d}'_1 - \vec{v}'_0 - y\vec{d}'_0) \times (\vec{v}'_2 + y\vec{d}'_2 - \vec{v}'_0 - y\vec{d}'_0)| \\ &= |(\vec{k}_1 + y\vec{l}_1) \times (\vec{k}_2 + y\vec{l}_2)|, \\ &\quad \text{where } \vec{k}_i = \vec{v}'_i - \vec{v}'_0 \text{ and } \vec{l}_i = \vec{d}'_i - \vec{d}'_0 \end{aligned}$$

Vector $\vec{v}'$ is $\vec{v}$ in beam space using the new basis. Since $\vec{k}_{iy} = \vec{l}_{iy} = 0 \ \forall i$, the length of the cross product will equal to its y-component.

Expanding the formula we get a second order expression over y in the form of $ay^2 + by + c$, where:

$$\begin{aligned} a &= \vec{l}_{1z}\vec{l}_{1x} - \vec{l}_{1x}\vec{l}_{2z}, \\ b &= \vec{k}_{1z}\vec{l}_{2x} - \vec{k}_{1x}\vec{l}_{2z} + \vec{l}_{1z}\vec{k}_{2x} - \vec{l}_{1x}\vec{k}_{2z}, \\ c &= \vec{k}_{1z}\vec{k}_{2x} - \vec{k}_{1x}\vec{k}_{2z} \end{aligned}$$

The a, b, c area coefficients are computed in the geometry shader and stored in a vertex attribute. The new coordinate system must also be forwarded to the fragment shader, which evaluates the volumetric rendering equation by ray-marching in beam space.

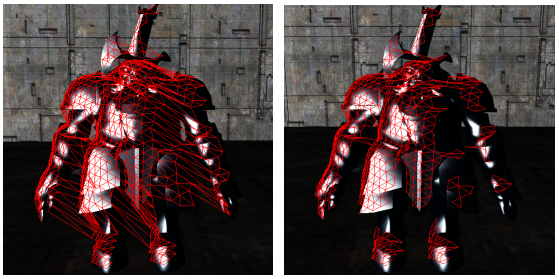

Figure 4: *The direct projection of the regular grid results in false triangles around depth discontinuities. Using simple heuristics based on the triangle normals and the light's direction vectors, these triangles are eliminated.*

The geometry shader must also remove triangles which would result in incorrect caustics, for example by not lying on the same surface. The projected grid is independent of the scene, therefore it is possible that some triangles go through extreme distortions. We use the dot product of the triangle normal $\vec{n}_\Delta$ and the light direction vector $\vec{d}_l$ as heuristics for discarding "wrong" triangles. We assume that if $\vec{n}_\Delta$ is almost parallel to $\vec{d}_l$ then the caustics effect is the most accurate, while if they are "almost" perpendicular to each other, the triangle must be discarded. In figure 4 we visualized the projected grid using the normal heuristics.

3.2. Ray-Marching Caustics Volumes

The fragment shader must first intersect the camera-ray with the beams, then solve the volumetric rendering equation in a set of sample points inside them. For homogeneous participating media this equation takes the following form:

$$L(\vec{\omega}) = \int_{\Delta x} e^{-\sigma_t(s_1+s_2)} \sigma_s P(\vec{\omega} \cdot \vec{\omega}') L_{in}(\vec{\omega}')\, ds$$

where $L(\vec{\omega})$ is the radiance seen from the camera, Δx is the interval of the ray-beam intersections, σ_t is the extinction coefficient, σ_s is the scattering coefficient, P is the phase function, and $s_1 + s_2$ is the travel length of caustic photons from the specular surface to the camera. The notations are visible in Figure 5. Ray-marching is a numerical approximation of the above integral. In the case of volumetric caustics, we have found that it is enough to take a single sample in the middle of the Δx interval. The simplified numeric approximation becomes

$$L(\vec{\omega}) \approx \Delta x e^{-\sigma_t(s_1+s_2)} \sigma_s P(\vec{\omega} \cdot \vec{\omega}') L_{in}(\vec{\omega}')$$

The sampling in the fragment shader is performed in three steps (The whole process is illustrated in figure 5):

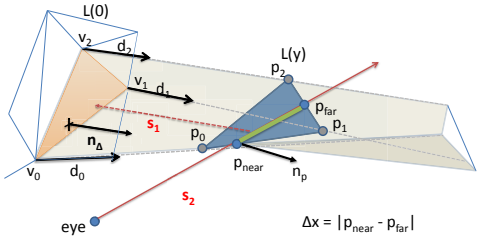

Figure 5: *Intersecting the caustic volume with a ray. The intersection of the beam edges and the ray-plane defines a triangle. Using 2D line-line intersections, the distance which the ray takes inside the volume is determined. The participating media equation is evaluated at a sample point inside this Δx-long interval. To get the caustic radiance values, the area function is evaluated at the corners of the intersection triangle.*

- *Transform the viewing ray into beam space.*

  The basis of the new coordinate system is given as a vertex attribute.

- *Intersect the beam with a plane containing the viewing ray.*

  The normal of such a plane is determined by:

  $$\vec{n_p} = r_{\vec{view}} \times (r_{\vec{view}} \times \vec{n_\Delta})$$

  where $r_{\vec{view}}$ is the viewing vector and $\vec{n_\Delta}$ is the triangle normal (the y basis in beam space). By intersecting the $\vec{d}_{0-2}$ caustic photon directions with this plane, we get a triangle in the same plane as the viewing ray.

- *Intersect the resulting triangle with the ray and trilinearly interpolate the radiance at sample point $\vec{p}$.*

  The radiance values at the corners of the intersection-triangle are determined efficiently by evaluating the area function. Since the area coefficients were precalculated in the geometry shader, the fragment shader only has to evaluate the dot product $(y^2, y, 1) \cdot (a, b, c)^T$.
  Since the triangle and the ray lie in the same plane, the 2D intersection simplifies to finding the intersections with the edges (2D line-line intersection), and interpolating between the caustic radiances at the triangle corners. If the ray does not intersect the triangle, the fragment is discarded.

4. Implementation

We implemented the algorithm in DirectX 10 using HLSL shaders, with our test platform equipped with Nvidia Geforce GTX 260 graphics card, Intel Core i7 920 CPU and 6 GB RAM.

The geometry buffer generation is a straightforward process. Note that instead of storing the data in textures, the geometry could have been rendered directly into the grid vertex buffer. This would avoid the execution of the vertex shader program in the second pass with multiple texture lookups. However, we did not use this solution for flexibility reasons: in the future we plan to implement adaptive hierarchical beam generation in the geometry shader which will make texture lookups necessary anyway.

The second rendering pass which performs the actual volume caustics rendering must be executed once for each caustic effect. For example if we want to visualize both reflective and refractive caustics the projected grid is to be rendered two times. Two sided refractions are not supported yet. The output format of the beam geometry shader is unusually complex, since it must encode the entire beam into one vertex – the fragment shader gets these parameters as interpolated attributes. The beam parameters are packed into 10 *float4* vertex attributes as shown in table 1.

Field Name	Components			
Params0	$v1^b.x$	$v1^b.z$	$v2^b.x$	$v2^b.z$
Params1	$e0^b.x$	$e0^b.z$	$e1^b.x$	$e1^b.z$
Params2	$e2^b.x$	$e2^b.z$	2A	
L0	Radiance of edge 0			
L1	Radiance of edge 1			
L2	Radiance of edge 2			
Basis0	ex			v0.x
Basis1	ey			v0.y
Basis2	ez			v0.z
wpos	World position			
AreaCoeffs	a	b	c	

Table 1: *Vertex attributes.*

In the *Params0-2* attributes the beam geometry is stored in beam space. Note that we do not need to store the coordinates of $\vec{v_0}$ and any y-components since these are zero in this basis. *Params2* also stores the double area of the caustic triangle. L0-2 holds the caustic radiances at the corners of the specular triangle. *Basis0-2* are also very important, since they encode the transformation into beam space. Finally *wpos* is the world position of the vertices (the viewing ray is the difference between the world position and the eye position) and *AreaCoeffs* holds the area coefficients.

5. Results

The test renderings used for performance measurement are shown in figure 6. The results are summarized in table 2. Note that the test scenes contain other shaders not related

to our method, like a naive implementation of volumetric shadows with ray-marching.

Grid res.	Total	Caustics	Without PS	
128x128	92.5	80	11.3	(10.8 FPS)
64x64	45	31.4	9.1	(22.2 FPS)
32x32	24.1	9.7	6.2	(41.4 FPS)

Table 2: *Frame times of the paraboloid scene (figure 6) with different grid resolutions. The first three columns are in milliseconds. Besides the total frame time, we measured separately the time of rendering the caustic beams (Caustics) and the time without the caustics pixel shader (Without PS). Note, that the high fill-rate is the bottleneck of the algorithm.*

To measure the performance of the geometry shader – which is definitely a critical stage on Shader Model 4 GPUs – we also run the tests without the costly ray-marching pixel shader. These results can be read under the caption "without PS".

6. Discussion and Future Work

Using the projected grid concept the performance is not significantly influenced by the geometric complexity. The main problem of rendering volume caustics is finding the golden mean between shading quality and fill-rate. When using a large grid resolution, the caustic effect becomes accurate, but several beams are rasterized to the same pixel. On the other hand, reducing the grid resolution introduces sampling artifacts which are particularly noticeable at geometric discontinuities (silhouettes).

As a future work, we would like to add shadowing support and multiple specular bounces to our implementation. An important form of the latter is the support of double-sided refractions. During the geometry shader execution, approximate ray-tracing methods can be used to determine the termination points for the caustic beams.

The other main future direction is the implementation of the adaptive hierarchical sampling in light's image space, similarly to Wyman's work[9]. Initially using a coarse vertex grid, the geometry shader can look for discontinuities in the geometry buffer to refine the resolution of the projected grid. Since we are using the grid for beam extrusion and not for direct rendering, the cracks introduced by such subdivisions do not cause artifacts. An adaptive algorithm would also reduce the overhead of discarded triangles in the geometry shader.

The next important way to attack the fill-rate bottleneck is off-screen rendering. Instead of rendering caustics to the screen directly, the beams can be rendered into a downsampled buffer, significantly improving the performance, while only slightly degrading the quality.

Acknowledgement

This work has been supported by Crytek GmbH.

References

1. M. Ernst, T. Akenine-Möller and H. W. Jensen Interactive Rendering of Caustics using Interpolated Warped Volumes *Proceedings of Graphics Interface 2005*, pp. 87-96 (2005) 1, 2, 3, 4
2. H. W. Jensen Global Illumination Using Photon Maps *Proceedings of 7th Eurographics Workshop on Rendering*, pp. 21-30 (1996) 2
3. H. W. Jensen, P. H. Christensen Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps *Computer Graphics (SIGGRAPH 98)*, pp. 311-320 (ACM Press, 1998) 2
4. T. J. Purcell, C. Donner, M. Camarano, H. W. Jensen and P. Hanrahan Photon Mapping on Programmable Graphics Hardware *Graphics Hardware*, pp. 41-50 (Eurographics Association, 2003) 2
5. L. Szirmay-Kalos, B. Aszódi, I. Lazányi and M. Premecz Approximate Ray-Tracing on the GPU with Distance Impostors *Computer Graphics Forum (Eurographics 2005)*, pp. 695-704 (2005) 2
6. T. Umenhoffer, G. Patow and L. Szirmay-Kalos Caustic Triangles on the GPU *Proceedings of of Computer Graphics International*, (2008) 2, 3
7. C. Wyman, C. Dachsbacher Improving Image-Space Caustics via Variable-Sized Splatting *Technical Report* UICS-06-02, University of Utah (2006) 2
8. C. Wyman Hierarchical Caustic Maps *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, pp 163-171 (2008) 1, 2
9. C. Wyman, G. Nichols Adaptive Caustic Maps Using Deferred Shading *Computer Graphics Forum 28(2)*, pp. 309-318 (2009) 1, 2, 6
10. P. S. Heckbert, P. Hanrahan Beam Tracing Polygonal Objects *Computer Graphics (SIGGRAPH 84)*, pp. 119-127 (ACM Press, 1984) 2
11. T. Nishita, E. Nakamae Method of Displaying Optical Effects within Water using Accumulation Buffer *Computer Graphics (SIGGRAPH 94)*, pp. 373-379 (ACM Press, 1994) 2
12. K. Iwasaki, Y. Dobashi and T. Nishita An Efficient Method for Rendering Underwater Optical Effects Using Graphics Hardware *Computer Graphics Forum, 21(4)*, pp. 701-712 (2002) 2
13. C. Johanson Real-Time Water Rendering - Introducing the Projected Grid Concept *Master of Science Thesis (Lund University*, (2004) 2

Figure 6: *Test renderings of various volume caustics using our algorithm. Left: a paraboloid surface focuses the light. (Rendering time with volumetric shadows: 30 FPS). Right: focused reflections of a car body. For this quality a* 128×128*-sized projected grid was necessary. Because of the extremely high fill-rate the rendering time dropped to 8 FPS. Both images were rendered in* 900×600*.*

14. M. Müller, S. Schirm and S. Duthaler Screen Space Meshes *Proc. of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 9-15 (2007) 2