# Interactive Image-Space Techniques for Approximating Caustics

Chris Wyman*
University of Iowa

Scott Davis†
University of Iowa

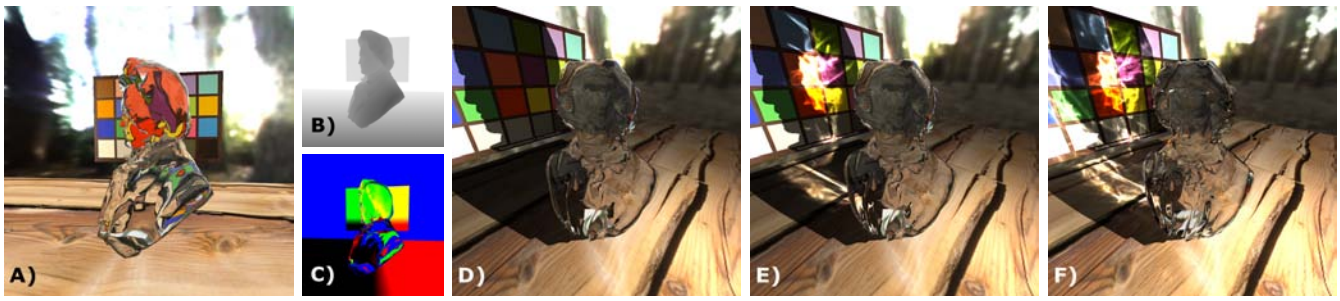


Figure 1: *Our approach to interactive caustics builds off work for interactive reflections and refractions. The first pass renders (a) the view from the light, (b) a depth map for shadow mapping, and (c) final photon locations into buffers. The second pass gathers the photons and (e) renders a result from the eye's point of view. The result significantly improves on (d) existing interactive renderings, and compares favorably with (f) photon mapping.*

## Abstract

Interactive applications require simplifications to lighting, geometry, and material properties that preclude many effects encountered in the physical world. Until recently only the most simplistic reflections and refractions could be performed interactively, but state-of-the-art research has lifted some restrictions on such materials. This paper builds upon this work, but examines reflection and refraction from the light's viewpoint to achieve interactive caustics from point sources. Our technique emits photons from the light and stores the results in image-space, similar to a shadow map. We then examine various techniques for gathering these photons, comparing their advantages and disadvantages for rendering caustics. These approaches run interactively on modern GPUs, work in conjunction with existing techniques for rendering specular materials, and produce images competitive with offline renderings using comparable numbers of photons.

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

**Keywords:** interactive rendering, caustics, image-space techniques, hardware

## 1 Introduction

Interactivity constraints in many applications limit rendered realism to those effects that run efficiently on graphics hardware. Unfortunately, efficient algorithms often must make significant simplifications to the underlying physical models. These simplifications typically restrict the global interactions that can occur between objects, and dynamic specular interactions are frequently eliminated first.

While perfect specular interactions are easily modeled using a ray tracing paradigm, these techniques do not apply well to the GPU pipeline. Brute-force ray tracing for mainstream applications seems distant (either via parallel implementations [Wald et al. 2002], ray tracing accelerators [Woop et al. 2005], or GPU ray tracing [Purcell et al. 2002]), but techniques for applying rasterization to interactively approximate specular effects are beginning to emerge (e.g., the work of Wyman [2005a; 2005b] and Yu et al. [2005]). Building off these techniques allows the consideration of more complex light interactions, such as caustics.

In this paper, we present an interactive image-space technique for approximating caustics. Our technique is physically based, though we introduce numerous steps in the process where approximation can reduce resource usage. The limitations include: restriction to point lights, no volumetric scattering effects, GPU numerical inaccuracies and aliasing, and deficiencies inherited from the underlying interactive reflection or refraction techniques. However, since humans miss many details of indirect effects such as shadows, color bleeding, and caustics, the results of our technique look quite compelling (see Figure 1).

The rest of the paper is organized as follows: Section 2 outlines related work on interactive reflections, refractions, and caustics. Section 3 describes our two stage process: emitting and storing photons from the light, and gathering and rendering these photons from the eye's point of view. Implementation details and results are outlined in Section 4, followed by discussion and conclusion.

## 2 Previous Work

One of the biggest challenges in interactive graphics is dealing with specular effects. These are particularly challeng-

*E-mail: cwyman@cs.uiowa.edu
†E-mail: scodavis@cs.uiowa.edu


I3D 2006, Redwood City, California, 14–17 March 2006.
© 2006 ACM 1-59593-295-X/06/0003 $5.00

ing as they require global knowledge of the scene. Even in ray and path tracing global knowledge comes at a cost, but for rasterization-based applications access to such data is severely restricted. Until recently, only rendering reflections or refractions through a few planar surfaces remained interactive [Diefenbach and Badler 1997]. Kay and Greenberg [1979] allowed slightly more complex geometry, but only under fairly severe restrictions.

More recent work by Ofek and Rappoport [1998] created virtual reflected geometry based on curved reflectors and Schmidt [2003] examined a similar approach for refraction. Yu et al. [2005] developed a technique for indexing reflection directions into a light field compressed into GPU memory. This allows for dynamically changing reflections as the reflector moves within a constrained region.

Wyman [2005a] offered a new technique for rendering refractive geometry inside an infinite environment using a two-pass image-space approach. While this method works best for convex geometry, use on concave refractors gives generally plausible results. An extension to this approach [Wyman 2005b] approximates refraction of nearby geometry via an additional pass.

These methods pertain to caustic research, as bright caustic regions occur where light focuses due to reflection or refraction. Another set of researchers have examined techniques explicitly for interactive caustic generation. One subset focuses on rendering caustics via photon tracing. This idea originates from backwards ray tracing [Arvo 1986], which stored contributions from photons in textures mapped onto scene geometry. A more flexible approach, photon mapping [Jensen 2001], stored photons in a three-dimensional kd-tree that allows for fast access during rendering. Interactive ray tracers [Parker et al. 1999; Wald et al. 2002] allowed distributed approaches to caustic generation using photon tracing methods (e.g., Günther et al. [2004] and Wyman et al. [2004]). Purcell et al. [2003] examined GPU-accelerated photon mapping, but concluded a uniform grid was better than a kd-tree for GPU-based computations.

Another approach to interactive caustics stems from research into beam tracing [Heckbert and Hanrahan 1984]. Watt [1990] applied backwards beam tracing to generate caustic polygons and used these to render caustics focused through a water surface. Nishita and Nakamae [1994] used a similar technique but account for volumetric effects by considering the entire prism-shaped caustic volume, and Iwasaki et al. [2002] outlined details necessary for an interactive GPU-based implementation. Ernst et al. [Ernst et al. 2005] improved the rendering of caustic volumes by using warped volumes instead of prisms.

Wand and Straßer [2003] uniformly sampled reflective objects, treating each sample as a point light source, and used commodity graphics hardware to perform a gather of the contributions from each sample. This technique generates caustics from complex and environmental light sources interactively, but as many of the other techniques described is limited to single-interface reflections. Additionally this gather operation becomes quite expensive, especially when finely sampling geometry.

Our approach utilizes a photon emission stage similar to backwards ray tracing and uses an image-space photon gather, inspired by the work on GPU-accelerated diffuse interreflection [Dachsbacher and Stamminger 2005]. One problem with any gathering approach is correctly weighting the contributions, and we use a Gaussian filter, reported by various researchers to work well for photon counting [Jensen 2001; Russ 2002].

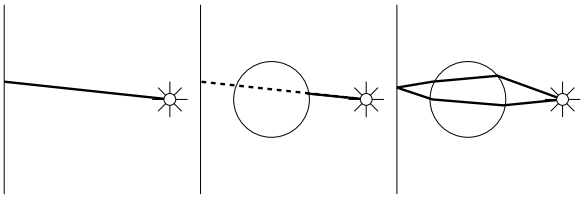

Figure 2: *Direct illumination (left) occurs when light reaches a surface, as opposed to shadows (center) where the light is blocked. Caustics (right) occur in regions where multiple paths from the light, focused via specular reflection or refraction, converge.*

Concurrently and independently of our work, Shah and Pattanik [2005] developed a similar method for interactive caustics. Their technique emits light using a vertex-tracing approach, similar to that of Ernst et al. [2005] and Ohbuchi [2003], gathers the results in a light-space texture, and performs some form of filtering. They also build on the refraction work of Wyman [2005a] and intersect background geometry similarly to the approach proposed by Ohbuchi [2003].

## 3 Image-Space Caustics

The typical approach to efficiently rendering caustics utilizes a two pass approach: emitting particles or beams from the light and gathering their contributions as viewed from the eye. Our method is no different, however the two passes have been modified to leverage the efficiencies of modern graphics accelerators.

### 3.1 Photon Emission

Traditional stochastic rendering techniques such as path tracing [Kajiya 1986], Metropolis sampling [Veach and Guibas 1997], and photon mapping [Jensen 2001], compute random paths of light through the scene or emit photons randomly from the light sources. While this eliminates aliasing artifacts seen with more regular sampling, rendering single rays in arbitrary directions is difficult to perform efficiently on current graphics cards. Thus we choose to emit photons in a regular pattern, defined by the image pixels in a rendering from the light's viewpoint.

Next we observe that caustics, like most global illumination problems, boil down to an issue of visibility. Caustic focal points occur where multiple paths from the light converge. In other words, if rendering from the light, locations visible multiple times belong to caustic regions just as shadows occur in areas not visible (see Figure 2).

This observation provides the motivation for our technique. Using recent work on reflections and refractions, we can relatively accurately render the light's view of a scene with complex specular geometry. Counting how many times the light sees a particular region is equivalent to counting how

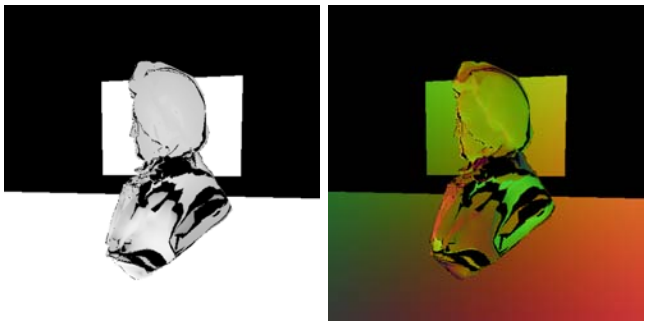

Figure 3: *In addition to the depth buffer and photon location buffer shown in Figure 1, buffers storing (left) attenuated photon intensity and (right) incident photon direction allow physically accurate caustic rendering.*

many photons hit that region, allowing an accurate determination of that region's intensity. This is similar to the concept of an illumination map [Arvo 1986], which counts how many photons hit each texel of a diffuse object. Since counting texel accesses is not straightforward on the GPU's streaming architecture, we instead chose to store the locations where photons land and perform the counting operation in a separate gather pass. This idea is similar to that of photon mapping [Jensen 2001], which stores photons in a relatively complex kd-tree structure.

Our photon emission stage can be summed up simply as a render pass as seen from the light. Instead of storing colors, we store the location of the first diffuse surface encountered in each pixel, as shown in Figure 1c. This process is analogous to shadow mapping, except as photons may change course via specularities, $x$ and $y$ values for intersections must be stored in addition to a $z$ value.

In order to generate physically accurate renderings, other buffers storing the incident photon direction and attenuated photon intensity must be rendered (as in Figure 3), though not all applications will require this accuracy. Furthermore, omnidirectional lights may result in caustics in many directions, so multiple renderings from the light may be necessary due to field-of-view limitations. For surfaces scattering photons multiple directions (e.g., via both reflections and refractions), multiple rendering passes would be required.

## 3.2 Photon Gathering

After storing photons in a buffer, modern graphics cards allow direct photon visualization relatively easily. For instance, a render-to-vertex-array approach can utilize these photon buffers as a vertex array of locations to directly visualize photons in a second pass (see Figure 4). Unfortunately, this approach generally leads to noisy images and coherency issues when animating the light or specular objects, particularly when the number of photons used (i.e., light-space image resolution) is small. The rest of this section describes the two techniques we examined for rendering photons, improvements on each, and discusses various tradeoffs inherent in the choice used to render.

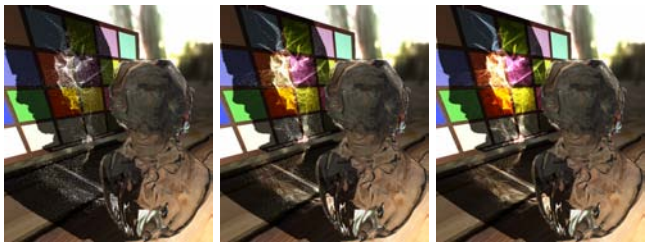

Figure 4: *Direct visualization of (left to right)* $512^2$, $1024^2$, *or* $2048^2$ *photons results in noisy images, particularly noticeable during animation, even for relatively dense photon samplings. Furthermore, using fewer photons results in each representing more energy, often causing pixels to burn out.*

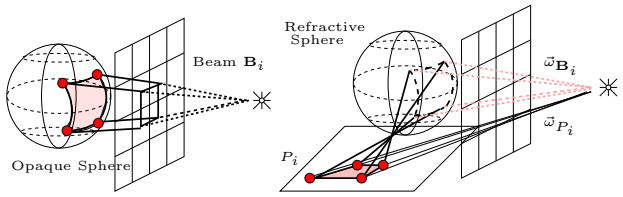


Figure 5: *Light-space pixels represents photons (dots), and can be connected into caustic polygons* $P_i$ *that approximate the filled regions. The beam* $\mathbf{B_i}$ *covering polygon* $P_i$ *contains a portion of the light's energy proportional to the solid angle* $\vec{\omega}_{\mathbf{B_i}}$ *it subtends. The polygon* $P_i$ *on the right would originally (without caustics) have received energy proportional to the solid angle* $\vec{\omega}_{\mathbf{P_i}}$ *it subtends but now receives that plus the energy from beam* $\mathbf{B_i}$

### 3.2.1 Caustic Rendering Using Quads or Triangles

The major problem with rendering photons as simple points is that photon energy is gathered into a single pixel instead of being spread over some larger finite area. This problem becomes worse when using fewer photons, as each carries a higher energy that often burns out the color as seen from the eye.

Our first, most straightforward approach solves this problem by rendering caustic polygons formed by connecting photons adjacent to each other in the light-space photon buffer. This is similar to the techniques of Watt [1990], Nishita and Nakamae [1994] and Ernst et al. [2005], which trace "beams" of light by connecting rays through adjacent vertices, except we determine beams based on image-space adjacencies rather than object-space ones.

To compute the intensity inside a caustic polygon $P_i$, we must consider the initial energy of the beam $\mathbf{B_i}$ (see Figure 5) bounded by the set of adjacent light-space pixels:

$$E_{\mathbf{B_i}} = \frac{\vec{\omega}_{\mathbf{B_i}}}{4\pi} E_L, \tag{1}$$

where $\vec{\omega}_{\mathbf{B_i}}$ is the solid angle subtended by the beam, as seen from the light, and $E_L$ is the total energy emitted by the light source. Note that this step is important, as the pixels in our image plane cover different solid angles. If photons could be emitted uniformly then $E_{\mathbf{B_i}} = \frac{1}{\#photons} E_L$, though for some interactive applications, such an approximation may still prove acceptable.

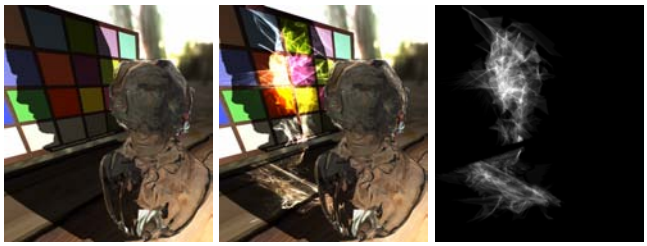

Figure 6: *The Beethoven scene without caustics (left) and with caustics (center). Note that these additional polygons can consume substantial fill-rate. The right image shows the depth complexity of the caustic polygons.*

This beam gets distorted due to specularities before reaching its final destination at the caustic polygon, defined by adjacent points in our photon buffer. Since interactive applications built on APIs such as OpenGL and DirectX often use arbitrary units for light intensity, the important quantity is the ratio $R$ of beam energy $E_{\mathbf{B_i}}$ to the energy incident on that polygon $E_{P_i}$ in absence of the specular object:

$$E_{P_i} = \frac{\vec{\omega}_{P_i}}{4\pi} E_L, \tag{2}$$

where $\vec{\omega}_{P_i}$ is the solid angle $P_i$ subtends as seen from the light. Thus our ratio is simply $R = \frac{\vec{\omega}_{B_i}}{\vec{\omega}_{P_i}}$, the ratio of solid angles. The beam solid angles $\vec{\omega}_{\mathbf{B_i}}$ can easily be precomputed for a specific resolution photon buffer, and computations for $\vec{\omega}_{P_i}$ can be performed in a relatively complex shader. We chose to approximate $\vec{\omega}_{P_i}$ by projecting the polygon vertices onto the unit sphere and computing the area of the projected polygon, which accurately approximates the solid angle due to their relatively small size. If the standard lighting computation $\mathbb{L}(\vec{V}, \vec{N}, \vec{L})$ takes a viewing vector $\vec{V}$, surface normal $\vec{N}$, and light direction $\vec{L}$ then the intensity of the caustic polygon $I_{P_i}$ can be computed (either once per polygon or per vertex) as:

$$\begin{aligned} I_{P_i} &= \mathbb{L}(\vec{V}, \vec{N}, \vec{L}_{direct}) + R * \mathbb{L}(\vec{V}, \vec{N}, \vec{L}_{photon}) \\ &= \mathbb{L}(\vec{V}, \vec{N}, \vec{L}_{direct}) + \frac{\vec{\omega}_{\mathbf{B_i}}}{\vec{\omega}_{P_i}} \mathbb{L}(\vec{V}, \vec{N}, \vec{L}_{photon}) \quad (3) \\ &\approx (1 + R) * \mathbb{L}(\vec{V}, \vec{N}, \vec{L_{direct}}) \quad (4) \end{aligned}$$

where $\vec{L}_{direct}$ is the direction from $P_i$ to the light, and $\vec{L}_{photon}$ is the direction the photons of $\mathbf{B_i}$ arrive at the polygon. To avoid storing the incident directions, Equation 4 often provides a reasonable approximation.

The resulting polygons can easily be alpha blended into an eye-space view of the scene rendered without caustics, shown in Figure 6. Caustic polygons that stretch between different surfaces can either be culled or treated the same as other polygons, as their intensity contributions are quite small. As in shadow volumes [Crow 1977], the fill rate consumed by these additional polygons can significantly reduce rendering performance. Another approach renders these caustic polygons into a light space buffer, analogous to a shadow map, that is later projected onto the environment to determine light intensity.

Rendering into a light-space map increases performance, as these polygons need not be rendered when the viewpoint changes. Projecting a light-space caustic map onto the scene, however, causes caustic aliasing based on the buffer resolution, just as in shadow mapping.

Another optimization only renders caustic polygons where photons encountered a specular surface. In Figure 1c, each set of adjacent pixels in the photon buffer represent one polygon, but many photons directly hit the background geometry or environment map. In these regions no caustics exist, so standard OpenGL lighting is accurate and much more efficient.

#### 3.2.2 Caustic Rendering Via Nearby Neighbor Gathering

The second alternative is based off counting photons rather than drawing and blending the contributions of multiple polygons. The motivation for this approach comes from techniques such as illumination mapping [Arvo 1986] and photon mapping [Jensen 2001]. Illumination mapping increments the count in a texture based on final photon location, and photon mapping stores photons in a 3D kd-tree. Neither the increment operation nor the kd-tree seamlessly map to the GPU, though Purcell et al. [2003] perform non-interactive GPU-based photon mapping using a regular grid.

We suggest avoiding both object-space texture increments and complex 3D data structures by utilizing an image-space photon count. This has an advantage over the polygon rendering method of Section 3.2.1, as each pixel is considered only once, potentially reducing the fill rate considerably. Using a stencil operation to confine this gather to regions with visible photons could improve our rendering speeds further.

The rationale is simple. Dachsbacher and Stamminger [2005] noted that points nearby in world-space tend to clump together in image-space and neighboring pixels in image-space often prove close in world-space. Thus a nearest neighbor search in image-space should return results similar to those of a 3D nearest neighbor gather.

Two possibilities also exist for this gather operation, it can be performed either in a light- or eye-space image. As when rendering caustic polygons, the tradeoff is aliasing due to light-space texture resolution versus speed gained by avoiding the gather when only the viewpoint changes.

This image-space gather approach proceeds as follows:

1. Render photons (from Figure 1c) into a eye- or light-space buffer.
2. Perform a pass that counts contributions of photons in neighboring texels.
3. Render the final scene, adding lighting contributions from step 2.

In the case of the eye-space gather, steps 2 and 3 can be combined. Combining steps 1 and 2 using splatting should improve performance, though we did not implement this approach.

Consider the situation shown in Figure 7. Each pixel $\mathbf{p}_i$ in step 2 above will contain a number of photons $\alpha_j$ that hit the corresponding region $A_{\mathbf{p}_i}$, and the final color of $\mathbf{p}_i$ depends on the photons arriving in $A_{\mathbf{p}_i}$ and nearby regions.

Assuming that the object visible in pixel $\mathbf{p}_i$ is a distance

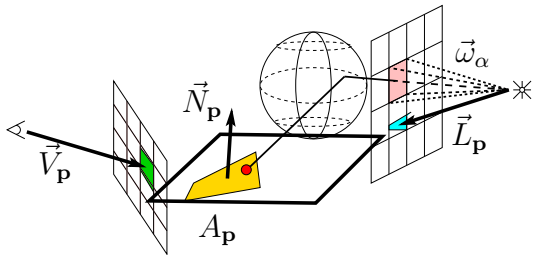


Figure 7: *Each pixel* $\mathbf{p}$*, as seen from the viewer, has some viewing direction* $\vec{V}_\mathbf{p}$*, normal* $\vec{N}_\mathbf{p}$*, direction to the light* $\vec{L}_\mathbf{p}$*, and represents a finite area* $A_\mathbf{p}$*. This pixel subtends a solid angle of* $\vec{\omega}^V_\mathbf{p}$ *(green region) from the viewer and* $\vec{\omega}^L_\mathbf{p}$ *(blue region) from the light. In this case, a photon* $\alpha$ *emitted by the light has been refracted so it contributes to pixel* $\mathbf{p}$*.*

from the viewer of $d^V_{\mathbf{p}_i}$, we can approximate $A_{\mathbf{p}_i}$ by:

$$A_{\mathbf{p}_i} \approx \frac{\vec{\omega}^V_{\mathbf{p}_i}(d^V_{\mathbf{p}_i})^2}{-\vec{V}_{\mathbf{p}_i} \cdot \vec{N}_{\mathbf{p}_i}}, \tag{5}$$

where $\vec{\omega}^V_{\mathbf{p}_i}$ is the solid angle the pixel subtends from the viewpoint, $\vec{V}_{\mathbf{p}_i}$ is the direction from the eye through the pixel, and $\vec{N}_{\mathbf{p}_i}$ is the normal at the visible surface. Given this area, we can perform a similar computation to determine the solid angle $\vec{\omega}^L_{\mathbf{p}_i}$ subtended as seen from the light:

$$\begin{aligned} \vec{\omega}^L_{\mathbf{p}_i} &\approx \frac{A_{\mathbf{p}_i}(-\vec{L}_{\mathbf{p}_i} \cdot \vec{N}_{\mathbf{p}_i})}{(d^L_{\mathbf{p}_i})^2} \\ &\approx \frac{\vec{\omega}^V_{\mathbf{p}_i}(d^V_{\mathbf{p}_i})^2(-\vec{L}_{\mathbf{p}_i} \cdot \vec{N}_{\mathbf{p}_i})}{(d^L_{\mathbf{p}_i})^2(-\vec{V}_{\mathbf{p}_i} \cdot \vec{N}_{\mathbf{p}_i})} \end{aligned} \tag{6}$$

where $d^L_{\mathbf{p}_i}$ is the distance from the light to the surface visible in pixel $\mathbf{p}_i$, and $\vec{L}_{\mathbf{p}_i}$ is the incident direction of the light.

Again, since interactive applications often use arbitrary values for light intensities, the ratio of the energy $E_{\alpha_j}$ carried by photon $\alpha_j$ to the energy $E_{\mathbf{p}_i}$ arriving at $A_{\mathbf{p}_i}$ in absence of the specular object is more important than absolute energy. As in the polygonal case, $E_{\alpha_j}$ depends on the solid angle $\vec{\omega}_{\alpha_j}$ subtended by the photon:

$$E_{\alpha_j} = \frac{\vec{\omega}_{\alpha_j}}{4\pi} E_L. \tag{7}$$

Similarly, the energy directly reaching $A_{\mathbf{p}_i}$ in absence of the focal object depends on the solid angle it subtends from the light's viewpoint:

$$E_{\mathbf{p}_i} = \frac{\vec{\omega}^L_{\mathbf{p}_i}}{4\pi} E_L \approx \frac{\vec{\omega}^V_{\mathbf{p}_i}(d^V_{\mathbf{p}_i})^2(-\vec{L}_{\mathbf{p}_i} \cdot \vec{N}_{\mathbf{p}_i})}{4\pi(d^L_{\mathbf{p}_i})^2(-\vec{V}_{\mathbf{p}_i} \cdot \vec{N}_{\mathbf{p}_i})} E_L \tag{8}$$

The solid angles $\vec{\omega}_{\alpha_j}$ and $\vec{\omega}^V_{\mathbf{p}_i}$ can be precomputed for specific resolutions of the caustic and view images, respectively. The direction from the light to the pixel $\vec{L}_{\mathbf{p}_i}$ and the distance $d^L_{\mathbf{p}_i}$ can be computed from the photon locations stored in the photon buffer (Figure 1c), and the normal $\vec{N}_{\mathbf{p}_i}$ is passed to OpenGL during rendering. The values of $\vec{V}_{\mathbf{p}_i}$ and $d^V_{\mathbf{p}_i}$ are determined during the eye-space gather.

If the photon gather occurs in light-space instead of eye-space, Equation 8 is unnecessary, as $\vec{\omega}^L_{\mathbf{p}_i}$ can be precomputed and used directly. In either case, the intensity computation at each pixel $I_{\mathbf{p}_i}$ proceeds similar to Equation 3:

$$I_{\mathbf{p}_i} = \mathbb{L}(\vec{V}_{\mathbf{p}_i}, \vec{N}_{\mathbf{p}_i}, \vec{L}_{\mathbf{p}_i}) + \sum_{\alpha_j \in A_{\mathbf{p}_i}} \frac{E_{\alpha_j}}{E_{\mathbf{p}_i}} \mathbb{L}(\vec{V}_{\mathbf{p}_i}, \vec{N}_{\mathbf{p}_i}, \vec{L}_{\alpha_j}) \tag{9}$$

$$\approx \mathbb{L}(\vec{V}_{\mathbf{p}_i}, \vec{N}_{\mathbf{p}_i}, \vec{L}_{\mathbf{p}_i}) + \mathbb{L}(\vec{V}_{\mathbf{p}_i}, \vec{N}_{\mathbf{p}_i}, \vec{L}^{avg}_{\alpha_i}) \sum_{\alpha_j \in A_{\mathbf{p}_i}} \frac{E_{\alpha_j}}{E_{\mathbf{p}_i}} \tag{10}$$

$$\approx \left(1 + \sum_{\alpha_j \in A_{\mathbf{p}_i}} \frac{E_{\alpha_j}}{E_{\mathbf{p}_i}}\right) \mathbb{L}(\vec{V}_{\mathbf{p}_i}, \vec{N}_{\mathbf{p}_i}, \vec{L}_{\mathbf{p}_i}), \tag{11}$$

where $\mathbb{L}(\vec{V}, \vec{N}, \vec{L})$ again represents the standard lighting computation given a viewing direction $\vec{V}$, surface normal $\vec{N}$, and direction to the light $\vec{L}$. The photon $\alpha_j$ is incident upon the region $A_{\mathbf{p}_i}$ from direction $\vec{L}_{\alpha_j}$, which can be stored when shooting photons (as in Figure 3b). To avoid storing the incident directions for each photon, the approximations in Equations 10 and 11 can be used. Equation 10 uses an average incident photon direction $\vec{L}^{avg}_{\alpha_i}$ for each pixel, whereas Equation 11 approximates the direction without any additional storage requirements. We use Equation 10.

As described so far, Equation 9 computes the intensity for photons hitting the surface inside a single pixel, so it leads to the results shown in Figure 4. Ideally, we should consider photons hitting nearby pixels as well. To handle photons in surrounding regions, our gather pass (step 2 from the previous page) proceeds as follows:

- For each pixel:
  1. Search the $7 \times 7$ pixel neighborhood.
  2. Directly compute the area $A_\mathbf{p}$ covered by nearby pixels (using the world-space locations that map to neighboring pixels), instead of via Equation 5.
  3. Perform the lighting operation $\mathbb{L}(\vec{V}_{\mathbf{p}_i}, \vec{N}_{\mathbf{p}_i}, \vec{L}^{avg}_{\alpha_i})$ from Equation 10 once for each neighbor.
  4. Weight the neighbor contributions using a Gaussian (reported by Jensen [2001] and others [Russ 2002] to work well for photon counting) and add in direct lighting.
  5. To reduce noise, discard contributions where fewer than three photons hit the neighborhood.

#### 3.2.3 Filtering for Noise Reduction and Improved Coherency

A number of problems exist in both the techniques above. First, even with relatively dense $2048^2$ light-space samplings, noise and other artifacts can still appear. While Jensen [2001] can reduce noise by discarding photon gathers averaging fewer than nine photons, discarding similar gathers with less dense sampling eliminates important parts of the caustic. Furthermore, Moiré artifacts can arise due to our regular sampling.

Both approaches lead to problems during animation. Neither approach enforces frame-to-frame coherency, which gives rise to typical popping artifacts. To maintain coherency in our gathers, we store photon counts from the previous three frames and utilize them in conjunction with

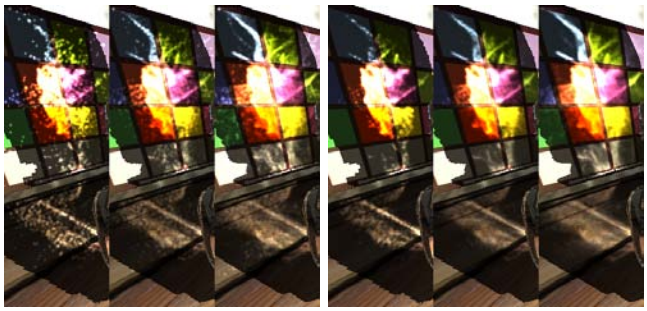

Figure 8: *Effects of temporally filtering caustics. (Left) Unfiltered caustics from the Beethoven bust with* $256^2$*,* $512^2$*, and* $1024^2$ *photons. (Right) Filtered caustics using* $256^2$*,* $512^2$*, and* $1024^2$ *photons per frame.*

photons from the current frame (see Figure 8). The additional data allows us to filter additional noise without eliminating valid caustic regions, helps reduce regular sampling artifacts, and maintains coherency across frames. Obviously fast moving objects and lights can cause ghosting artifacts, though an adaptive filter length could be used in such circumstances.

Note that we only store photon data (e.g., hit locations, incident directions, attenuation) for the *current* frame. We assume reusing this data for photons from all four frames provides a reasonable approximation; again, quickly moving lights and geometry break this assumption, probably calling for an adaptive-length filter. Note that we did not implement temporal filtering for the caustic polygons of Section 3.2.2, but it should provide similar benefits.

## 4 Results and Discussion

We implemented these techniques in OpenGL on an nVidia GeForce 7800 GTX with 256 MB memory. Our shaders were compiled using Cg's `vp40` and `fp40` profiles. For refraction, we build off the work of Wyman [2005a; 2005b] to determine where photons hit the background geometry. For reflection, we reflect about the surface normal and perform a ray-plane intersection with a single background plane. Again note that the environment maps do not contribute caustics in our results; caustics are generated from a single point light source in the scene.

The timings cited in Table 1 come from a 3.2 GHz Pentium 4 Xeon with 2 GB of memory. We optimized for display of intermediate results rather than speed, resulting in additional rendering passes of background geometry. We use `EXT_framebuffer_object` with multiple 16-bit floating point buffers storing photon location, incident directions, and final scene color (to allow additive alpha blending of small photon contributions). An optimized implementation could eliminate many of our intermediate buffers and reduce the bit-precision on others.

Figures 1, 9, 10, 11, and 12 show results of our technique on a variety of geometry, both with complex and simple backgrounds. Figure 9 shows examples of reflective caustics, and Figure 10 shows the caustic cast by a simple, unsubdivided octahedron using both a metal and glass material. Figure 11 compares our results to photon-mapping, and Figure 12 shows complex background geometry. Our results differ from photon mapping mainly because we use a static search radius, causing blurriness in areas of high photon density and noise in low density regions. Additionally, our use of per-pixel (instead of per-photon) incident directions can cause final intensity values to vary slightly.

Figure 9: *Reflective caustics from a metal ring and dragon.*

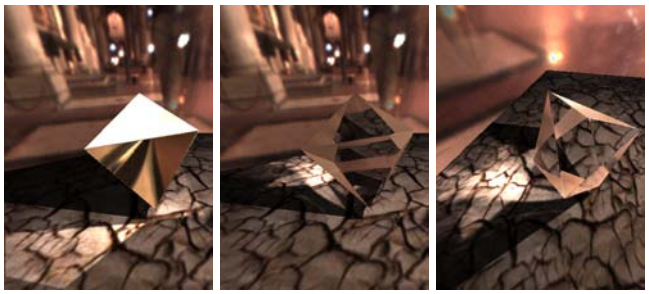

Figure 10: *An octahedron shown both (left) reflective and (center) refractive. Rotating around (right) shows the refractive caustic more clearly.*

Just as in stochastic renderers, the number of photons required increases as geometry becomes more complex and the caustic more intricate. In the case or reflective objects where light diverges, instead of converging into a caustic, our technique requires significant numbers of samples to eliminate artifacts. In the ring video, for instance, Moiré patterns are evident in the reflections from the ring exterior even with $1024^2$ photons and temporal filtering.

A number of features and issues become apparent when analyzing the timings. While our techniques are independent of scene complexity, the reflection and refraction techniques depend on model size, particularly when using complex background geometry. Using $2048^2$ photons, however, all scenes run at roughly the same speed. While we render the points (and quads) from the eye's viewpoint each frame, the photon buffer does not need rerendering under static lighting. This render-to-vertex-array step appears quite slow, so examining other approaches to access photon data (such as vertex shader texture accesses) might significantly improve performance.

Additionally, we observe a large drop in performance of complex scenes as we move from $1024^2$ to $2048^2$ even when simply visualizing photons as points. This suggests that texture memory swapping occurs between rendering steps at high sampling rates, so optimizing texture usage could significantly improve performance. Another anomaly oc-

| Scene | # Photons | Static Lighting | | | | Dynamic Lighting | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Points | Quads | $\text{Gather}_{eye}$ | $\text{Gather}_{light}$ | Points | Quads | $\text{Gather}_{eye}$ | $\text{Gather}_{light}$ |
| Ball & Dragon | $512^2$ | 73.2 | 30.6 | 38.4 | 41.8 | 27.1 | 1.6 | 19.2 | 17.0 |
| (251,000 triangles) | $1024^2$ | 40.2 | 10.3 | 27.5 | 41.5 | 10.5 | 0.4 | 11.4 | 7.0 |
| | $2048^2$ | 3.9 | n/a | 13.1 | 41.3 | 2.7 | n/a | 4.4 | 2.5 |
| Beethoven | $512^2$ | 123.1 | 29.7 | 47.7 | 53.5 | 35.8 | 1.5 | 30.9 | 17.5 |
| (5000 triangles) | $1024^2$ | 52.8 | 9.7 | 32.7 | 52.8 | 12.1 | 0.4 | 14.3 | 8.1 |
| | $2048^2$ | 17.9 | n/a | 14.5 | 52.5 | 2.7 | n/a | 4.7 | 2.6 |
| Metal Ring | $512^2$ | 151.2 | 39.6 | 52.8 | 59.3 | 40.3 | 1.5 | 34.7 | 18.9 |
| (6300 triangles) | $1024^2$ | 57.6 | 11.7 | 35.4 | 59.0 | 12.5 | 0.4 | 16.0 | 8.4 |
| | $2048^2$ | 18.8 | n/a | 15.2 | 58.3 | 2.8 | n/a | 5.0 | 2.6 |
| Metal Dragon | $512^2$ | 98.3 | 17.2 | 43.9 | 48.5 | 32.1 | 1.5 | 26.6 | 16.1 |
| (250,000 triangles) | $1024^2$ | 46.0 | 9.0 | 31.0 | 48.4 | 11.3 | 0.4 | 14.2 | 7.8 |
| | $2048^2$ | 4.4 | n/a | 14.2 | 48.0 | 2.8 | n/a | 4.9 | 2.7 |
| Octahedron | $512^2$ | 159.1 | 38.1 | 53.9 | 61.1 | 38.9 | 1.5 | 33.7 | 18.8 |
| (10 triangles) | $1024^2$ | 58.4 | 11.4 | 35.1 | 60.8 | 12.4 | 0.4 | 15.3 | 8.3 |
| | $2048^2$ | 18.1 | n/a | 14.7 | 60.5 | 2.7 | n/a | 4.8 | 2.6 |
| Two Dragons | $512^2$ | 41.4 | 12.3 | 27.5 | 29.4 | 14.5 | 1.4 | 13.3 | 10.1 |
| (500,000 triangles) | $1024^2$ | 28.6 | 5.5 | 21.7 | 29.3 | 7.7 | 0.4 | 8.1 | 5.6 |
| | $2048^2$ | 4.3 | n/a | 11.7 | 29.2 | 2.3 | n/a | 3.4 | 2.2 |

Table 1: *A sampling of timings for scenes used in the paper and accompanying video. Values are frames per second using a* $512^2$ *image resolution, with the light emitting either* $512^2$*,* $1024^2$*, or* $2048^2$ *photons in a* $90°$ *field-of-view. Caustic rendering was performed by rendering points or quads each frame, doing an eye-space nearest-neighbor gather each frame, or gathering in light-space whenever the lighting changed. Our implementation was unable to perform caustic polygon rendering at* $2048^2$*.*

curs using higher photon counts, where the eye-space gather runs quicker than a naive photon visualization. This occurs because we do not compute lighting contributions of each photon when gathering, but rather perform a single lighting computation for each pixel in the neighborhood.

Based on the timings in Table 1, it is obvious that for static lighting a single light-space gather as a projective texture gives the best performance. However, if aliasing and interpolation between texels is problematic, an eye-space gather still performs reasonably in static lighting environments. In dynamic lighting environments the eye-space gather performs better than the light-space gather due to the expense of an added light-space pass and memory buffer.

## 5 Conclusions and Future Work

This paper presented an adaption of the traditional two-pass approach for rendering caustics to run interactively on current graphics accelerators. Our technique is physically based, though many steps in the process can be approximated to reduce the required resources. Photon emission is simulated by storing the locations visible when rendering from the light's viewpoint. Photon gathering can be performed directly by drawing caustic polygons or by gathering photons using an image-space nearest neighbor search. This second option can occur in either eye- or light-space. We also showed that Gaussian filtering over the image-space neighborhood and temporal filtering over recent frames can significantly reduce noise and increase coherency.

An advantage of our technique over vertex-tracing techniques [Ernst et al. 2005; Shah and Pattanaik 2005] is that our models need not be finely tessellated to get high-quality caustics and our technique is relatively independent of object complexity. However since we render photons in a rectangular buffer, not all of them intersect the specular surface, so we waste some computations. Unlike Wand and Straßer [2003], we cannot render caustics from environment maps, but we seamlessly integrate more complex reflection and refraction models without adding the additional passes required by their dense object sampling.

Our interactive technique is limited to point light sources, though it could help accelerate photon emission from more complex sources. Additionally, the current approach does not consider volumetric scattering effects. Divergent "caustics," such as those from a metal ball, also prove problematic for our image-space gathering techniques as our neighborhoods do not dynamically expand to examine enough photons. Finally, as we build on interactive techniques for rendering reflections and refractions, our technique inherits their limitations.

Some interesting areas for future work include applying shadow mapping variants (e.g., adaptive shadow maps [Fernando et al. 2001] and perspective shadow maps [Stamminger and Drettakis 2002]) to focus computation on specular geometry, filtering techniques to further reduce noise and improve coherency, creating efficient data structures to hold photon data (e.g., Ma and McCool [2002]), and optimizing the pipeline to avoid vertex and fragment processing on uninteresting photons.

## References


Arvo, J. 1986. Backward ray tracing. *Developments in Ray Tracing*, 259–263. ACM Siggraph '86 Course Notes.

Crow, F. 1977. Shadow algorithms for computer graphics. In *Proceedings of SIGGRAPH*, 242–248.

Dachsbacher, C., and Stamminger, M. 2005. Reflective shadow maps. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 203–208.

Diefenbach, P., and Badler, N. 1997. Multi-pass pipeline rendering: Realism for dynamic environments. In *Proceedings of the Symposium on Interactive 3D Graphics*, 59–70.

Ernst, M., Akenine-Möller, T., and Jensen, H. W. 2005. Interactive rendering of caustics using interpolated warped volumes. In *Proceedings of Graphics Interface*, 87–96.

Fernando, R., Fernandez, S., Bala, K., and Greenberg, D. 2001. Adaptive shadow maps. In *Proceedings of SIGGRAPH*, 387–390.

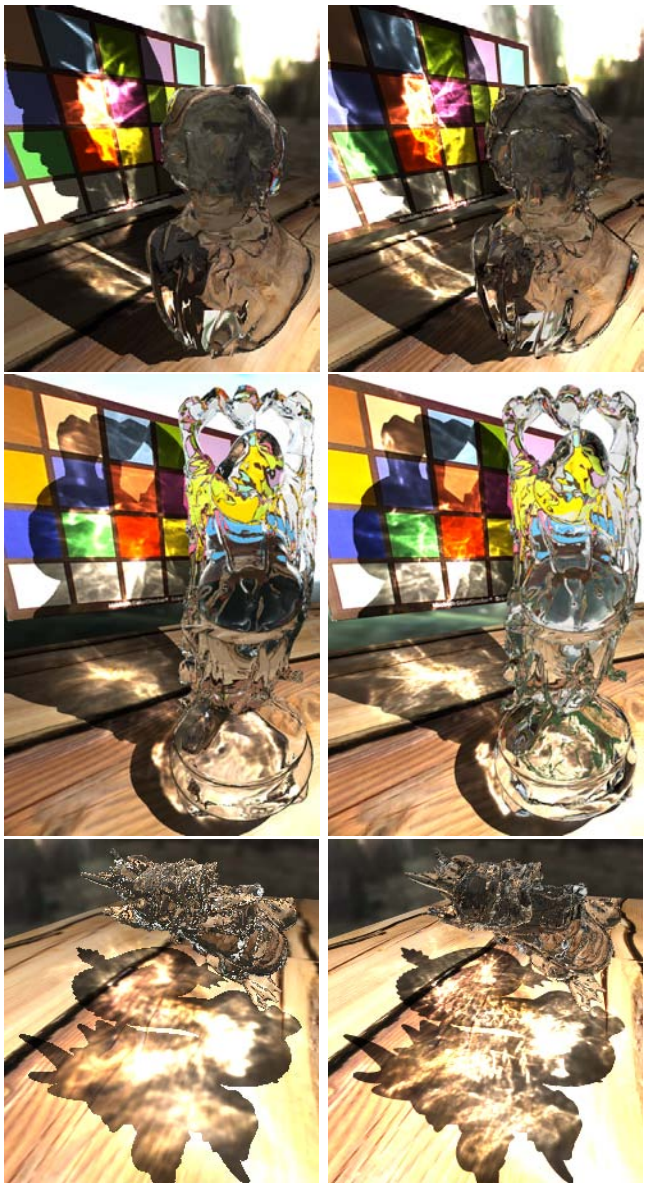

Figure 11: *Our results are on the left, with photon mapped comparisons on the right. The Beethoven bust uses* $512^2$ *photons and a light-space gather, the Buddha uses* $1024^2$ *photons and a light-space gather, and the Dragon uses* $2048^2$ *photons and a eye-space gather. In these cases, roughly* $\frac{1}{4}$ *to* $\frac{1}{3}$ *of the photons hit the refractor. For all three photon maps* $10^6$ *photons intersect the refracting geometry.*

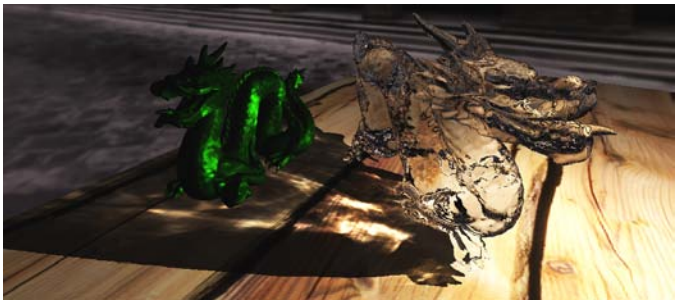

Figure 12: One dragon focuses light onto another dragon.


Günther, J., Wald, I., and Slusallek, P. 2004. Realtime caustics using distributed photon mapping. In *Proceedings of the Eurographics Symposium on Rendering*, 111–121.

Heckbert, P. S., and Hanrahan, P. 1984. Beam tracing polygonal objects. In *Proceedings of SIGGRAPH*, 119–127.

Iwasaki, K., Dobashi, Y., and Nishita, T. 2002. An efficient method for rendering underwater optical effects using graphics hardware. *Computer Graphics Forum 21*, 4 (November), 701–711.

Jensen, H. W. 2001. *Realistic Image Synthesis Using Photon Mapping*. AK Peters.

Kajiya, J. 1986. The rendering equation. In *Proceedings of SIGGRAPH*, 143–150.

Kay, D. S., and Greenberg, D. 1979. Transparency for computer synthesized images. In *Proceedings of SIGGRAPH*, 158–164.

Ma, V., and McCool, M. 2002. Low latency photon mapping using block hashing. In *Proceedings of Graphics Hardware*, 89–99.

Nishita, T., and Nakamae, E. 1994. Method of displaying optical effects within water using accumulation buffer. In *Proceedings of SIGGRAPH*, 373–381.

Ofek, E., and Rappoport, A. 1998. Interactive reflections on curved objects. In *Proceedings of SIGGRAPH*, 333–342.

Ohbuchi, E. 2003. A real-time refraction renderer for volume objects using a polygon-rendering scheme. In *Proceedings of Computer Graphics International*, 190–195.

Parker, S., Martin, W., Sloan, P.-P., Shirley, P., Smits, B., and Hansen, C. 1999. Interactive ray tracing. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*, 119–126.

Purcell, T. J., Buck, I., Mark, W. R., and Hanrahan, P. 2002. Ray tracing on programmable graphics hardware. *ACM Transactions of Graphics 21*, 4, 703–712.

Purcell, T., Donner, C., Cammarano, M., Jensen, H. W., and Hanrahan, P. 2003. Photon mapping on programmable graphics hardware. In *Proceedings of Graphics Hardware*, 41–50.

Russ, J. C. 2002. *The Image Processing Handbook*, fourth edition ed. CRC Oress, Boca Raton, Florida.

Schmidt, C. M. 2003. *Simulating Refraction Using Geometric Transforms*. Master's thesis, Computer Science Department, University of Utah.

Shah, M., and Pattanaik, S. 2005. Caustics mapping: An image-space technique for real-time caustics. Tech. Rep. CS-TR-50-07, University of Central Florida, August.

Stamminger, M., and Drettakis, G. 2002. Perspective shadow maps. In *Proceedings of SIGGRAPH*, 557–562.

Veach, E., and Guibas, L. J. 1997. Metropolis light transport. In *Proceedings of SIGGRAPH*, 65–76.

Wald, I., Kollig, T., Benthin, C., Keller, A., and Slusallek, P. 2002. Interactive global illumination using fast ray tracing. In *Proceedings of the Eurographics Rendering Workshop*, 15–24.

Wand, M., and Strasser, W. 2003. Real-time caustics. *Computer Graphics Forum 22*, 3, 611–620.

Watt, M. 1990. Light-water interaction using backward beam tracing. In *Proceedings of SIGGRAPH*, 377–385.

Woop, S., Schmittler, J., and Slusallek, P. 2005. Rpu: a programmable ray processing unit for realtime ray tracing. *ACM Transactions on Graphics 24*, 3, 434–444.

Wyman, C., Hansen, C., and Shirley, P. 2004. Interactive caustics using local precomputed irradiance. In *Proceedings of the Pacific Conference on Computer Graphics and Applications*, 143–151.

Wyman, C. 2005. An approximate image-space approach for interactive refraction. *ACM Transactions on Graphics 24*, 3 (July), 1050–1053.

Wyman, C. 2005. Interactive image-space refraction of nearby geometry. In *Proceedings of GRAPHITE. (to appear)*.

Yu, J., Yang, J., and McMillan, L. 2005. Real-time reflection mapping with parallax. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 133–138.

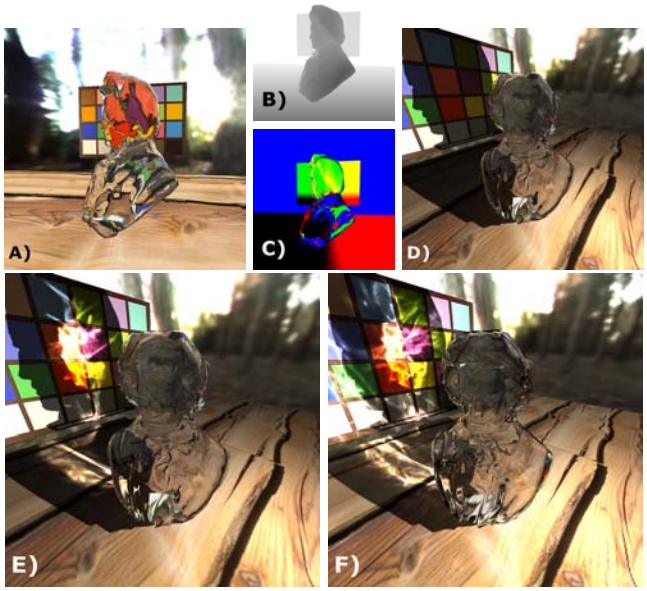


Figure 1: *Our approach to interactive caustics builds off work for interactive reflections and refractions. The first pass renders (a) the view from the light, (b) a depth map for shadow mapping, and (c) final photon locations into buffers. The second pass gathers the photons and (e) renders a result from the eye's point of view. The result significantly improves on (d) existing interactive renderings, and compares favorably with (f) photon mapping.*

Figure 9: *Reflective caustics from a metal ring and dragon.*

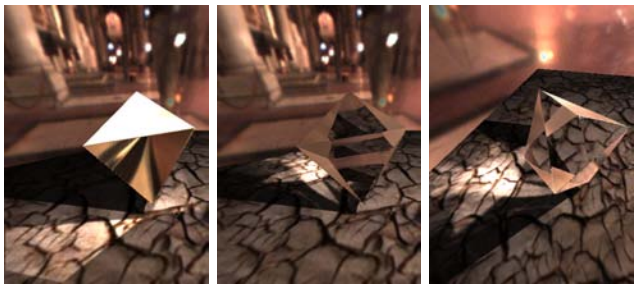

Figure 10: *An octahedron shown both (left) reflective and (center) refractive. Rotating around (right) shows the refractive caustic more clearly.*

Figure 11: *Our results are on the left, with photon mapped comparisons on the right. The Beethoven bust uses* $512^2$ *photons and a light-space gather, the Buddha uses* $1024^2$ *photons and a light-space gather, and the Dragon uses* $2048^2$ *photons and a eye-space gather. In these cases, roughly* $\frac{1}{4}$ *to* $\frac{1}{3}$ *of the photons hit the refractor. For all three photon maps* $10^6$ *photons intersect the refracting geometry.*

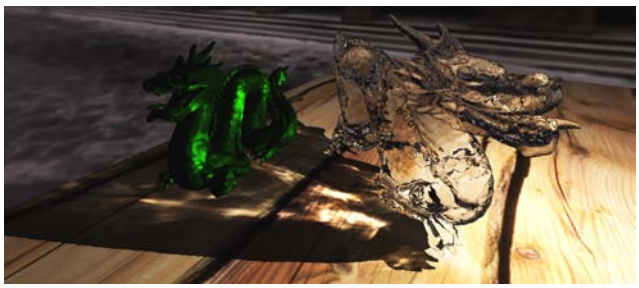

Figure 12: One dragon focuses light onto another dragon.