Hierarchical Caustic Maps

Chris Wyman[*]
University of Iowa

Abstract

Interactive applications typically rely on local models for lighting, occasionally augmented by GPU-friendly methods for approximating global illumination. *Caustic mapping* approximates the specular focusing of light using a light-space image, akin to a shadow map, which is projected onto the scene during final rendering. Unfortunately, existing caustic map implementations must choose between quality and speed. Quickly generated maps use few photons and look extremely blurry, while sharper maps created from millions of photons only render at a few frames per second. This paper introduces a number of hierarchical enhancements to caustic mapping that allow real-time rendering with high quality caustic maps, even when using maps from multiple light sources. These techniques utilize the geometry processing stage of recent GPUs to avoid processing every photon and to render a pyramidal caustic map that allows photon splats of varying diameters without the increased costs inherent in rasterizing large splats.

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

Keywords: interactive rendering, caustics, hardware

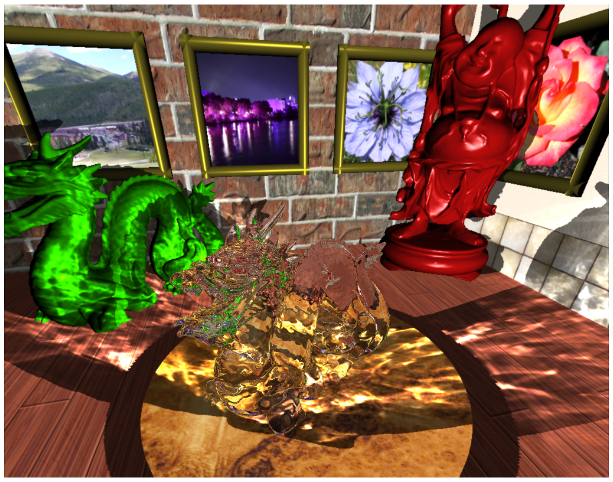

Figure 1: *This scene contains 560k triangles and two lights emitting 4 million photons each, yet hierarchical caustic maps still remain interactive, running at 9 fps for a resolution of* 2048^2.

1 Introduction

While speed takes precedence over realism in most interactive applications, the desire to improve graphics quality has motivated the development of various fast approximations for realistic lighting. Simple shadows, now common in such applications, are typically rendered using some variant of shadow mapping [Williams 1978] or shadow volumes [Crow 1977], and various methods allow shadows from larger area lights [HasenFratz et al. 2003]. Diffuse global illumination can be stored in textures, approximated via ambient occlusion [Bunnel 2005], or dynamically computed with precomputed radiance transfer [Sloan et al. 2002; Ng et al. 2004] or splatting techniques [Dachsbacher and Stamminger 2006].

One challenging problem has been rendering perfectly specular materials. For such materials, specific geometry is reflected or refracted at each surface point, so approximations that boost speed by blurring are unacceptable. A decade ago, only environment mapped reflections [Greene 1986] and planar specular effects [Diefenbach and Badler 1997] were feasible, but more recent work can approximate more complex reflections [Estalella et al. 2006; Umenhoffer et al. 2007; Yu et al. 2005] and refractions [Oliveira and Brauwers 2007; Davis and Wyman 2007; Wyman 2005].

Building on these interactive approximations for specular materials, *caustic mapping* [Shah et al. 2007; Szirmay-Kalos et al. 2005; Wei and Kaihuai 2007; Wyman and Davis 2006] follows specularly reflected and refracted photons backwards from the light, as in photon mapping [Jensen 2001], storing their hit positions into a photon buffer. A second pass then reorganizes and splats these stored photons into a caustic map, which is projected onto the scene similar to a shadow map. Effectively, shadow maps allow a binary light visibility test at geometry in the scene, whereas caustic maps account for multiple paths from the light to each surface.

Two major problems limit the applicability of caustic mapping, however.

1. Each photon in the photon buffer must be processed, so developers must choose between severely undersampled caustics or a large speed penalty to use enough photons for caustics comparable to offline methods.
2. Complex specular interactions cause oversampling in bright focal regions while other areas of the caustic map remain undersampled and noisy.

This paper introduces a number of hierarchical improvements to existing caustic mapping techniques that address these issues. First, the photon buffer is processed in a hierarchical manner, allowing many irrelevant photons to be discarded simultaneously. Second, during this hierarchical processing, clusters of converging photons are identified and combined into a single splat. This significantly reduces the cost of oversampling focal regions. Finally, using information about the convergence or divergence of neighboring pho-

[*]E-mail: cwyman@cs.uiowa.edu

tons, we use photon splats of varying sizes to reduce undersampling noise. A render-to-mipmap technique virtually eliminates the cost of rasterizing large splats, and also allows photon splats to exceed the hardware point size limits. Section 4 discusses numerous implementation-specific optimizations that significantly improve caustic mapping performance, even without these hierarchical techniques.

2 Previous Work

Graphics researchers have investigated caustics for more than twenty years, starting with Kajiya's [1986] work on the rendering equation. More recent work can, broadly, be split into wavefront, beam, and particle techniques.

Wavefront techniques trace waves of light, i.e., the surfaces perpendicular to light rays, through the scene. Mitchell and Hanrahan [1992] first applied wavefronts to the computation of caustics. Ihrke et al. [2007] precompute a volumetric representation of a specularly deformed wavefront, allowing the interactive rendering of specular effects including complex volumetric caustics. Unfortunately, significant precomputation costs preclude application in dynamic scenes.

Beam techniques arise from the application of beam tracing [Heckbert and Hanrahan 1984] to energy leaving the light source. Watt [1990] traces light beams through a water surface to generate underwater caustics. Nishita and Nakamae [1994] use beams for the same purpose while also accounting for volumetric scattering inside the water, and Iwasakai et al. [2002] outlined a GPU-accelerated underwater caustic technique using beams. Using warped caustic volumes and an improved intensity interpolation, Ernst et al. [2005] further improve caustic quality.

Arvo [1986] suggests emitting rays from light sources and accumulating them in illumination maps. Besides eliminating the need for per-object illumination textures, Jensen's [2001] photon mapping also renders other global illumination effects. As most photon mappers accelerate photon gathering using kD-trees, a difficult data structure to efficiently implement using GPUs [Foley and Sugerman 2005], researchers have only had limited success utilizing photon mapping in an interactive context [Purcell et al. 2003].

Other interactive techniques capture caustics from single reflective interfaces. Wand and Straßr [2003] subdivide reflective surfaces, treating each sample as a point light source and using the GPU to gather energy from each. Dachsbacher and Stamminger [2006] note that surfaces visible in a shadow map reflect light towards other surfaces. By intelligently splatting photons representing a subset of shadow map texels, they approximate indirect illumination including caustics from simple surfaces.

Recent work proposes creating a projective light-space illumination map called a *caustic map* that can be projected onto a scene to render realistic caustics. Researchers concurrently developed a whole class of caustic mapping techniques, which all follow the basic three-step process illustrated in Figure 2. The first step emits photons from the light by rendering the scene from the light position, storing the locations where photons first intersect with diffuse materials into an *photon buffer*. The second pass treats these photons locations as point primitives, transforms them via vertex shaders, and splats them into a caustic map. The third step projects the caustic map onto the scene similar to (and often in conjunction with) a shadow map.

Szimay-Kalos et al. [2005] use a low resolution light image and large Gaussian splats. This very quickly produces very blurry caustic maps. Wyman and Davis [2006] suggest using many more photons and explore a variety of ways to gather photons into the caustic map. Their approaches produce higher fidelity caustics, but at much slower framerates. Wei and Kaihuai [2007] propose using per-object instead of per-light caustic maps, similar in spirit to illumination mapping. Shah et al. [2007] introduce a vertex tracing technique for the first pass, instead of a full-fledged rasterization pass. This can improve performance, but caustic quality becomes dependent on the refractor's tessellation level.

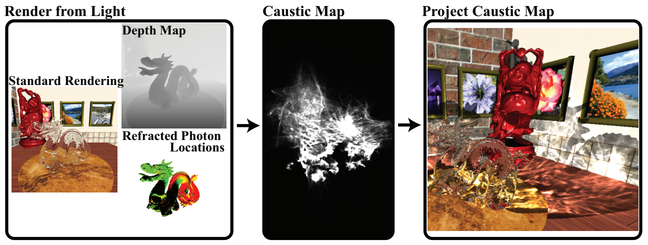

Figure 2: *Caustic mapping is a three-pass process. The first pass renders from the light, creating a* photon buffer *that stores locations where photons are absorbed. The second pass reorganizes these photons into a* caustic map*, which is projected onto the scene in the third pass in conjunction with a shadow map.*

Kruger et al. [2006] improve caustic map quality by using surface-aligned splats and additionally propose a line tracing technique that enables volumetric caustics. Wyman and Dachsbacher [2007] vary splat sizes to reduce undersampling noise when caustic photons diverge. A per-photon thin lens model approximates photon divergence. They further reduce noise by rendering multiple splat sizes and interpolating between them during caustic map projection.

2.1 Caustic Map Limitations

Unfortunately, caustic mapping still has limitations in interactive contexts. Previous work remains real-time only with around 10,000 photons. As only some of these photons interact with specular surfaces to focus into a caustic, the results are extremely blurry. Shah et al.'s vertex tracing method uses exclusively relevant photons, but geometry tessellation affects caustic quality, requiring the use of more complex models. Wyman and Dachsbacher achieve high quality results using a million photons, but with that many photons framerates never exceed 10 Hz.

Even with millions of photons, undersampling and oversampling are evident in the caustic maps. A regular photon sampling, inevitable given that photons are emitted by rasterizing the scene, exacerbates the problem. Varying the splat sizes reduces noise but slows rendering. During animation, this noise becomes particularly objectionable.

3 Hierarchical Caustic Maps

Previous caustic mapping techniques all use a constant number of photons. Either a fixed photon buffer resolution is chosen *a priori* or one photon is emitted per refractor vertex. In either case, once a photon count is fixed all photons are processed throughout the entire caustic map creation process. This leads to three problems:

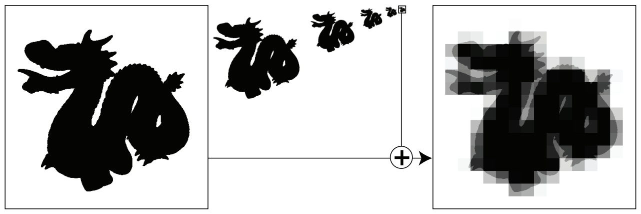

Figure 3: *(Left) Many irrelevant photons (white) are typically processed in caustic mapping. By first transforming a reduced resolution photon buffer, most irrelevant photons are discarded cheaply. (Right) A reduced resolution photon buffer overlaid on the original buffer, showing that most irrelevant photons can be discarded at quite low resolutions in a hierarchical photon buffer.*

1. Wasted computation when processing irrelevant photons (those not encountering a specular surface),
2. Oversampling in regions where photons converge, and
3. Undersampling in regions where photons diverge.

Only the last problem has been addressed previously. Most researchers simply splat photons into the caustic map with a large enough Gaussian splat to eliminate objectionable noise, though this comes at a cost of blurring the caustics. Wyman and Dachsbacher [2007] use a varying splat size to reduce undersampling while maintaining crisp caustics, though their work does not address the other problems and their splat sizes are determined using empirical methods. We address all three of these problems using hierarchical techniques, using hardware generated mipmaps and a render-to-mipmap technique.

3.1 Eliminating Irrelevant Photons

Because photons are generated by rasterizing a light view and treating each pixel as an emitted photon, they are arranged in a regularly sampled 2D buffer. Many photons will not hit a specular surface. Since, by definition, caustics only involve photons that interact with specular surfaces, other photons should ideally be discarded immediately.

One solution would resize the light view each frame to tightly bound the specular object. This significantly reduces the problem, but many extraneous photons still exist for concave objects. Also, a fixed size photon buffer generally allows a tight bound in only one dimension, e.g., for a long narrow object. For many implementations bounding the object will be infeasible, due to a coupling of photon buffer and shadow map view parameters to allow combining their rendering passes. In our implementation the refraction quality also varies slightly with field-of-view, so tight bounds are not desirable.

Existing implementations process all photons via a vertex shader that repositions caustic photons to their appropriate location in the caustic map. Typically, irrelevant photons are moved to locations outside the view frustum, so they are culled prior to rasterization. With the introduction of geometry shaders, these photons can be explicitly discarded before culling. We found this does improve performance, but only by a few percent, as invoking a geometry shader for the extra photons has similar cost to hardware culling.

Since we cannot simply ignore irrelevant photons, our approach attempts to discard them as cheaply as possible. To do this, we observe that irrelevant photons tend to be grouped together. Thus, a hierarchical approach should allow us to discard large groups of unnecessary photons at once. Figure 3 shows the relevant and irrelevant photons (black and white, respectively) from the photon buffer in Figure 2. By creating a mipmap hierarchy and first processing the low resolution buffers, we can discard texels whose children contain no valid photons. Each discarded texel from a 16^2 mipmap level eliminates 1024 unnecessary photons in the corresponding 512^2 buffer. To discard these texels, we use a geometry shader that discards unnecessary photons and streams others back as input into the pipeline.

Note, however, that texels in the low resolution photon buffers are only discarded if *none* of its children are valid photons. This means a texel might contain only a single valid photon, resulting in wasted processing for thousands of others. To reduce this problem, we discard photons in a recursive manner, traversing the mipmap quadtree in a breadth-first manner:

```
PhotonList photonsForLevel[maxMipLevels+1];

// Only one texel in top mipmap level!
photonsForLevel[maxMipLevels].Add( root );

for (i=maxMipLevels down to 1) do
  for all (photons p ∈ photonsForLevel[i]) do
    if ( IsInvalid( p ) ) then
      continue;
    photonsForLevel[i-1].Add( p.Child(0) );
    photonsForLevel[i-1].Add( p.Child(1) );
    photonsForLevel[i-1].Add( p.Child(2) );
    photonsForLevel[i-1].Add( p.Child(3) );

TransformPhotons( photonsForLevel[0] );
```

We found that beyond the first few levels, processing at each mipmap level discards roughly 5% to 30% of the remaining photons. Obviously, earlier levels tend to discard a larger percentage. To avoid processing mipmaps containing only a handful of photons, our implementation starts with a mipmap of resolution 16^2.

3.2 Reducing Oversampling

Oversampling occurs where many photons converge to a single texel in the caustic map (as in Figure 4). Unlike photon mapping, where a k-nearest neighbor search averages photons over an arbitrary region, the smallest neighborhood discernible using caustic mapping is a single caustic map texel. One photon contributing to a texel is treated the same as any other contributing photon, and after noise in the texel is eliminated (with roughly a dozen photons) any additional contributions mainly affect the texel intensity.

Due to the regular sampling of photons, attempts to reduce objectionable undersampling noise by increasing the number of emitted photons simultaneously increase oversampling. While this oversampling does not reduce caustic quality, it does introduce photons that needlessly increase computational costs.

We propose augmenting the hierarchical process employed above to avoid repeatedly splatting converging photons into a single texel. If convergence is detected at a coarse resolution, all child photons can be treated as a single photon with a corresponding increase in intensity. For scenes such as Figure 4 that exhibit severe oversampling, as many as 256 photons can be treated as a single bright photon, significantly reducing the cost of oversampling when using high

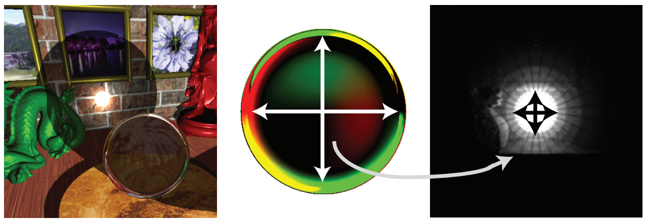

Figure 4: *A scene with severe oversampling in the caustic map. The central image shows the photon buffer, where the darker photons roughly correspond to the bright central region of the caustic map, shown to scale at right. Using 3^2 photon splats, roughly 90 photons contribute to each texel in the focal region.*

resolution photon buffers. The algorithm from Section 3.1 changes as follows:

```
PhotonList photonsForLevel[maxMipLevels+1];

// Only one texel in top mipmap level!
photonsForLevel[maxMipLevels].Add( **root** );

**for** (i=maxMipLevels down to 1) **do**
   intensityModifier = $4^i$
   **for all** (photons $\mathbf{p} \in$ photonsForLevel[i]) **do**
      **if** ( IsInvalid( **p** ) ) **then**
         **continue**;
      **if** ( DoChildrenConverge( **p** ) ) **then**
         photonsForLevel[0].Add( **p**, intensityModifier );
      **else**
         photonsForLevel[i-1].Add( **p**.Child(0), 1 );
         photonsForLevel[i-1].Add( **p**.Child(1), 1 );
         photonsForLevel[i-1].Add( **p**.Child(2), 1 );
         photonsForLevel[i-1].Add( **p**.Child(3), 1 );

TransformPhotons( photonsForLevel[0] );
```

Due to the behavior of geometry shaders, photons cannot be emitted in two different streams as suggested by this algorithm. Instead, our implementation emits all photons to a single stream and a flag identifies if a photon is ready to transform or if further hierarchy recursion is needed.

We found a simple convergence test worked reasonably well. If *all* child photons project to the same texel in the caustic map, we count the children as converging. While conservative, this test requires little additional state to be passed from level to level. In fact, we store the intensity modifier in the (unused) alpha component of the photon position. More complex tests could determine if photons converge primarily in one dimension, which may allow more aggressive photon clustering.

Always checking for photon convergence proves costly, especially for low resolution mipmap levels where essentially incoherent photons are unlikely to be clustered. We also implement a modified approach, where convergence is only tested at the last mipmap level (e.g., for the 512^2 resolution level when using a 1024^2 photon buffer). This reduces the shader cost for processing most mipmap levels, in exchange for missing some converging photons.

3.3 Reducing Undersampling

Perhaps the biggest obstacle to attaining high quality interactive caustics is difficulty eliminating undersampling noise. A simple way to remove this noise is to shoots more photons.

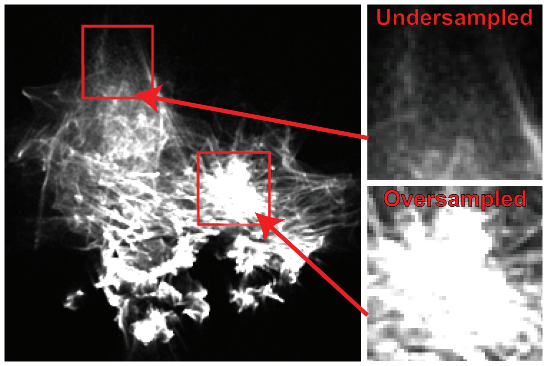

Figure 5: *Even after emitting millions of photons, caustic maps contain undersampled regions where lack of photons leads to noise. Due to the regularly sampled photon buffer, increasing photon count to reduce undersampling simultaneously worsens oversampling.*

Unfortunately, only parts of the caustic map are undersampled. In the same caustic map, there may be as many oversampled regions as undersampled regions (e.g., Figure 5). While the technique discussed in Section 3.2 reduces the costs associated with oversampling, it still incurs a penalty. Furthermore, even with millions of photons many caustics still contain undersampling noise. More than 16 million photons are infeasible using current hardware, and memory requirements for large buffers will inhibit the use of more photons on future hardware.

Once crisp caustics have been obtained in focal regions, blurring caustics in slowly changing regions should be acceptable in lieu of emitting more photons. Wyman and Dachsbacher [2007] take this approach, suggesting the use of variable-sized photon splats during caustic map creation. Their first approach interpolates between splats with four different radii, but this requires all photons to be rasterized with the largest splat size. Their second approach varies each photon's splat size individually, approximating photon divergence using a per-pixel thin-lens model. Unfortunately, this still requires costly rasterization of extremely large splats.

We observe that photon convergence and divergence is already implicitly stored in the photon buffer (similar to discussion by Collins [1995]), so a fragile thin-lens model is unnecessary; the photon buffer may be directly used to compute a photon's splat size based on the amount of magnification caused by specular interactions. Each photon $\mathbf{p}_i$ in the photon buffer actually represents a beam with solid angle ω_i emitted from the light. We compute a beam's magnification using a ratio of areas: the area of an splat from a theoretical unoccluded beam to the actual area spanned by 4 adjacent photons. The magnification gives the multiplicative factor applied to photon intensity, whereas the inverse of the magnification modifies the splat size (i.e., high magnification implies convergence, where photon energy is distributed over a smaller region).

Photon $\mathbf{p}_i$ hits a surface a distance d_i from the light. An unoccluded beam from the light would affect an area of size $\omega_i d_i^2$ (i.e., the surface area on a sphere of radius d_i). If the photons adjacent to $\mathbf{p}_i$ form a quadrilateral of area A_i, the ratio of areas $(\omega_i d_i^2 / A_i)$ describes the photon's intensity and the photon should be splatted into the caustic map with a

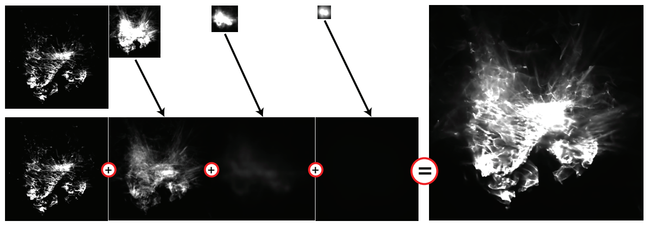
Figure 6: *High frequency regions of caustics, generated with small photon splats, are stored in the highest resolution mipmap level of the caustic map to preserve detail. Progressively lower frequency caustics are stored in lower resolution levels, where the large splats required to eliminate noise are cheaper to rasterize. Because energy from texels in lower resolution mipmap levels are spread over larger regions, contributions from mipmap level i must be scaled by* $(\frac{1}{4})^i$. *Summing contribution from all mipmap levels gives the final caustic map intensity.*

splat of radius r_i of

$$r_i = \sqrt{\frac{A_i}{\omega_i d_i^2}}. \tag{1}$$

Note that the square root accounts for using a ratio of areas rather than a ratio of lengths. Consider the effect of surface orientation on intensity. If $\mathbf{p}_i$ hits a surface with normal $\vec{\mathbf{N}}$ from direction $\vec{\mathbf{D}}_i$, then an unoccluded photon beam would illuminate an area of size $\frac{\omega_i d_i^2}{\vec{\mathbf{N}} \cdot \vec{\mathbf{D}}_i}$, leading to splat radius of:

$$r_i = \sqrt{\frac{A_i(\vec{\mathbf{N}} \cdot \vec{\mathbf{D}}_i)}{\omega_i d_i^2}}. \tag{2}$$

We found beam area computations lack robustness due to numerical precision errors on the GPU, so we use an approximation based on the maximum distance between adjacent photons rather than area. This slightly enlarges splats in regions where photons diverge more strongly in one direction.

This method for computing splat radii generates splat sizes that reduce virtually all noise, whereas Wyman and Dachsbacher's [2007] thin-lens approach still has undersampling noise. However, since photons can diverge arbitrarily far, our technique generates photon splats of arbitrarily large dimensions. These are often significantly larger than those generated by the thin lens method. Larger splats cost more to rasterize, with expense growing proportional to the square of the radius.

Our initial varying splat implementation reduced rendering speeds by an order of magnitude compared to simply using techniques from Sections 3.1 and 3.2. Here we apply a second observation. Extremely large Gaussian photon splats are very blurry and very dim. Thus, there is no need to rasterize them into a full resolution caustic map. A splat of radius 5 in a 128^2 buffer appears similar to a splat of radius 20 in a 512^2 buffer after scaling and normalization, yet the smaller splat costs significantly less to rasterize.

Instead of rendering our caustic map as a single image with a large variation of splat sizes, we propose rendering the caustic map as a large number of varying resolution images each with a small variation of splat sizes. If these splat sizes are all relatively small, the cost will be similar to generating a caustic map with small, fixed size splats. This can be seen as an image hierarchy, with high frequency caustics stored at the highest resolution and lower frequency caustics stored at progressively coarser resolutions. To make calculations simple, we use a mipmap image pyramid to halve the image dimensions with each level of coarsening (see Figure 6).

Given new functionality in DirectX 10 [Blythe 2006] class hardware, GPUs can render to multiple images simultaneously and each primitive independently selects its desired output image. Our implementation uses this feature to directly render all levels of the caustic hierarchy in a single pass. While current hardware requires all images to have the same resolution, the OpenGL specifications suggest this limitation will be lifted eventually, allowing direct render-to-mipmap. For now, we use render targets of the same resolution, using progressively smaller regions of each.

Our `TransformPhotons()` routine mentioned in Sections 3.1 and 3.2 must change from a simple projection of photons into the caustic map, as it must now calculate splat size and corresponding map level:

```
for all (photons $\mathbf{p}_i$ to transform) do
    Lookup solid angle $\omega_i$
    $d_i = \sqrt{dot(\mathbf{p}_i, \mathbf{p}_i)}$
    Find adjacent photons $\mathbf{p}_{j,k,l}$
    Find area $A_i$ of $\mathbf{p}_{i,j,k,l}$
    Find $r_i$ as in Equation 2.
    Find diameter $D_i = 2r_i$.

    // Max splat diameter in any level is 5.0
    photonMipLevel = trunc( $\log_2$( max( 1, $\frac{D_i}{5.0}$ ) ) )
    splatSize = $D_i$ / pow( 2, photonMipLevel )

    SplatPhoton( $\mathbf{p}_i$, photonMipLevel, splatSize )
```

This approach gives results like the hierarchy shown in Figure 6. However, texels in lower resolution caustic images cover larger regions of space than high resolution texels. This means their energy is spread over larger regions, and must be renormalized. Texels in the 1st level are 4 times too bright, texels at the 2nd level are 16 times too bright, and texels at the ith level are 4^i times too bright. Utilizing the multi-resolution caustic map costs more than a simple caustic map, as contributions from multiple levels must be renormalized and summed. To compute the caustic intensity C:

$$C = \sum_{i=0}^{N} \frac{C_i}{4^i}, \tag{3}$$

where C_i is the intensity in mipmap level i and N is the number of mipmap levels used.

We found that few splats are large enough to render to the 4th and higher mipmap levels, as such splats would have a diameter of more than 40 pixels. Furthermore, such photons contribute little to final caustic intensity. We discard such photons before splatting them into the caustic map, and only sum contributions from the base caustic map and the first 3 mipmap levels.

4 Implementation Issues

As shown in Figure 2, caustic mapping takes three passes: render a photon buffer, create a caustic map, and apply the caustic map in a final render. Commonly, implementations use the render-to-vertex buffer technique described by the `GL_ARB_pixel_buffer_object` OpenGL extension to send photons from the photon buffer back through the pipeline during caustic map creation. Unfortunately, this approach

# Photons	Render Photon Buffer	Copy to Vertex Array	Splat Photons	Eye Render
256^2	3.2 ms	4.8 ms	4.9 ms	8.2 ms
512^2	3.7 ms	15.5 ms	10.4 ms	8.2 ms
1024^2	5.2 ms	56.0 ms	35.8 ms	8.1 ms
2048^2	10.4 ms	215.5 ms	142.9 ms	8.0 ms
4096^2	31.7 ms	861.5 ms	777.4 ms	8.4 ms

Table 1: *Costs for the caustic mapping phases of Wyman and Davis [2006] for the scene in Figure 7. Note that rendering to a vertex buffer currently requires an internal copy from the framebuffer to a buffer object.*

# Photons	16-bit FBO to 32-bit VBO	32-bit FBO to 32-bit VBO	16-bit FBO to 16-bit VBO
256^2	4.8 ms	1.0 ms	0.5 ms
512^2	15.5 ms	3.6 ms	1.8 ms
1024^2	56.0 ms	14.0 ms	6.9 ms
2048^2	215.5 ms	57.6 ms	27.4 ms
4096^2	861.5 ms	244.9 ms	120.6 ms

Table 2: *Data type significantly affects copy costs. Naive use of a standard 32-bit float vertex buffer and a space-saving 16-bit float framebuffer gives extremely poor performance. Using identical data types on both sides of the copy allows the GPU to avoid conversion, speeding performance.*

requires a copy from framebuffer memory into a vertex buffer. As shown in Table 1, this copy cost dominates rendering time, especially as photon count increases. These timings describe render times for the image in Figure 7.

Fortunately, careful consideration shows a number of simple ways to reduce this cost. First, data type dramatically affects copy cost. To reduce framebuffer memory and allow larger photon buffers, one might use a 16-bit floating point buffer to store photon hitpoints. Using *glReadPixels()* to copy photons into a floating point vertex buffer gives the results in Table 1. However, this requires the GPU to convert from 16- to 32-bit floats. Using a consistent 16- or 32-bit data type on both sides of the copy eliminates this conversion and reduces copy time by a factor of 4 to 8 (see Table 2).

However, even these savings do not completely eliminate copy cost. In the past this copy was necessary, as a vertex shader texture lookup to find a photon's location was, collectively, more costly than even a naive copy. DirectX 10 class GPUs largely eliminate differences between vertex and fragment shaders, making direct texture lookups feasible and completely eliminating the need to copy framebuffer data to a vertex buffer.

Interestingly, by using the Shader Model 4.0 built-in variables `gl_VertexID` or `gl_PrimitiveIDIn`, no data need be passed to the pipeline; even the photon's vertex location is superfluous, as the shader can fetch the photon position from texture based upon the photon's ID. Unfortunately, OpenGL requires vertex data for each point, even if it remains unused.

Still, it seems wasteful to setup a 1024^2, 2048^2, or 4096^2 array of garbage to bind via *glVertexPointer()* simply to satisfy the pipeline's need for vertex data. Instead of using *glDrawArrays()* or *glDrawElements()* to plow through such a buffer, we create a smaller buffer of size 1024, 2048, or 4096 and use *glDrawArraysInstancedEXT()* to repeatedly draw the same array of vertex data. This reduces memory overhead and also simplifies shader computations, as we can use built-in variables `gl_InstanceID` and `gl_VertexID` as s and

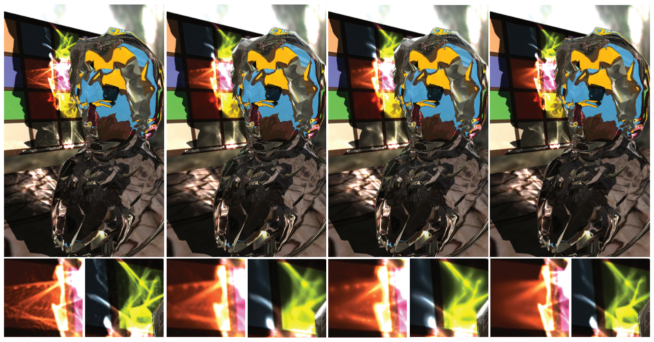
Figure 7: *A comparison of the quality of (from left to right) caustic maps with constant splat sizes, varying splats via a thin-lens model, interpolating multiple resolution splats, and our new approach proposed in Section 3.3.*

t coordinates into our photon buffer. This speeds results further, and the numbers reported in the following section reflect all these improvements.

5 Results and Discussion

We implemented hierarchical caustic maps in OpenGL, with our test platform running on a 2.6 GHz quad-core Intel Xeon processor and a GeForce 8800 GTX. All timings quoted throughout the paper reflect a display resolution of 2048^2, which was resampled for an antialiased 1024^2 final image.

Table 3 compares the single-frame cost for a variety of steps in the caustic mapping process using hierarchical and non-hierarchical techniques. Figure 7 shows the frame used for timing. These results show that our caustic map creation costs are up to 50 times faster than previous work [Wyman and Davis 2006], and even after factoring out implementation optimizations our approaches improve performance as much as ninefold.

Generally we found the hierarchical gather with converging photon clustering to be fastest. Except for very high photon counts, photon convergence need be tested only at the lowest mipmap level—the additional shader costs for testing convergence are not balanced by a corresponding performance improvement without a very dense photon sampling. However, combining converging photons at just the final mipmap level improves performance by 30-50% when using more than 1 million photons.

Creation of a multi-resolution caustic map takes roughly twice as long due to the slight variation in splat size, though this is significantly faster than splatting arbitrary sized splats into a single caustic map. Performance can be improved by changing the threshold where points are rasterized into different resolution images. However, we found a splat size threshold of 5.0 gave the best performance without introducing noticeable aliasing artifacts from using lower resolution caustic images.

Table 4 provides framerates for the same scene in Figure 7. Because the framerates were averaged over an animation while timings in Table 3 came from a single frame, the numbers are not exactly comparable.

Figures 1, 4, and 8 show more complex scenes, with between 316k and 560k polygons and two light sources. Table 5 shows framerates for these scenes. Until at least 2048^2 photons are used, the pipeline is mostly geometry bound. Keep

# Photons	Render Photon Buffer	Wyman & Davis [2006] Gather	W&D After Improving w/ Sec 4	Hierarchical Gather	Combine$_{last}$ Converging Gather	Combine$_{all}$ Converging Gather	Varying Splats, Multi-Res Map	Varying Splats, Single Map	Final Render w/ Caustic Map	Final Render w/ Multi-Res Caustic Map
256^2	3.2 ms	4.9 ms	1.0 ms	1.3 ms	1.1 ms	1.1 ms	1.5 ms	4.8 ms	8.2 ms	8.3 ms
512^2	3.7 ms	10.4 ms	1.9 ms	1.6 ms	1.5 ms	1.5 ms	2.6 ms	12.9 ms	8.2 ms	8.2 ms
1024^2	5.2 ms	35.8 ms	5.7 ms	3.2 ms	2.3 ms	2.6 ms	5.2 ms	31.6 ms	8.1 ms	8.3 ms
2048^2	10.4 ms	142.9 ms	20.8 ms	9.8 ms	5.6 ms	5.8 ms	13.3 ms	78.7 ms	8.0 ms	8.3 ms
4096^2	31.7 ms	777.4 ms	136.0 ms	39.4 ms	16.5 ms	15.1 ms	39.3 ms	330.1 ms	8.4 ms	8.5 ms

Table 3: *Costs for various stages of the caustic mapping process for the scene in Figure 7. The gather stages represent the cost to splat the points from the photon buffer into the caustic map, with Section 3.1 describing the hierarchical gather, Section 3.2 describing the two converging gathers, and Section 3.3 describing the varying gather. During the final render, using the caustic image pyramid requires slightly more time than using a standard caustic map.*

# Photons	W/D (fps)	W/D+ (fps)	Hier (fps)	Comb$_{last}$ (fps)	Comb$_{all}$ (fps)	Varying (fps)
256^2	55.2	75.6	70.6	71.6	71.0	69.9
512^2	32.9	69.0	66.8	68.1	67.6	62.5
1024^2	12.8	51.2	55.8	58.2	57.4	47.9
2048^2	3.6	25.5	32.8	37.9	37.4	27.7
4096^2	0.6	5.8	12.3	16.7	17.0	11.4

Table 4: *Framerates for the techniques listed in Table 3. Note these are averages over an animation sequence, and thus are not equivalent to the single-frame values in Table 3.*

	# Photons				
	256^2	512^2	1024^2	2048^2	4096^2
Sphere in Room (316k triangles)	32.4	31.8	29.5	24.4	15.5
Shark in Room (318k triangles)	34.0	32.8	30.5	25.0	13.7
Triceratops in Room (318k triangles)	33.6	32.4	29.5	22.3	12.3
Buddha in Room (365k triangles)	25.4	24.8	22.9	17.4	9.9
Bunny in Room (385k triangles)	19.4	19.2	18.2	14.9	9.3
Dragon in Room (565k triangles)	9.9	9.9	9.6	8.9	6.3
Gargoyle in Room (515k triangles)	7.3	7.2	7.0	6.5	4.7
Armadillo in Room (530k triangles)	7.1	7.1	6.9	6.5	5.2

Table 5: *Framerates (fps) for the scenes shown in this paper using hierarchical caustic maps and varying photon counts. The final rendered images were* 2048^2.

in mind the refractor is rendered 6 times (using [Wyman 2005]) and the rest of the scene is rendered 3 times, so between 1 and 2.4 million triangles are rasterized in addition to photon processing. The hierarchical techniques presented in the paper reduce the number of photons processed by more than an order of magnitude. For the dragon, even with two lights emitting 4096^2 photons, fewer than 2 million points are processed (in total) during the hierarchical traversal—this is fewer than the 2.4 million triangles required to simply draw the scene from the eye and the two lights.

Object size also plays a factor in rendering costs. With previous techniques, this was not a factor since all photons were processed without regard to their validity. The armadillo and gargoyle in Figure 8 are significantly larger than other models, so more photons interact with them and must be processed. Since both refraction and caustic splatting costs depend on the object's pixel coverage, these two models run slower than the more geometrically complex dragon.

6 Conclusion and Future Work

This paper presented three hierarchical improvements to existing caustic mapping techniques and discussed a number of implementation optimizations. This results in up to fifty-fold speedup for caustic map creation, with up to an order of magnitude improvement attributable to our hierarchical work. These speedups allow scenes using a couple million caustic photons to remain real-time, and even scenes with more than 32 million photons still remain interactive. This should allow developers to utilize larger photon buffers for improving caustic quality without sacrificing too much performance.

Section 3.3 introduced a new method for varying photon splat sizes based on the local divergence of light. This virtually eliminates undersampling noise by replacing noise with large blurry photons, yet it avoids reducing fidelity in sufficiently sampled focal regions. This improves upon the work of Wyman and Dachsbacher [2007], which still leaves noise and can introduce blurring or halo artifacts in focal regions.

Our work is generally orthogonal to other caustic map improvements; for instance, the line tracing work of Krueger et al. [2006] should directly benefit from a hierarchical processing of photons. However, our work benefits mainly applications utilizing dense photon samplings, as a hierarchical traversal of the photon buffer improves performance only for photon buffers containing more than 512^2 photons.

Beyond speed and noise considerations, our work retains standard caustic mapping problems. In particular, area and environmental light sources are not handled. Additionally, interactive reflection and refraction are not solved problems. Thus, reflected and refracted photons may be incorrectly traced through the scene, causing the rendered caustics to exhibit artifacts.

According to the timings in Table 3, the current bottleneck during caustic mapping is rendering the photon buffer. Furthermore, larger photon buffers are simply infeasible due to hardware memory limits. This suggests a need for more intelligent photon emission to avoid a full, regularly sampled framebuffer. Adaptive and perspective shadow maps [Fernando et al. 2001; Stamminger and Drettakis 2002] propose solutions in the context of shadow mapping, and we believe similar approaches may prove fruitful for caustics.

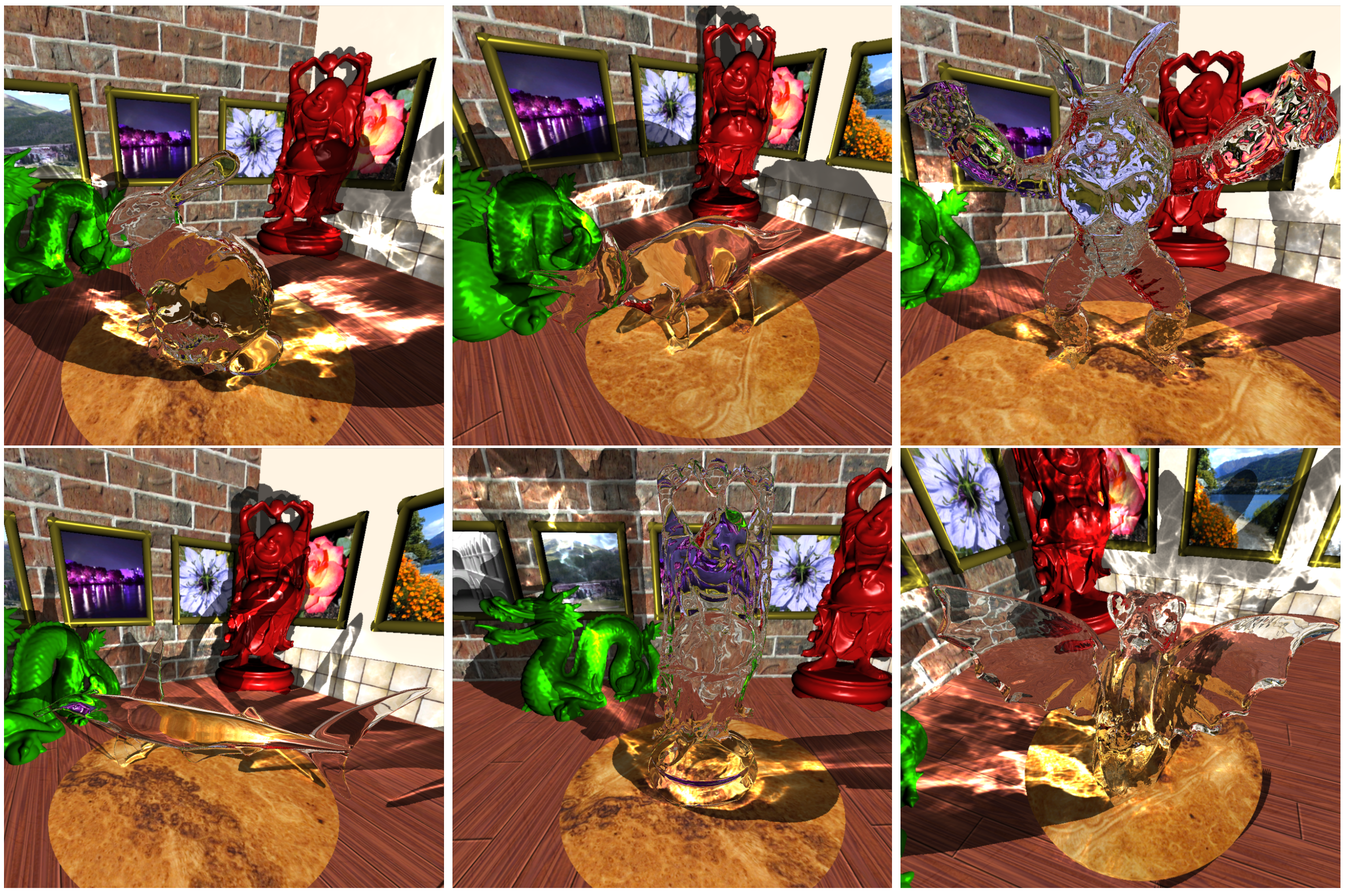

Figure 8: *Results using hierarchical caustic mapping with complex refractors and multiple lights.*

References

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

BLYTHE, D. 2006. The direct3d 10 system. *ACM Transactions on Graphics 25*, 3, 724–734.

BUNNEL, M. 2005. *GPU Gems 2*. Addison-Wesley, ch. Dynamic Ambient Occlusion and Indirect Lighting, 223–233.

COLLINS, S. 1995. *Photorealistic Rendering Techniques*. Springer, ch. Adaptive Splatting for Specular to Diffuse Light Transport, 121–135.

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

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

DAVIS, S., AND WYMAN, C. 2007. Interactive refractions with total internal reflections. In *Proceedings of Graphics Interface*, 185–190.

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.

ESTALELLA, P., MARTIN, I., DRETTAKIS, G., AND TOST, D. 2006. A gpu-driven algorithm for accurate interactive reflections on curved objects. In *Proceedings of the Eurographics Symposium on Rendering*, 313–318.

FERNANDO, R., FERNANDEZ, S., BALA, K., AND GREENBERG, D. 2001. Adaptive shadow maps. In *Proceedings of SIGGRAPH*, 387–390.

FOLEY, T., AND SUGERMAN, J. 2005. Kd-tree acceleration structures for a GPU raytracer. In *Proceedings of Graphics Hardware*.

GREENE, N. 1986. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications 6*, 11, 21–29.

HASENFRATZ, J.-M., LAPIERRE, M., HOLZSCHUCH, N., AND SILLION, F. 2003. A survey of real-time soft shadow algorithms. *Computer Graphics Forum 22*, 4, 753–774.

HECKBERT, P. S., AND HANRAHAN, P. 1984. Beam tracing polygonal objects. In *Proceedings of SIGGRAPH*, 119–127.

Ihrke, I., Ziegler, G., Tevs, A., Theobalt, C., Magnor, M., and Seidel, H.-P. 2007. Eikonal rendering: efficient light transport in refractive objects. *ACM Transactions on Graphics 26*, 3 (July).

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.

Kruger, J., Burger, K., and Westermann, R. 2006. Interactive screen-space accurate photon tracing. In *Proceedings of the Eurographics Symposium on Rendering*, 319–329.

Mitchell, D., and Hanrahan, P. 1992. Illumination from curved reflectors. In *Proceedings of SIGGRAPH*, 283–291.

Ng, R., Ramamoorthi, R., and Hanrahan, P. 2004. Triplet product wavelet integrals for all-frequency relighting. *ACM Transactions on Graphics 23*, 3, 477–487.

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

Oliveira, M. M., and Brauwers, M. 2007. Real-time refraction through deformable objects. In *Proceedings of the 2007 Symposium on Interactive 3D graphics and games*, 89–96.

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.

Shah, M., Konttinen, J., and Pattanaik, S. 2007. Caustics mapping: An image-space technique for real-time caustics. *IEEE Transactions on Visualization and Computer Graphics 13*, 2 (March/April), 272–280.

Sloan, P.-P., Kautz, J., and Snyder, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics 21*, 3, 527–536.

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

Szirmay-Kalos, L., Aszodi, B., Lazanyi, I., and Premecz, M. 2005. Approximate ray-tracing on the gpu with distance impostors. *Computer Graphics Forum 24*, 3, 685–704.

Umenhoffer, T., Patow, G., and Szirmay-Kalos, L. 2007. *GPU Gems 3*. Addison-Wesley, ch. Robust Multiple Specular Reflections and Refractions, 387–407.

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.

Wei, H., and Kaihuai, Q. 2007. Interactive approximate rendering of reflections, refractions, and caustics. *IEEE Transactions on Visualization and Computer Graphics 13*, 1 (January/February), 46–57.

Williams, L. 1978. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH*, 270–274.

Wyman, C., and Dachsbacher, C. 2007. Improving image-space caustics via variable-sized splatting. *Journal of Graphics Tools (to appear)*.

Wyman, C., and Davis, S. 2006. Interactive image-space techniques for approximating caustics. In *Proceedings of ACM Symposium on Interactive 3D Graphics and Games*, 153–160.

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

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.