Real-Time Volume Caustics with Adaptive Beam Tracing

Gábor Liktor * Carsten Dachsbacher †
Computer Graphics Group / Karlsruhe Institute of Technology

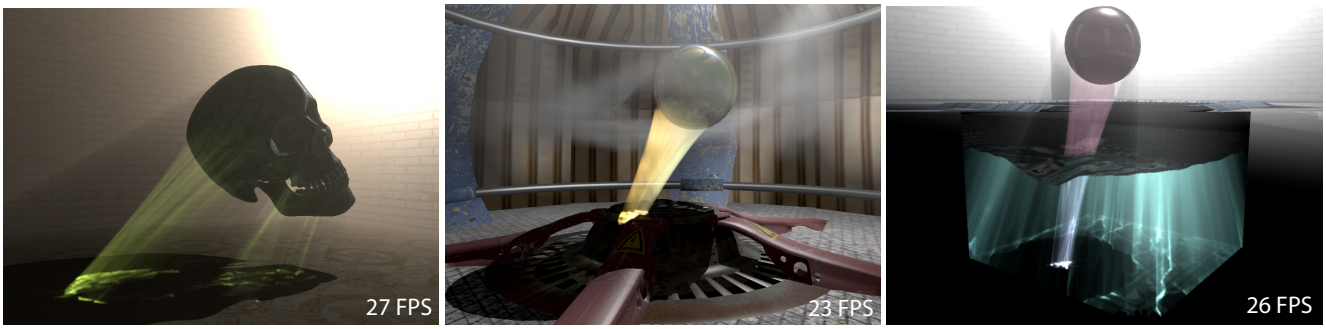


Figure 1: *Our method renders surface and volume caustics using approximate beam tracing. These results demonstrate two-sided refractions, inhomogeneous participating media as well as multi-bounce light-surface interactions rendered at real-time frame rates.*

Abstract

Caustics are detailed patterns of light reflected or refracted on specular surfaces into participating media or onto surfaces. In this paper we present a novel adaptive and scalable algorithm for rendering surface and volume caustics in single-scattering participating media at real-time frame rates. Motivated by both caustic mapping and triangle-based volumetric methods, our technique captures the specular surfaces in light-space, but traces beams of light instead of single photons. The beams are adaptively generated from a grid projected from the light source onto the scene's surfaces, which is iteratively refined according to discontinuities in the geometry and photon distribution. This allows us to reconstruct sharp volume caustic patterns while reducing sampling resolution and fill-rate at defocused regions. We demonstrate our technique combined with approximate ray tracing techniques to render surfaces with two-sided refractions as well as multiple caustic light bounces.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

Keywords: volume caustics, beam tracing, specular effects, level-of-detail

1 Introduction

Reflection, refraction, and the scattering of light as it interacts with surfaces and different materials can result in interesting visual effects and stunning light patterns. However, simulating these effects is often computationally expensive and interactive rendering of dynamic scenes with changing lighting, materials, and geometry is challenging. In particular this holds for caustics and volumetric effects, and of course for the combination of both. Methods such as (bidirectional) path tracing [Veach 1997] and photon mapping [Jensen 2001] are widely used to compute accurate results, however, not at interactive speed. Radiosity-based methods, such as [Dachsbacher et al. 2007], and precomputed radiance transfer, e.g. [Sloan et al. 2002], are typically not well suited to render caustics, as the high frequency light transport forming the caustics needs to be represented in the respective basis functions.

Two different approaches, however, allow for interactive rendering of caustics: particle tracing, e.g. [Wyman and Nichols 2009; Hu et al. 2010] (discussed in Sect. 2), and beam tracing methods [Ernst et al. 2005]. Methods that fall within the latter category compute beams of light reflected or refracted at a part of a surface, e.g. a triangle in a mesh. Every beam is bounded by a base triangle and three bilinear patches, and its contribution to the image is computed by intersecting the beams with surfaces (surface caustics), and by intersecting eye rays with the beam (volume caustics). Computing the inscattered light for each ray-beam intersection is non-trivial and the radiance along these "warped volumes" has been deeply analyzed by Ernst et al. [2005]. Liktor and Dachsbacher [2010] demonstrated a variant of this method generating the warped volumes and accumulating inscattered light entirely on the GPU, however, in a brute force manner with a huge number of beams and high fill-rate consumption. Recently, Hu et al. [2010] described a fast technique resorting to accumulating the inscattered light using many lines (light rays) instead of beams, which, however, requires rendering a large number of lines to not to suffer from undersampling problems.

We present a novel scalable method for rendering volume caustics in single-scattering participating media which is based on adaptive beam tracing [Heckbert and Hanrahan 1984]. We render the scene from the light source's view to capture the locations of the first light-surface interaction. We then generate the beams from this projected grid and use approximate ray tracing algorithms to handle two-sided refraction and multiple bounces. The beam generation adapts to geometric discontinuities of the specular objects, the light's intensity in the case of spot or textured light sources, and to the beams' contribution in the camera image. Our method does not require any precomputation and handles fully dynamic scenes. We also present a tight bounding volume computation that saves fill-rate when accumulating the inscattered light from caustic beams.

*e-mail: gabor.liktor@kit.edu
e-mail: dachsbacher@kit.edu

I3D 2011, San Francisco, CA, February 18 – 20, 2011.
© 2011 ACM 978-1-4503-0565-5/11/0002 $10.00

2 Previous Work

In principle, rendering surface and volume caustics can be easily accomplished using path tracing, but capturing the high frequency caustics adequately requires bidirectional methods such as bidirectional path tracing, metropolis light transport [Veach 1997], or photon mapping [Jensen 2001]. However, the prohibitively high render cost of these methods (for converged images) is typically only affordable for off-line rendering. Recently, Wang et al. [2009] presented a method for interactive global illumination, including surface caustic rendering, based on a fast GPU-based photon mapping algorithm of Zhou et al. [2008].

Surface Caustics Most methods that render surface caustics at interactive speed work similar to photon mapping and trace photons from the light sources until they hit a diffuse surface. Typically they avoid costly ray-triangle intersections by using image-based approximations of the geometry, and the nearest neighbor search by accumulation or splatting of the photons. The latter can take place in texture space [Szirmay-Kalos et al. 2005], image space [Dachsbacher and Stamminger 2006; Wyman and Dachsbacher 2006; Shah et al. 2007; Wyman and Nichols2009], or the objects' coordinate system [Umenhoffer et al. 2008]. Photon splatting typically suffers from the varying photon density resulting in noise in regions that receive few photons only. Wyman and Dachsbacher[2006] address this problem by adapting splat sizes. Recent work of Wyman et al. [2008;2009] renders high-quality caustics by hierarchical sampling of the caustic map. Umenhoffer et al. [2008] reconstruct smooth caustics from multiple reflections and refractions (computed using layered distance maps [Szirmay-Kalos et al. 2005]) by rendering caustic triangles instead of splatting.

Ernst et al. [2005] render caustic beams by rasterizing a convex bounding prism and using a pixel shader to accumulate the inscattered light. Their algorithm yields high quality for surface and volume caustics, however, it does not handle occlusions and requires CPU assistance. We improve on this method in several aspects by computing tighter bounding volumes, supporting multiple bounces of light, and adaptively generating caustic beams.

Yu et al. [2007] also present a geometric method based on caustic surfaces. These surfaces are swept by the foci of light rays and thus can be used to visualize caustic phenomena. Instead of using beams to discretize rays, their approach uses the two-plane parameterization of the General Linear Camera model [Yu and McMillan 2004] (cross-slit camera) to estimate caustic surfaces by finite sets of rays. Their characteristic equation for finding caustic surfaces can be also interpreted as the area of the triangles formed by triplets of neighboring rays, closely related to the *area function* used in this paper.

Volume Caustics Rendering is much more intricate if participating media is present. Sun et al. [2008] render caustics from refractions in material with varying scattering properties at almost interactive speed. They trace photons and accumulate the out-scattered radiance along a line segment (in between two refractions) into a 3D grid. The photon tracing is accelerated representing the refractive object using an octree. The final image is generated by ray marching along primary rays through the grid. Eikonal rendering [Ihrke et al. 2007] also achieves interactive rendering using wavefront tracing, but requires precomputation and thus does not allow fully dynamic scenes. Nisthita and Nakamae [1994] render underwater volume caustics using a beam tracing variant. Iwasaki et al.[2002] adapted this algorithm for GPUs, however, they used constant shading of the caustic beams resulting in blocky artifacts when using too few beams. Recently, Hu et al. [2010] described a technique for volume caustics which is based on tracing photons and rendering line primitives in image space. Their method achieves interactive to real-time performance, but can suffer from undersampling problems in regions where light rays are highly divergent. Most related to our work, Liktor and Dachsbacher [2010] render volume caustics using beams generated from a projected grid. However, the beam generation is not adaptive as in our method, and the accumulation of the beams consumes a vast amount of fill-rate due to the use of suboptimal bounding prisms.

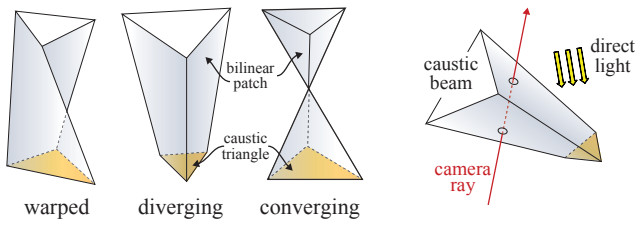


Figure 2: *Caustic beams, formed by a base triangle (yellow) and three bilinear patches (blue), can represent warped, diverging, or converging bundles of rays (left). To compute the inscattered light from a beam one intersects the warped volumes with camera rays (right).*

3 Overview

Generally speaking the primary goal of all interactive caustics rendering methods is to decouple the caustic photon distribution starting at the light sources, from the gathering of photons scattered towards the camera. The gathering of inscattered light is mostly performed by rasterization, as it trivially solves the direct visibility problem. Our work is motivated by the observation that accumulating caustic light contributions using bounding volumes or lines is largely bound by the cost of the rasterization itself. Triangle-based methods are usually more scalable in terms of fill-rate and screen resolution, but less flexible due to their direct dependency on the surface geometry.

We combine ideas of caustic mapping approaches with triangle-based caustic rendering to provide adaptive sampling in the light's image space while reducing the rasterization cost as much as possible. Fig. 3 provides an overview of our algorithm. Akin to caustic mapping, we first capture all directly visible surfaces from the light source by rasterizing them into geometry buffers. Then we transform this representation into a triangle-based one by resampling the geometry buffer according to a 2D grid. Instead of using a regular grid with a resolution fine enough to faithfully represent all surface details, we start with a coarser resolution and adaptively refine the grid in subsequent steps. Each triangle in the sampling grid is a base of a caustic beam, which we trace across surfaces to accumulate focused light in the volume.

For every beam we then need to determine the radiance scattered towards the camera depending on the phase function and density of the participating media. For this we rasterize bounding volumes of the caustic beams and perform the integration of the inscattered light in a pixel shader.

When also considering occlusion and higher order caustics, we store the primary beams after generation for later passes. The beams are then intersected with the surrounding scene geometry using approximate ray tracing [Szirmay-Kalos et al. 2005]. The challenging part of higher order beam tracing is the non-trivial splitting of beams where a surface is partially intersected. We handle this problem similarly to the primary adaptive grid refinement, however, using scene geometry impostors instead of the light space geometry buffer.

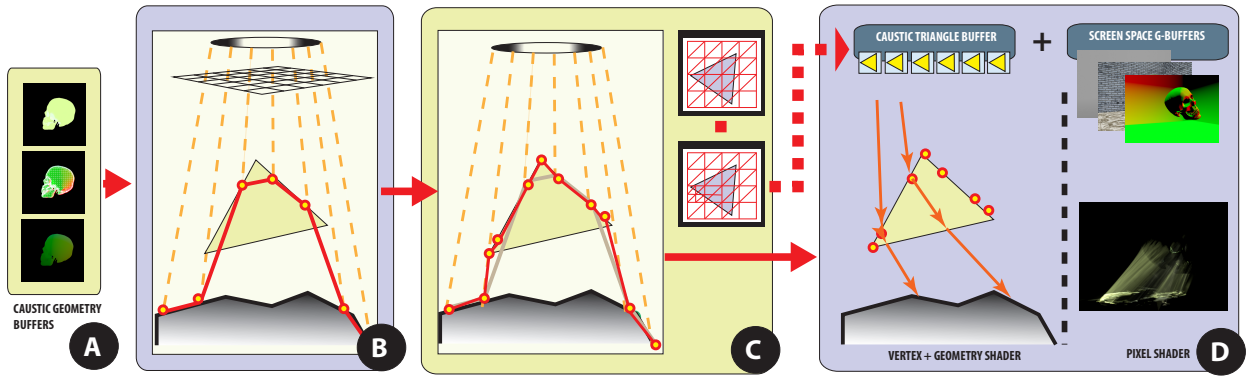


Figure 3: *The outline of our algorithm. The caustic generator surfaces are rendered into a light G-buffer (A). A regular grid is projected to the sampled geometry (B), then adaptively refined based on surface discontinuities and camera distance (C). Finally, the beam bounding volumes are extruded in the geometry shader, and accumulated by rasterization to the screen (D). Screen-space G-buffers provide the data for culling the beams and rendering surface caustics.*

4 Beam Generation

The adaptive beam generation from rasterized surfaces in the light's G-buffer is the key component of our method. In this section we detail the adaptive refinement process, and the formation of tight bounding volumes.

4.1 Hierarchical Projected Grid

In the first pass of our algorithm, we rasterize the specular surfaces as seen from the light source to off-screen geometry buffers storing positions, normals, and material properties (including transparency, index of refraction, etc.). This essentially provides a simple 2D parameterization over the directly lit surfaces. Note that this step is identical to the initial phase of standard caustic mapping algorithms, but instead of originating caustic photons at the sampled locations, we use this texture parameterization to project a planar grid to the lit surfaces. Taking a coarse, regular grid with the same texture parameterization, the projection step becomes trivial as each vertex can be moved to the position sampled from the corresponding G-buffer location.

We can regard this projected grid as a downsampled, piecewise linear approximation of the lit surfaces. The main advantage of capturing the surfaces in this way is the decoupling of geometric complexity from the actual caustic generation process: the sampled mesh can have densely triangulated as well as coarse regions which do not necessarily coincide with the caustic sampling. Instead of requiring surfaces to be modeled with uniform mesh density, we use this new "light-space approximation" to generate the primary caustic beams due to the first light-surface interaction. Each cell of the initial regular grid is treated as two triangles to later form caustic beams with triangular bases. Naturally, invalid triangles crossing the silhouettes of specular objects or lying outside are omitted.

Of course, projected grids have limitations as well: the boundaries of generated beams typically do not match the silhouettes (discontinuities) of the surfaces. We can largely suppress these problems using adaptive subdivision, however, at the expense of generating more beams. The goal of our hierarchical projected grid method is to generate caustic beams that adapt not only to discontinuities of the specular object, but also to the light's emission characteristics, and the beam's contribution to the final rendered image.

4.2 Split Heuristics and Refinement

After generating potential caustic triangles from the initial coarse grid we perform adaptive refinement that first detects undersampled regions in the grid, and splits the corresponding grid triangles into smaller ones. At the core of this procedure is an oracle deciding whether a beam should be kept, subdivided or discarded. The decision is based on an empirically set-up heuristic depending on three factors:

- **Discontinuities of the Specular Surface** To render detailed caustics we refine beams if we detect that they span geometric discontinuities. For this we compare the sampled surface parameters, i.e. normal and depth, at the location where the beam edges start, and split the triangles if they differ significantly (similar to [Nichols et al. 2009]).

- **Beam Energy** We also account for the beams' energy and their contribution to the final image. Divergent beams are typically unproblematic (unless their energy is very high) as their contribution is distributed rather evenly across their projected screen area. Convergent beams, however, focus the light and are refined with higher priority. In case of non-isotropic light-sources, e.g. spotlights or textured lights, we could account for lighting discontinuities in this step as well.

- **Camera Parameters** Lastly, we consider the relative position of the beams to the camera. Beams closer to the viewer are further refined than those that are far away and point away from the camera. For this we would have to consider the spatial extent of the entire beam, and not only the caustic triangle. As the exact geometry of the beam is not yet known in this step, we use a simple but well working estimate: we find the closest point to the camera on the beam's central ray (average of the three beam edges), and we then base our decision on the distance of this point to the camera.

The whole refinement process is implemented using a geometry shader which allows outputting variable amount of vertices and streaming data back to GPU memory. In out experiments 3 to 4 refinement steps proved to be sufficient for most of the cases (Fig. 4).

4.3 Bounding Volume Extrusion

Finally, we have to invoke the pixel shader on fragments influenced by each caustic beam. The previously refined grid and the light source defines incident beams on the caustic surface. These beams may get both reflected and refracted on the surface originating the caustic beams. For now, let us assume that we generate one caustic beam for every base triangle, i.e. the surfaces are reflective, but not transparent. We will extend this to surfaces that are both reflective and refractive and to multiple bounces of beams in Sect. 6. If the vertices of the base triangle are denoted as $\mathbf{v_0}$, $\mathbf{v_1}$, $\mathbf{v_2}$, and the edges of the beam (the reflection or refraction direction of the light impinging on the caustic triangle's vertices) as $\mathbf{d_0}$, $\mathbf{d_1}$, $\mathbf{d_2}$, then the beam is bounded by the edges $\mathbf{v_i} + t_i\mathbf{d_i}$, $i \in \{0..2\}$; where two of these edges form a bilinear patch. Now we need to generate a bounding volume for every beam that can actually be rasterized, as bilinear patches cannot be rendered directly.

Extrusion Length In order to compute the bounding volumes we need to first determine the required extrusion length of the beams. A trivial maximum length can be computed by intersecting the beams with the view frustum, but we also handle occlusion from other objects in the scene. For this we use approximate ray tracing, as described by Szirmay-Kalos et al. [2005], where the geometry shader intersects the beam edges with surrounding geometry to determine the extrusion length. Furthermore, we only want to rasterize caustic beams that have a significant contribution to the resulting image. It means that if a beam is divergent, there is a specific distance above which we consider the contribution of the beam negligible, therefore we maximize the beam extrusion using energy approximation.

Area Function As we describe in Sect. 5, the caustic radiance depends on the change of the cross section along the beam. Where the cross section parallel to the caustic triangle is smaller than its area, the rays get focused. The area of the caustic triangle can be easily computed using the cross product of the edges (in fact the length of the cross product is the double of the area). In order to approximate the caustic radiance, we need a way to evaluate the cross-section of the beam at any distance along the rays. Ernst et al. [2005] applied a special coordinate system, which we will call *beam space* in this paper, where the cross-section calculation can be performed efficiently. Here we briefly overview the idea for completeness, for details please refer to [Ernst et al. 2005]. The coordinate system is chosen so that the origin coincides with one vertex of the caustic triangle, the y-axis is the triangle normal, and the beam direction vectors are scaled so that their y-coordinate is one. Using this coordinate system, it is possible to express the area of the parallel cross-section of the beam as a second order expression $A(y)$ over the local y-coordinate. We precompute the three coefficients of $A(y)$ during the beam-generation step, so later on we can evaluate it with a single dot product.

Bounding Volumes Prior to rasterization, a geometry shader processes each caustic triangle and emits new triangles forming the bounding geometry of the caustic volume. However, the extrusion is non-trivial, as the sides of the caustic beams are bilinear patches, which cannot be represented by a finite set of triangles in the general case. Ernst et al. [2005] proposed to set up touching planes for each side of the beam: three touching planes are constructed, each containing one edge of the base triangle, and one of the top vertices such that the entire beam lies in the negative half space of the plane (Fig. 5). The intersection of these planes then yields the edges of the bounding volume.

Improved Bounding Volumes Accumulating the beams' contribution consumes a considerable amount of fill rate, and thus it is of utmost importance to keep the number of rasterized fragments for the individual beams as low as possible. The bounding volume generation by Ernst et al. [2005] works well for slightly warped beams, but generates overly large volumes in the worst case as depicted in Fig. 5. It is easy to recognize that the bounding volumes could be much tighter if one would perform the same procedure using the top edges as base edges of the bounding prism, then intersect the two prisms with each other. We use a simplification to make this computationally intensive task practical: as each bounding plane is defined by one triangle edge and one vertex of the opposite triangle, we can take the formed triangle directly as one side of the bounding volume. Thus the final bounding volume is defined by six triangles, selected for each edge independently. This method greatly reduces fill-rate, however, suffers from numerical problems: the bounding volume might have multiple consistent triangulations (i.e. one side of the beam is close to planar), but we select each triangle independently. Instead of performing a costly test again to ensure a watertight bounding volume, we revert to the bounding prism method when we find multiple triangle candidates for any edge.

The bounding prism method yields the worst approximation to the beam boundaries, when the beam is strongly warped. This is usually the case when the beam is focused so that the top triangle "flips over" compared to the bottom one. Our method results in a tighter convex hull with the same number of triangles, significantly reducing the fill-rate in such cases (Fig. 6).

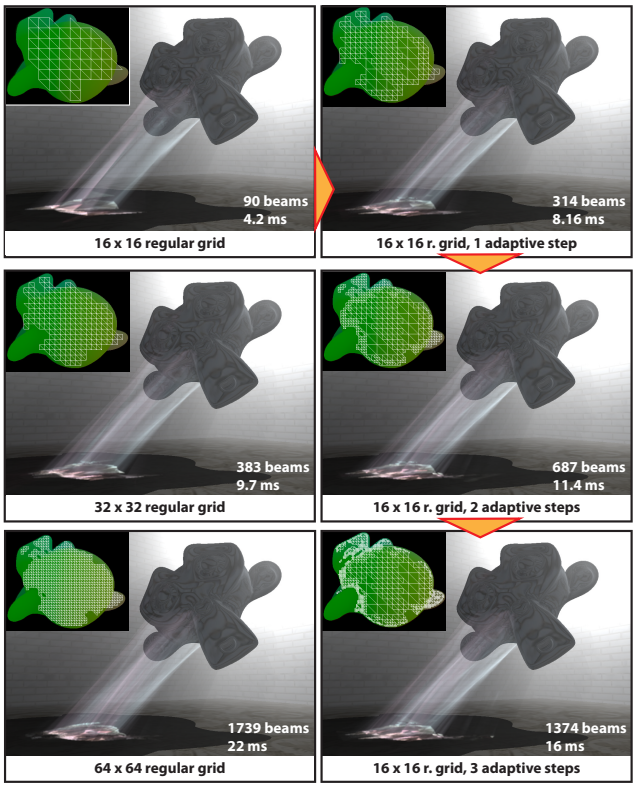


Figure 4: *Quality and performance of the caustics rendering method using different grid resolutions and refinement steps. The left side uses static projected grids (equivalent to Liktor et al. [2010]), while the right side shows the progress of our adaptive method using the smallest regular grid as a basis. The small insets visualize the beam origins in light s image space (Please zoom in in the electronic version). The caustics beams in the blank areas were discarded due to total internal refraction. The time values are for the volume and surface caustic effect only.*

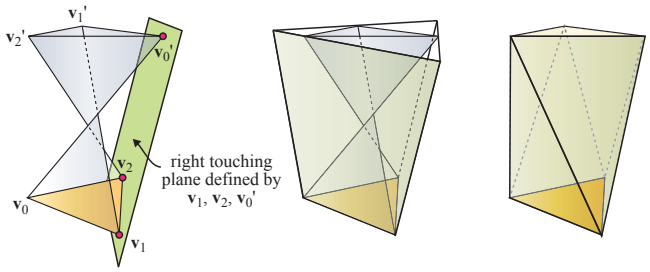


Figure 5: *Computing tight bounding volumes for warped beams is non-trivial.* ***Left, middle:*** *Ernst et al. propose to set up three touching planes each containing one edge of the base triangle and one vertex of the top triangle.* ***Right:*** *we select 6 triangles using the edges of the two triangles and consequently one vertex for each on the opposite triangle. In case of a strongly warped beam this can greatly reduce the pixel coverage of the volume.*

5 Inscattering and Surface Caustics

In the last step of our method, the fragment shader computes the amount of radiance scattered towards the viewer. In principle, every beam is intersected (at every fragment) with a camera ray and the contribution is (Fig. 7):

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

where $L(\omega)$ is the radiance scattered towards the viewer, Δx is the interval of the ray-beam intersection, σ_t the extinction coefficient, $\sigma_s(\mathbf{x})$ the scattering coefficient at a point $(\mathbf{x})$, $p(\cdot)$ is the phase function, and $s_1 + s_2$ form the length of the light's path from the caustic triangle to the camera. Assuming that the beams are narrow enough in image space, a single sample in the integration domain is sufficient to approximate this integral with a low (and imperceptible) error:

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

For the evaluation, the shader has to perform the following steps for every fragment:

- Get the intersection of the beam (not the bounding volume) and the viewing ray.
- Compute the incident radiance at the sample on the ray and the scattered fraction thereof towards the camera.

5.1 Beam Intersection

When the bounding volumes emitted by the geometry shader are rasterized, we need to determine if the camera ray corresponding to each fragment intersects the beam, and if so the location of the intersection. Ramsey et al. [2004] describe an exact computation of ray-bilinear patch intersection, however, it is too costly for a large number of caustic beams (in our typical scenes the number of beams is thousands). Instead we resort to the scan plane-based estimate [Ernst et al. 2005] (Fig. 7). The scan plane is an auxiliary construction, in principle an arbitrary plane containing the camera ray. The intersections of the scan plane and the edges of the caustic beam form a triangle containing the caustic ray segment of length Δx. The inaccuracy of this method stems from the fact that the edges of the scan plane triangle are actually curved. This error can be minimized by choosing the normal vector of the plane to be close to the normal of the caustic triangle:

$$\mathbf{n_p} = \mathbf{r_{view}} \times (\mathbf{r_{view}} \times \mathbf{n}\ )$$

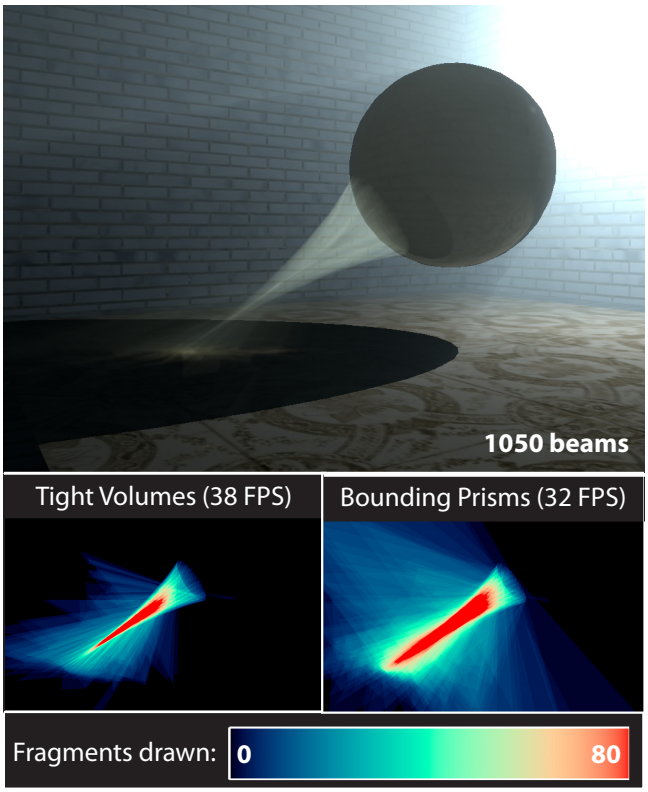


Figure 6: *As the rendering of caustics is highly pixel shader bound, it is crucial to minimize the number of fragments rasterized for each beam.* ***Top:*** *a glass sphere generates strongly focused caustic beams. This is a typical scenario when the bounding prism method of Ernst et al. [2005] generates looser volumes than our 6-triangle method. We have visualized the fill-rate of our volume generation (**bottom left**) and the bounding prism method (**bottom right**), rendering equivalent images.*

where $\mathbf{r_{view}}$ is the viewing ray and $\mathbf{n}$ the triangle normal. The desired length of the ray segment Δx can then be obtained by a cheap 2D ray-triangle edge intersection.

5.2 Radiance Estimation

After having determined an intersection of a view ray and a beam, we need to compute the radiance scattered towards the viewer. The amount depends on the reflected or refracted radiance at the specular surface and the cross section of the beam at the intersection location. If we would neglect attenuation in the participating media – here only for simplicity of the explanation – then the radiant flux due to caustic triangle Δ_c

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

would be constant along the beam ($\mathbf{p}$ is a point on the infinitesimal surface element, θ is the angle between $\mathbf{n}$ and ω). Caustics are formed due to the narrowing of the beams, and thus increased radiance. Note that we account for attenuation in our implementation.

We compute the inscattered radiance as described by Ernst et al. [2005]. To perform the intersection, the fragment shader requires considerable amount of data: $\mathbf{v_i}$, $\mathbf{d_i}$, the area of $\triangle(\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2})$, and the radiance values at the base triangle's vertices. It is important to

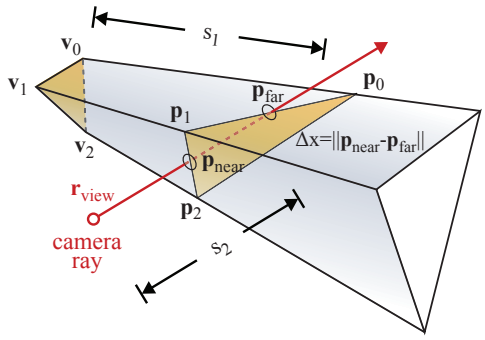


Figure 7: *Intersection of a caustic beam with a camera ray. The contribution of the beam to the image is computed using the intersection length Δx, the cross section at the vertices of the scan triangle $\mathbf{p_i}$, and the distance s_1, from the caustic triangle.*

tightly pack this information into vertex attributes in the preceding geometry shader.

To intersect the beams we first transform the view ray to the beam's local space introduced in Sect 4.3. The fragment shader takes a single representative point on Δx (in our implementation we simply take the midpoint) and then evaluates the area function $A(y)$. This cross section area determines the radiance scaling according to the (de)focus of the beam. Finally the shader computes the fraction of light scattered towards the viewer according to the scattering in the participating medium.

Caustic beams rendered in this manner suffer from clipping artifacts: when intersecting solid surfaces, the volumetric effect suddenly disappears as the rasterized fragments are occluded. We solve this problem by applying a smoothening in the spirit of soft-particle methods [Lorach 2007]: using the difference between the screen space depth of the caustic receiver and the depth value of the beam's faces we can smoothly attenuate the caustic effect to avoid discontinuities.

5.3 Surface Caustics

Our method can also render surface caustics using the very same beams as for volume caustics. For this we generate an additional geometry buffer for the camera view prior to caustics rendering. This buffer holds the diffuse surface colors and view-space depth values (z-coordinate) of the caustics receivers. In order to add surface caustics to our method we only need to extend the aforementioned caustic fragment shader: during rasterization we sample the camera-view geometry buffer and perform point-in-volume tests on each surface point covered by the beam, similarly to [Ernst et al. 2005]. First we intersect the beam with a plane parallel to the caustic triangle. This is trivial, as the y coordinate of the surface point defines its distance along the caustic triangle normal in the local beam space (refer back to 4.3). If the point is inside the resulting triangle, we use the barycentric coordinates to interpolate between the initial radiance values of the beam edges, and $A(y)$ to determine the amount of relative radiance change. Note that this screen-space method is completely independent from the receiver surfaces and automatically adapts to high frequency details (in contrast to splatting methods, where the splatting resolution needs to be increased at high frequency details).

6 Multi-Bounce Caustics

So far we considered beams originated from the first light-surface intersections. Our method easily extends to multiple surface interactions, including refractive and reflective surfaces, two-sided refraction for transparent objects, and also multiple subsequent light bounces across the scene.

Reflection and refraction Transparent objects typically both reflect and refract light. Note that it is sufficient to run the refinement process once for both reflection and refraction, since the heuristic only depends on the geometry, but not the material properties. After refinement, we compute the reflection direction for every incident beam's edges to create a new beam representing the reflection caustics. Since transparent objects typically refract light on both the front and back side, we handle two-sided refractions directly: we first compute the refraction beam at the first intersection, and immediately compute the second refraction. To intersect rays from inside the objects, we use a distance impostor: a cube map storing the surface normals and distance from the reference point (typically the object's center) [Szirmay-Kalos et al. 2005]. Computing both refractions at once speeds up the rendering, however, caustics inside the object cannot be rendered.

Multiple surface interactions The input of the beam rasterization step can be also used to take the simulation one step further and compute higher-order light bounces. Having the scene rasterized into impostors (using a representative center of projection), we can detect whether a caustic beam intersects other specular surfaces. Note that we can use a slightly modified version of the beam splitting oracle we introduced in Sect. 4. This allows us to split secondary beams at silhouettes (which is often intricate with beam tracing). Instead of examining the contents of the light's geometry buffer, we can now compare the difference of photon hits at beam edges and surface normals to detect depth discontinuities. Fig. 1. (right) demonstrates multiple refractions on the surface of water of a beam focused by a glass sphere.

7 Implementation

We have implemented our method using Direct3D 11 and HLSL (The algorithm only requires Shader Model 4 architecture). In this section we briefly summarize the implementation specific details of the above procedures.

Beam generation The generation of the primary caustic beams takes place in light-space and the adaptive subdivision is performed globally for the first reflective and refractive bounce. For this we use a geometry shader that is executed recursively (via stream-out feedback) for each refinement step. Instead of processing triangle primitives, we pack the vertices of each beam triangle into a single point primitive. This decision has positive effect on the performance of the subdivision for two reasons. First, the vertex shader can also do part of the triangle processing, taking off some burden from the geometry shader. Second, we expected that handling the data at higher granularity gives a hint to the pipeline that specific attributes belong together, and the geometry shader outputs 0-4 vertices instead of 0-12.

In all our examples, 3-4 iterations produced pleasing results for a coarse initial 16×16 grid. The beam extrusion is performed separately for reflection and refraction in the beam-rasterization pass, using the same adaptive base grid. If multiple bounces are enabled, an additional stream-out pass is necessary to determine the intersections with the environment. After having computed these intersections, the adaptive beam refinement is restarted as previously described.

Surface caustics Ernst et al. [2005] smoothly interpolate between neighboring caustic volumes using adjacent caustic triangles and instead of using a single area function, one area function is used for each triangle vertex and averaged over the caustic beams. One limitation in our current implementation is that we use a single area function per beam. This is due to the adaptive subdivision in the

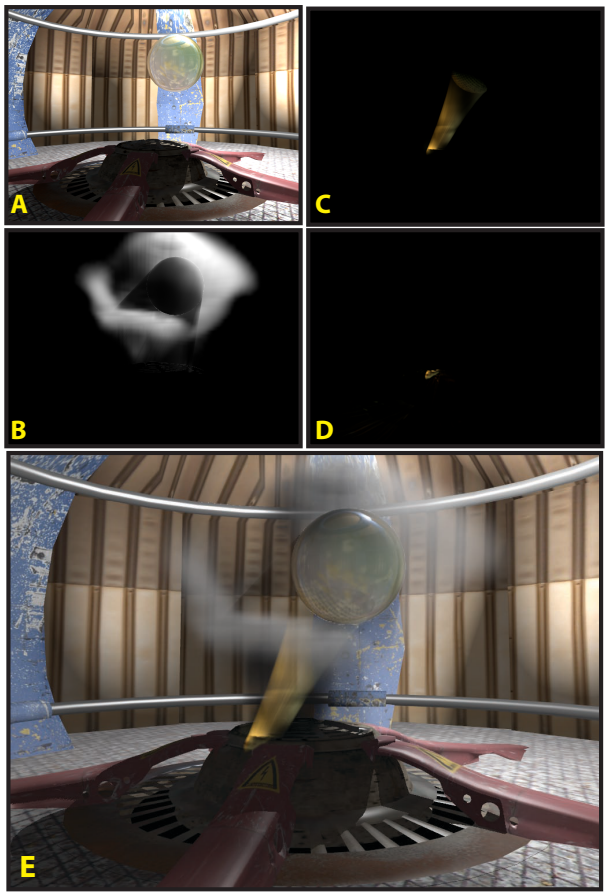


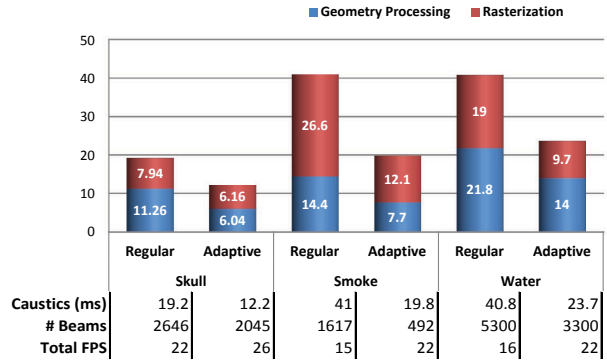
Figure 8: *The final image (E) is composited of the direct illumination of surfaces (A), volumetric lighting with shadows (B), the volume caustics (C) and the surface caustics (D). Note that we render surface caustics separately to perform low-pass filtering.*

geometry shader where triangle adjacency information is lost on the GPU after stream-out. As a result of using only one area function, small discontinuities appear at beam boundaries when using coarser grids. This is insignificant in case of the volume caustics, but causes sharp edges on the generated surface caustics. As our method primarily targets volume caustics rendering, we can still use the implicit surface caustics, but we have to apply low-pass filtering to remove sharp edges from the light patterns. Note that we only use the filter on the surface caustics. The advantage of this method is that the surface caustics comes virtually "for free", since it is directly evaluated during beam rasterization. When high quality surface caustics is required we recommend combining our method with caustic photon splatting methods, such as [Wyman and Nichols2009].

Volumetric Lighting As volume caustics are not plausible without the presence of lit participating media in the image, we have implemented additional volumetric illumination techniques. The final image is composed of: the direct surface illumination term, the lit participating media with volumetric shadows, and two caustic buffers (Fig. 8). We render surface caustics to a separate render target to apply the aforementioned low-pass filtering.

8 Results and Discussion

Figure shows three of our test scenes we have used to evaluate our method. All measurements have been made using a test platform



	Skull		Smoke		Water	
Caustics (ms)	19.2	12.2	41	19.8	40.8	23.7
# Beams	2646	2045	1617	492	5300	3300
Total FPS	22	26	15	22	16	22

Table 1: *Performance results on various test scenes. The above chart provides detailed information about the time spent on the generation of beam volumes and rasterization in milliseconds.*

equipped with an NVIDIA Geforce GTX 470 GPU and an Intel Core i7 920 CPU (Table 8). The featured images were all rendered at a resolution of 1024×768 pixels. The resolution of the initial coarse grid is 16×16, and we performed three steps of adaptive subdivision. To evaluate the performance gain of our method compared to static beam generation [Liktor and Dachsbacher 2010], we have also rendered the same images on a 64×64 regular grid. This resolution is a subjective choice, which we have found to be the closest match to the adaptively generated images. As the relatively simple implementation of the finely sampled volumetric lighting takes considerable amount of rendering time, we measured the timings of the caustics rendering method separately besides the total frame rate.

- The glass skull scene contains a simple environment receiving caustics from the translucent model; only single-bounce refraction caustics are rendered. This is the most frequent scenario for real-time caustics, when the frame budget for additional global illumination effects is low.
- The water scene demonstrates a complex, multi-bounce effect when refracted caustics beams are refracted once more on the water surface. Underwater caustics is also challenging in terms of performance, as hardly any caustic beams are occluded and thus the fill-rate is very high.
- The smoke scene demonstrates volumetric caustics in inhomogeneous participating media. When evaluating the incident radiance, the fragment shader performs ray-marching inside the participating media to numerically approximate the volume rendering equation.

One of the ideal examples for the advantages of the adaptive scheme is the water scene, where both refractors are relatively smooth surfaces, but there is a sudden discontinuity at the silhouette of the sphere as seen from the light source. While regular grid can only capture the surface borders at a high resolution, our method increases the number of beams at the boundary regions while keeping the resolution low at the remaining parts.

Obviously, if high detail is required everywhere, then the adaptive subdivision does not significantly reduce the total number of beams. This is the case in the skull example, where the refractive geometry contains high-frequency details (teeth, eye sockets). As the generation of the beams is a geometry-intensive process, the rendering of very high-frequency refractors might perform better using an approach similar to [Hu et al. 2010]. On the other hand, the approximation of volume caustics with solid beam volumes always provides a continuous effect and does not suffer from line aliasing artifacts.

9 Conclusion and Future Work

In this paper we presented a real-time method for rendering volume and surface caustics with multiple reflections and refractions. It is based on adaptive beam tracing and can be entirely implemented on programmable graphics hardware. We showed how to adaptively generate and split the beams, and compute tight bounding volumes for an efficient accumulation of the beams' contribution.

Presumably the primary source of inaccuracy in our method is the approximate ray tracing that is used for multiple bounces, as impostors are generally not able to capture all surfaces in the scene. Although this can be sidestepped, e.g. using depth peeling, the emergence of fast methods for constructing spatial index structures for triangle-based ray tracing possibly suggests to rely on those in the future [Grün 2010].

An interesting direction of future work is the replacement of the geometry shader stage in the adaptive grid refinement step with a compute shader kernel. This would allow us to keep track of the topology of the grid, maintaining a spatial indexing structure. Using the adjacency information, we can generate a smooth transitions among the beams, thus significantly improve the quality of the surface caustics.

Acknowledgements

The first author of this paper has been funded by Crytek GmbH.

References

DACHSBACHER, C., AND STAMMINGER, M. 2006. Splatting indirect illumination. In *Proc. of the 2006 Symposium on Interactive 3D Graphics and Games*, 93–100.

DACHSBACHER, C., STAMMINGER, M., DRETTAKIS, G., AND DURAND, F. 2007. Implicit visibility and antiradiance for interactive global illumination. In *ACM Trans. on Graphics (Proc. of SIGGRAPH 07)*, vol. 26(3), 61.

ERNST, M., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Interactive Rendering of Caustics using Interpolated Warped Volumes. *Proc. of Graphics Interface*, 87–96.

GRÜN, H., 2010. Direct3d 11 indirect illumination. Presentations at Game Developers Conference.

HECKBERT, P. S., AND HANRAHAN, P. 1984. Beam Tracing Polygonal Objects. *Computer Graphics (Proc. of SIGGRAPH 84)*, 119–127.

HU, W., DONG, Z., IHRKE, I., GROSCH, T., YUAN, G., AND SEIDEL, H.-P. 2010. Interactive Volume Caustics in Single-scattering Media. In *Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 109–117.

IHRKE, I., ZIEGLER, G., TEVS, A., THEOBALT, C., MAGNOR, M., AND SEIDEL, H.-P. 2007. Eikonal Rendering: Efficient Light Transport in Refractive Objects. In *ACM Trans. on Graphics (Proc. of SIGGRAPH 07)*, vol. 26(3), 59.

IWASAKI, K., DOBASHI, Y., AND NISHITA, T. 2002. An Efficient Method for Rendering Optical Effects Using Graphics Hardware. *Computer Graphics Forum 21*, 4, 701–712.

JENSEN, H. W. 2001. *Realistic Image Synthesis using Photon Mapping*. A. K. Peters, Ltd.

LIKTOR, G., AND DACHSBACHER, C. 2010. Real-Time Volumetric Caustics with Projected Light Beams. *Proc. of the 5th Hungarian Conf. on Computer Graphics and Geometry*, 12–18.

LORACH, T., 2007. Soft Particles. NVIDIA White Paper.

NICHOLS, G., SHOPF, J., AND WYMAN, C. 2009. Hierarchical Image-Space Radiosity for Interactive Global Illumination. *Computer Graphics Forum 28*, 4, 1141–1149.

NISHITA, T., AND NAKAMAE, E. 1994. Method of Displaying Optical Effects within Water using Accumulation Buffer. *SIGGRAPH 94*, 373–379.

RAMSEY, S. D., POTTER, K., AND HANSEN, C. 2004. Ray Bilinear Patch Intersections. *Journal of Graphics, GPU, and Game Tools 9*, 3, 41–47.

SHAH, M. A., KONTTINEN, J., AND PATTANAIK, S. 2007. Caustics Mapping: An Image-Space Technique for Real-Time Caustics. *IEEE Trans. on Vis. and Computer Graphics 13*, 272–280.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed Radiance Transfer for Real-time Rendering in Dynamic, Low-frequency Lighting Environments. In *ACM Trans. on Graphics (Proc. of SIGGRAPH 02)*, vol. 21(3), 527–536.

SUN, X., ZHOU, K., STOLLNITZ, E., SHI, J., AND GUO, B. 2008. Interactive Relighting of Dynamic Refractive Objects. In *ACM Trans. on Graphics (Proc. of SIGGRAPH 08)*, vol. 27(3), 1–9.

SZIRMAY-KALOS, L., ASZÓDI, B., LAZÁNYI, I., AND PREMECZ, M. 2005. Approximate Ray-Tracing on the GPU with Distance Impostors. *Computer Graphics Forum*, 695–704.

UMENHOFFER, T., PATOW, G., AND SZIRMAY-KALOS, L. 2008. Caustic Triangles on the GPU. *Proc. of Computer Graphics International*.

VEACH, E., 1997. Robust Monte Carlo Methods for Light Transport Simulation. Ph.D. dissertation, Stanford University.

WANG, R., WANG, R., ZHOU, K., PAN, M., AND BAO, H. 2009. An efficient GPU-based approach for interactive global illumination. In *ACM Trans. on Graphics (Proc. of SIGGRAPH 09)*, vol. 28(3), 1–8.

WYMAN, C., AND DACHSBACHER, C. 2006. Improving Image-Space Caustics via Variable-Sized Splatting. *Journal of Graphics Tools 13*, 1.

WYMAN, C., AND NICHOLS, G. 2009. Adaptive Caustic Maps Using Deferred Shading. *Computer Graphics Forum 28*, 2, 309–318.

WYMAN, C. 2008. Hierarchical Caustic Maps. *Proc. of the 2008 Symposium on Interactive 3D Graphics and Games*, 163–171.

YU, J., AND MCMILLAN, L. 2004. General linear cameras. In *ECCV (2)*, 14–27.

YU, X., LI, F., AND YU, J. 2007. Image-space caustics and curvatures. *Computer Graphics and Applications, Pacific Conference on 0*, 181–188.

ZHOU, K., HOU, Q., WANG, R., AND GUO, B. 2008. Real-Time KD-Tree Construction on Graphics Hardware. In *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia 08)*.