Interactive Shadowed Caustics Using Hierarchical Light Volumes

Josh Barczak * Marc Olano*

University of Maryland Baltimore County

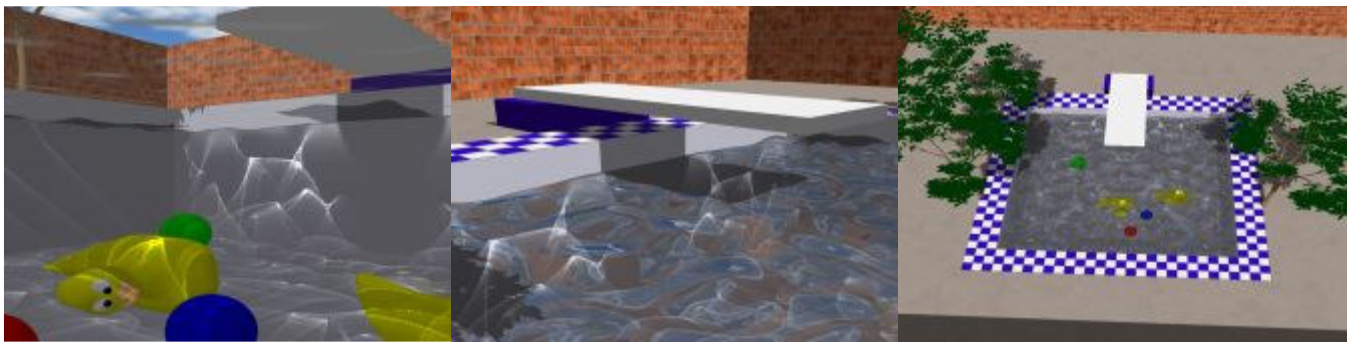


Figure 1. Three examples of refractive caustics and shadows, rendered using our interactive algorithm

Abstract

The interplay of refracted light and shadow is an important component of underwater scenes. Full and correct interactive rendering of refracted caustics remains beyond the capabilities of even the most recent graphics hardware. We present an interactive caustic-beam rendering algorithm that improves on prior methods in three important ways: it uses a hierarchical structure over the refracting object to speed culling and clamping of light beams; it correctly handles shadowing by objects above the water surface, and can eliminate computation for shadowed regions; and it efficiently uses GPU vertex programming to reduce the rendering cost for caustic beams. In addition, we present a variation to the algorithm using a floating point texture for scaled vector rather than intensity accumulation for more correct caustic appearance at a cost in performance.

CCS Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, Shading, Shadowing and Texture

Keywords: Real-Time Rendering, Global Illumination, Illumination Volumes, Caustics, Shadows

1 Introduction

Caustics, the beautiful patterns created when moving water focuses refracted light, are a significant factor in the appearance of underwater scenes, but are difficult and expensive to reproduce correctly in interactive computer graphics. Figure 1 shows several examples of refractive caustics in water. Various techniques have been used to approximate caustics from water in interactive applications, but these techniques tend to be slow. Most previous techniques have been confined to rendering caustics onto flat planes or simple height fields, and are unable to handle the general problem of rendering caustics onto arbitrary objects suspended in water.

Another difficult aspect of refractive caustics is the interplay between caustics and shadows. An object above a pool of water will cast a shadow on underwater objects, but the shadow will be irregularly shaped, since light can be refracted by the surface of the water into the area underneath the object. Similarly, an object suspended under the water will cast an irregular shadow onto the ground. No previous interactive technique includes realistic interactions between caustics and shadows, while still rendering caustics on arbitrary objects in real time.

We present a new technique for rendering caustics from water which is able to render caustics onto any underwater object at acceptable frame rates on current graphics hardware. Our method supports dynamic changes to the scene, and also provides a partial solution to the shadowing problem, in the case where an occluding object lies between the light source and the water's surface. Our work improves upon previous caustic rendering methods based on light beam tracing [Nishita 1994, Iwasaki et al. 2001, 2003, Watt 1990] by introducing a hierarchical data structure which reduces the rendering overhead and allows efficient culling of beams shadowed by occluding objects between the water surface and the light source.

The remainder of this paper is organized as follows: Section 2 provides an overview of previous work related to caustic rendering, in both non-interactive and interactive applications, Section 3 describes our new rendering technique, and Section 4 provides a performance analysis

2 Previous Work

Caustics are a form of specular to diffuse transport, which is most commonly associated with images produced by ray tracing. Typically, rays are traced from the light source into the scene, and the density of hits in a particular location is used to determine an estimate of the irradiance at that location. Backward ray tracing was one of the earliest techniques capable of generating caustics [Arvo 1986], but more recently photon mapping has gained popularity for its ability to model a wide variety of lighting effects, including caustics [Jensen 1996]. Although photon mapping has been implemented in a modified form on a commercial graphics accelerator [Purcell et al. 2003], the technique is not yet practical for dynamic scenes, since the photon tracing process must be repeated whenever there is a change to the objects or lights.

Wyman et al. [2003, 2004] render dynamic caustics by sampling the radiance in a region of space around each caustic producing object in their scenes, and using the sampled caustic information to shade nearby surfaces. They are able to achieve interactive frame rates using a large shared-memory machine, but their technique consumes a great deal of memory and will not scale well to complex scenes on commercial graphics hardware.

*{jbarcz1, olano}@cs.umbc.edu

Spherical harmonics can be used to compress the sampled caustic information, but because of the high frequency nature of caustics, many coefficients are needed. In addition, the method uses ray-tracing to sample the caustics from an object as a pre-process, which means that caustics could not be cast from deformable objects or moving light sources. Pre-computed radiance transfer [Sloan et al. 2002] can be used in a similar fashion to calculate light transfer functions from an object to nearby points in space, but it suffers from the same limitations.

A novel technique developed by Wand and Staber [2003], allows interactive caustics with dynamic lighting by subdividing each caustic producing object into a series of sample points. Each sample point projects the incoming light onto the surface of nearby objects, and the caustic intensity is computed by summing over the sample points. The algorithm is implemented by computing the light from each sample point in a pixel shader, and filtering based on the surface curvature to prevent aliasing. Although Wand and Staber considered only reflective caustics, it is straightforward to apply their technique to refractive caustics in an underwater scene. The main limitation of this method is scalability. To apply the technique to our test scenes with any degree of accuracy, hundreds, perhaps thousands, of surface samples would have to be evaluated per pixel for each underwater object.

Because caustics are such an important aspect of underwater scenes, many special purpose techniques have been developed to approximate refractive caustics from water in interactive applications. An early example is the work of Stam [1996], who developed a technique that uses pre-computed, animated caustic textures which are projected onto the objects in the scene. Because the caustic patterns are computed in advance and stored in texture maps, caustic rendering is inexpensive, but the caustics will not look correct when projected onto an underwater object, and cannot be made to change in response to changes in the water surface geometry.

Trendall [2000], in an early example of general purpose computation on graphics hardware, demonstrates a technique to calculate caustics on a flat plane analytically from a dynamic water surface. Because Trendall's technique uses an integral equation for caustic intensity, it cannot be easily extended to handle shadows.

A more popular and general method, which has been applied to ocean scenes [Jensen et al. 2001, Lovisarch 2003], is to tessellate the water surface and project the resulting mesh onto the sea floor. The projected water mesh is rendered, and the contributions of the individual triangles to the final image are summed at each pixel. This is an efficient technique for rendering dynamic caustics, and can support displacement mapping of the receiving surface. However, it does not handle caustics on arbitrary objects.

Nishita and Nakamae [1994] present a beam-tracing technique for rendering caustics from water, as well as shafts of light due to scattering from impurities in the water. Their method is based on a beam tracing technique published by Watt [1990], and a modified version has been implemented on graphics hardware [Iwasaki et al. 2001]. Each triangle of the water surface is swept down through the water, creating a pyramid shaped light beam. The energy incident on the surface of the triangle travels down through this beam, and the intensity at any point along the beam can be approximated by projecting the beam onto a plane containing the point, and using the ratio of the area of the projected triangle to the area of the original triangle. This allows caustics to be rendered with reasonable accuracy onto any underwater object, given a fine enough subdivision of the water surface. The hardware implementation also introduced the use of a shadow map to produce shadows in the caustics from underwater objects, but this is only a rough approximation, since it does not account for the possibility of light scattering into the space beneath the object from another direction.

Iwasaki et al. have published a more recent work [2003], which uses a different technique to allow for refraction mapping on the water surface. In this new method, underwater objects are rendered into a series of slice images, and caustics are calculated by projecting the caustic beams onto the slicing planes for each object. The slices are mapped onto the water surface and rendered using texture-based volume rendering techniques [Engel et al. 2001]. The use of object slices enables their new algorithm to handle the difficult problem of reflected or refracted images of nearby objects on the water surface by ray casting through the slices, but it also requires the caustic beams, the objects, and the water surface to be rendered multiple times for each object in the scene. Our rendering techniques are complementary and could be applied to accelerate their algorithm.

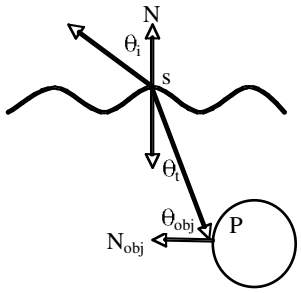


Figure 2. Geometry of caustic refraction.

3 Our Rendering Algorithm

Nishita et al's illumination volumes [Nishita 1994, Iwasaki et al. 2001, 2003], allow caustics to be projected from an arbitrary light source, through a dynamic polygonal surface, onto any type of underwater object. The method will work regardless of the way the underwater object is modeled, as long as a Z-buffered rendering of the object surface can be produced. Also, each light beam only needs to be rendered a constant number of times, regardless of the complexity of the objects in the scene. Because it supports rendering dynamic caustics onto any type of object, this method is the most general way to interactively render refractive caustics from water, and we have adopted it as the basis for our work.

3.1 Illumination Model for Caustics

Figure 2 shows the relevant angles and vectors for caustic refraction of an incident light ray through water, and shining on an underwater object. The energy of light reaching a point *p* underwater, from a point *s* on a surface triangle can be expressed as:

$$I_{sp} = I_s * T(\theta_{is}, \theta_{ts}) * (F_{sp})$$

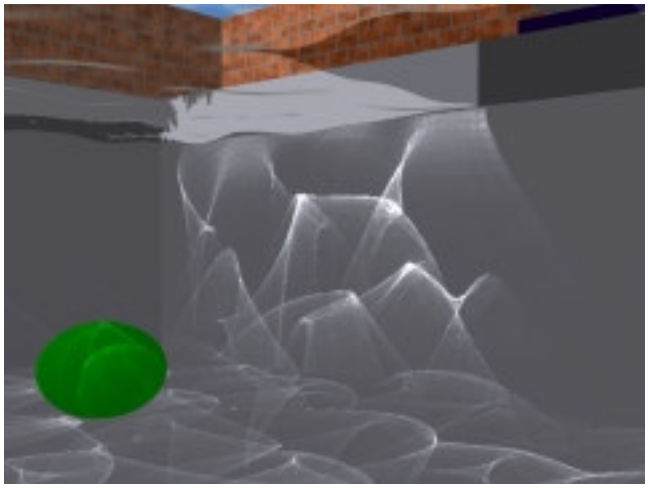
Where: I_s is the light intensity reaching point *s*, θ_{is} and θ_{ts} are the incidence and transmittance angles for a ray striking the water at point *s*, $T(\theta_{is}, \theta_{ts})$ is the Fresnel transmittance of the surface, and F_{sp} is the flux density ratio at *p*, which can be approximated by using the ratio of the area of a surface triangle containing *s* to the area of its projection onto a plane containing *p*. The reflected radiance from *p*, assuming perfectly diffuse objects, can be approximated as:

$$I_r = R_{obj} * (Ka + \sum I_{sp} * \cos(\theta_{obj}))$$

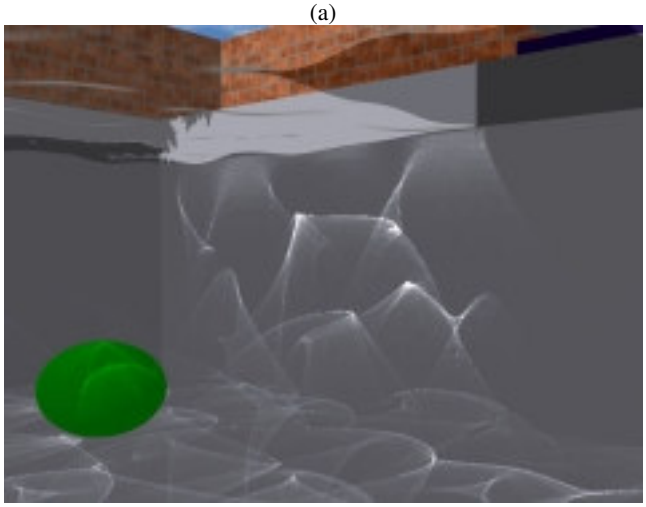
Where: θ_{obj} is the angle between the object normal and the refracted ray, and R_{obj} is the reflectance of the object. The total illumination reaching the point is the sum of the intensity contributed by each point s on the water surface. In addition, we add an ambient term: Ka, to account for indirect illumination due to diffuse interactions between objects and light scattering by water particles.

3.2 Caustic Beam Rendering

We render individual caustic beams using the same method as Iwasaki's hardware implementation of illumination volumes [2001]. Each triangle of the water surface is projected along the refracted ray direction at each vertex, producing a pyramid-shaped light beam. The light beam is sliced by a set of fixed, equally



(a)



(b)

Figure 3. (a) Flat-water approximation, accumulating only caustic beam intensities (14.98 fps). (b) Floating point scaled vector accumulation (11.64 fps).

spaced planes. For simplicity, our implementation always uses slicing planes perpendicular to the Y axis, but any set of parallel planes could be used. The caustic intensity is computed at each slice vertex and interpolated along the length of each slice.

Prior to rendering caustic beams, we initialize the stencil buffer to a reference value, and render the underwater scene with color outputs disabled in order to obtain depth values for each pixel. We then render the faces of each caustic beam to determine its effect on the visible points in the image. This is similar to stencil-based shadow volume rendering [Heidmann 1991]. For each beam, we compute the caustic intensity for each pixel and sum the intensities into a screen-sized texture.

Each caustic beam is rendered in two passes. In the first pass, we increment the stencil value for counter-clockwise faces, and decrement for clockwise faces. This causes the stencil value for each pixel to differ from the reference value if and only if the point under the pixel is inside the light beam. In the second pass, we output the computed energy for those pixels, and reset the stencil value for the next beam. Once all beams have been processed, the underwater objects are rendered a second time. In this pass, the lighting model is evaluated using the caustic texture for the summation term. Like previous implementations [Nishita 1994, Iwasaki et al. 2001, 2003], we approximate the cosine term in the lighting model by computing the refracted light direction assuming a flat water surface.

Because graphics hardware is beginning to support floating point frame buffer blending, we can use a floating-point caustic texture to remove the flat-water cosine approximation. Instead of accumulating intensities ($\sum I_{sp} N \bullet L_{flat}$), we can accumulate scaled caustic light vectors ($N \bullet \sum I_{sp} L$). Removing the flat-water cosine approximation has the greatest visual effect for surfaces perpendicular to the flat water surface. Observe the pool sides in Figure 3. Floating point vector accumulation incurs a frame-buffer bandwidth overhead relative to the simpler intensity accumulation of the flat-water model, which can lead to a significant performance penalty. This performance penalty is analyzed in more detail in Section 4.

The per-vertex calculations for each beam slice are implemented using the GPU. When rendering a beam slice, we use an array of dummy vertices, three for each slicing plane. Each dummy vertex stores the location of the slicing plane, and a vertex ID which maps the dummy vertex to a vertex on the surface triangle. The vertices of the surface triangle are packed into constant registers, and used to compute the projected position and intensity for each dummy vertex. Using the vertex shader allows our implementation to exploit the parallelism available in the GPU vertex pipeline for better performance. In addition, because all beams use a common array of dummy vertices stored in GPU memory, we avoid streaming large amounts of vertex data to the GPU for every beam, removing a potential performance bottleneck.

Because beams are numerous, and are rendered one at a time, pixel fill rate and driver overhead are major bottlenecks for this algorithm. In order to achieve interactive frame rates, view frustum culling must be performed to avoid rendering calls for invisible beams, and rendered beams must be clamped to the bounding boxes of the underwater objects in order to minimize the fill-rate demand. As the complexity of the scene increases, these operations become increasingly more numerous, and their computational cost becomes a more important component of the rendering time.

3.3 Hierarchical Beam Rendering

In order to reduce the computational cost of caustic rendering, we have developed a hierarchical rendering algorithm to accelerate the clamping and culling operations.

As a pre-processing step, we organize the triangles of the water mesh into a series of buckets, each of which covers a certain area of the water surface. These buckets are each recursively split a pre-set number of times, producing a set of full quadtrees. Each node of a quadtree stores a list of all triangles whose bottom-left vertex lies within the node's designated area of the water surface. Although we have chosen to use a regular grid in our implementation, arbitrary bucket layouts could be used depending on the shape of the water body and the locations of potential occluders.

Whenever the water geometry is updated, we compute a bounding volume for each leaf node by projecting the surface triangles onto the most distant slicing plane and computing an axis-aligned bounding square in the plane which encloses all of the projected points. We connect the corners of this box to the corners of the node's water surface area to produce a convex hull which fully encloses every caustic beam produced by the triangles in the node (Figure 5). We refer to these convex hulls as *parent beams*. After computing parent beams for the leaf nodes, we use them to recursively compute a convex hull for the intermediate nodes, such that each parent beam fully encloses all of its child beams, and therefore, all of the caustic beams owned by the parent. The cost of maintaining this hierarchical structure varies depending on the depth of the hierarchy and the number of root buckets used, but we have found that it is relatively cheap compared to the cost of clamping, culling, and rendering.

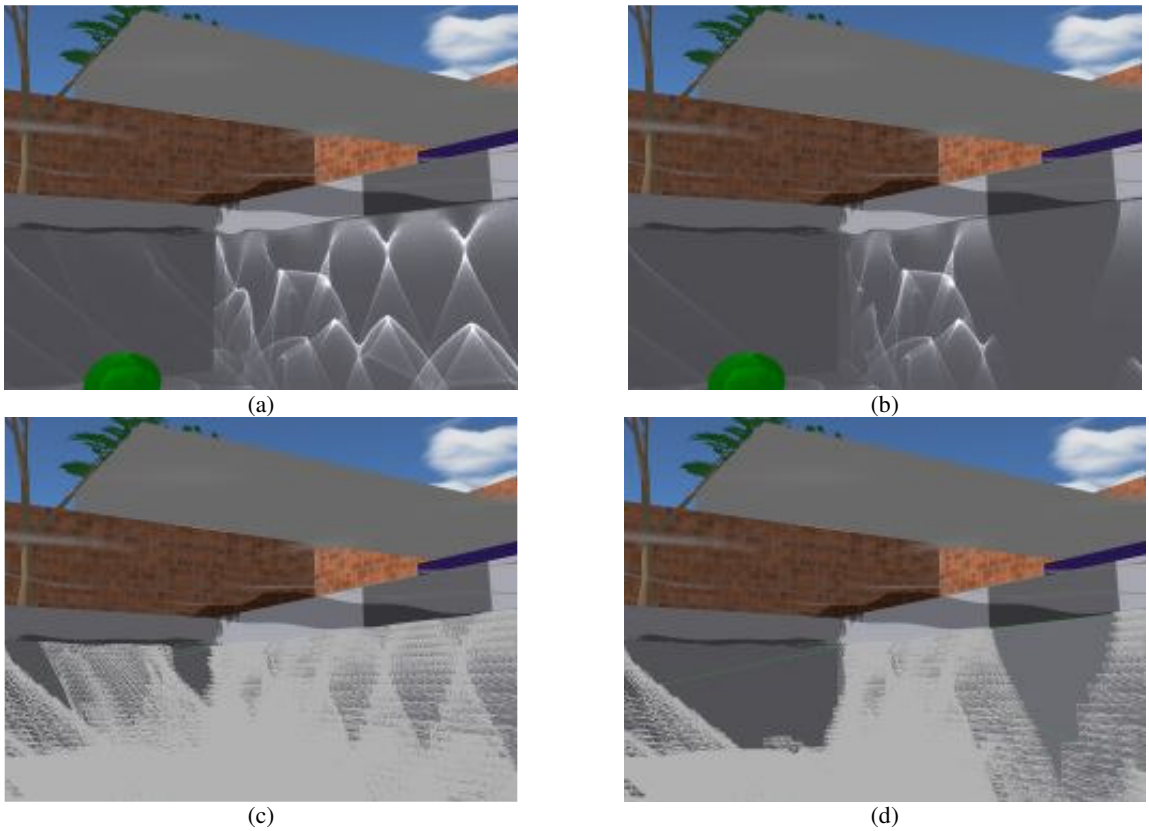


Figure 4. Effect of shadowing on caustics. (a) Caustics without shadowing. (b) Caustics with vertex shadow map and shadow culling. (c) Caustic beam structure without shadow culling (d) Caustic beam structure with shadow culling.

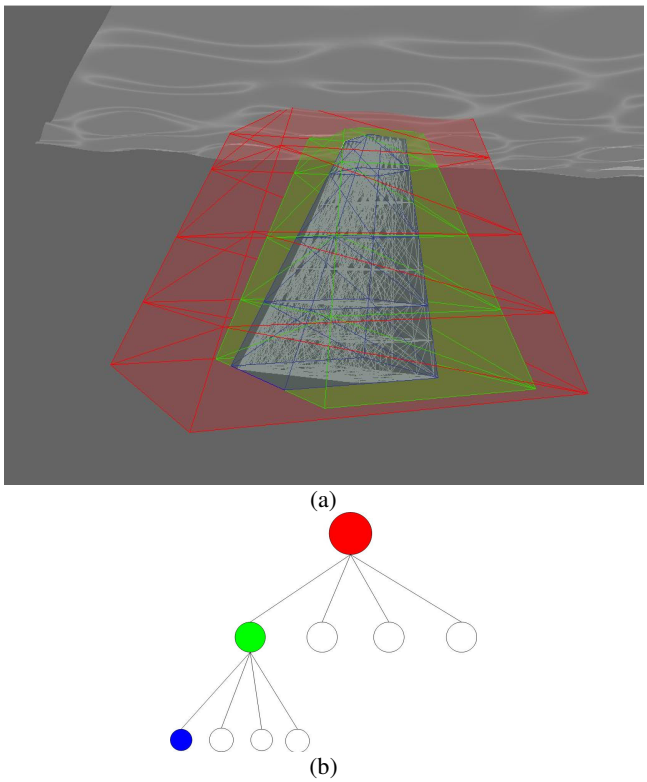


Figure 5. Hierarchical beam structure. (a) Geometry of three levels of hierarchy. (b) Quadtree layout for these beams.

This hierarchical beam structure reduces the cost of beam clamping and culling by allowing us to avoid performing these operations on individual beams if a test of the parent volumes will suffice. If a parent beam is found to intersect with the viewing frustum, we recursively test the child beams until we come to the leaves, at which point we test the individual caustic beams. If any parent beam is completely inside or outside the frustum, we can skip the culling tests on its children.

To perform beam clamping, we first test the parent beams for intersection with the bounding boxes of each underwater object. If an intersection is found, we recursively test all of its child beams, then clamp only those caustic beams which actually hit the box. The clamping can be made even more efficient by checking to see if the entire parent beam is contained in the bounding box at any point along its length. If this is the case, we know that every caustic beam must eventually strike the box, and we can clamp all of them to the object's extents without further testing.

3.4 Hierarchical Shadow Culling

The introduction of vertex texturing in the current generation of graphics hardware makes it possible to improve the realism of our caustic renderings by using a shadow map in the vertex shader to detect caustic beams which are shadowed by objects above the water. For each vertex of the water surface mesh, we can use the shadow map to determine whether the vertex is visible from the light's point of view, and if it is not, then the intensity projected through that vertex of the light beam is set to zero. The top half of Figure 4 demonstrates the increased realism that shadows can bring to a scene.

Performance (FPS)	Figure 1, Left			Figure 1, Center			Figure 1, Right			Figure 3, Top		
Reference	17.6	10.44	6.87	23.00	13.83	9.10	13.05	7.67	5.01	27.7	16.97	11.15
Hierarchical	28.75	18.74	12.74	31.4	20.73	14.14	27.70	18.01	12.19	33.6	22.47	15.30
Speedup (%)	63.35	79.50	85.44	36.52	49.89	55.38	112.26	134.81	143.31	21.3	32.41	37.22

Table 1. Clamping and Culling Performance. Results for each view are in ascending order by water mesh size. Mesh sizes, from left to right, are: 120x120, 160x160, and 200x200 vertices.

	Figure 1, Left			Figure 1, Center			Figure 1, Right			Figure 3		
Reference	6.51	4.57	3.33	10.75	7.15	4.98	5.51	3.31	2.19	11.51	8.37	6.20
Hierarchical	6.85	4.89	3.58	11.98	7.98	5.17	5.86	3.50	2.25	12.55	9.39	6.46
Shadow Culling	7.33	5.40	4.00	13.34	9.32	6.05	6.55	4.07	2.59	14.98	11.50	7.90
Vector Accum	5.58	4.26	3.27	11.48	8.51	5.81	6.34	4.00	2.54	11.64	9.44	7.24

Table 2. Rendering Performance. Bottom row shows framerate of hierarchical rendering with shadow culling and vector accumulation lighting (see Section 3.2).

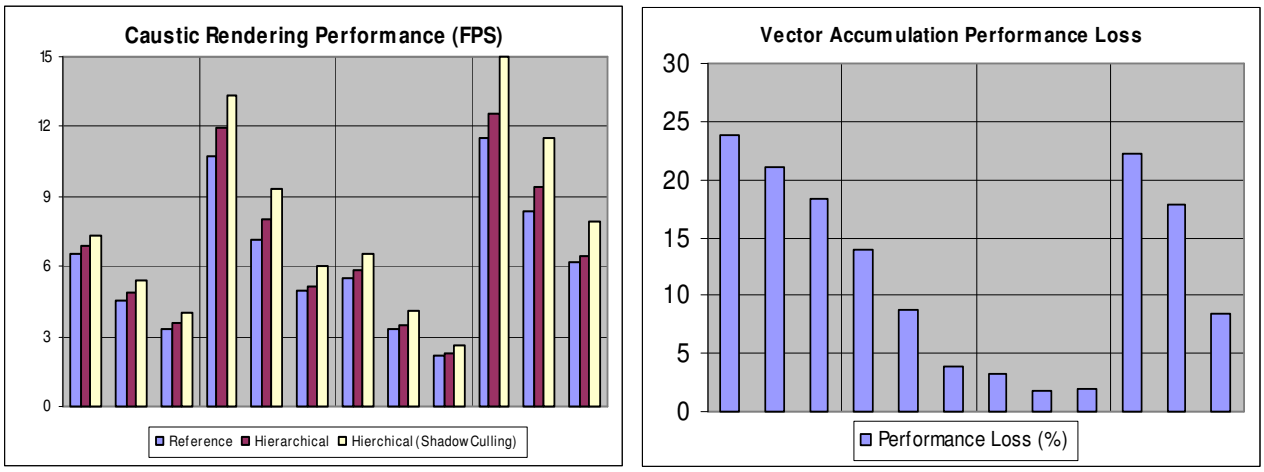


Figure 6. Left: Caustic Rendering Performance. Right: Performance Loss From Vector Accumulation. Results order as in Tables 1 and 2.

Because some of the caustic beams will no longer contribute to the final image, we can obtain a performance boost in scenes with occluded caustics by using the hierarchical structure to detect heavily shadowed regions of the water surface, and eliminating the caustic beams in these regions. Depending on the scene, this can significantly reduce the number of caustic beams which need to be rendered. The bottom half of Figure 4 illustrates the benefits of shadow culling.

Our shadow culling algorithm works by estimating the fraction of shadowed surface area for each node in the hierarchy. We will refer to this fraction as the *shadow ratio*. We define a *shadowing tolerance* S_{min}, which is the minimum shadow ratio at which to cull a leaf node. For each node, we render the top of the parent beam against the shadow map with the depth test set to GREATER, and use hardware occlusion queries to count the occluded pixels. This number is then divided by the estimated pixel coverage of the beam to produce the shadow ratio. If the shadow ratio for a parent beam is low enough that none of its descendents could have a ratio greater than S_{min}, we assume that all beams have a significant visual contribution and do not test any further. Otherwise, the process is repeated for each child node. If any leaf nodes are found whose shadow ratio is greater than S_{min}, then the beams under this leaf are assumed to contribute very little to the final image and are culled.

The accuracy of the shadows which are produced by the algorithm can be controlled by adjusting the value of S_{min}. Lower values will cause more beams to be culled, which lowers the rendering cost per frame, but may also reduce the accuracy of the caustics by incorrectly culling beams which are not shadowed themselves, but which reside under a heavily shadowed parent. Higher values of S_{min} can produce more accurate caustics, but will cause fewer beams to be culled, which means a lower performance gain.

Shadow culling alone could be used to mimic the effect of shadow maps on GPUs without vertex texture support. If the occluders have relatively simple shapes, convincing shadows can still be produced this way, given a high enough S_{min} value. However, for a very complicated occluder such as a tree, shadow culling alone will almost certainly produce visual artifacts, regardless of the tolerance. In these situations, vertex textures are needed to handle the intricate structure of the shadows. It is worth noting, however, that even if some beams are culled incorrectly, the visual effects can often go unnoticed in a dynamic scene with animated caustics, and the performance boost that is gained by culling may be of greater value than the increased realism that would be gained by not culling.

4 Performance Analysis

We conducted a series of experiments to compare the performance of our hierarchical clustering algorithm to that of a non-hierarchical reference implementation. These tests used four views of our test environment. Test scenes one, two, and three are shown in Figure 1, and scene four is shown in Figure 3.

The non-hierarchical method performs clamping and culling operations on each caustic beam individually, while the hierarchical implementation uses the techniques described above. For each test scene, we measured the frame rates using the non-hierarchical algorithm, the hierarchical algorithm without shadow culling, and the hierarchical algorithm with shadow culling. The tests were conducted three times for each scene, using water mesh sizes of 120x120, 160x160, and 200x200 vertices. The value of S_{min} used for shadow culling was 65%.

The test application was developed in C++ using Direct3D 9.0c. The test views were rendered at a resolution of 1024 by 768 pixels on a PC with a 3.2 GHz Pentium 4 CPU and a NVIDIA GeForce 6800 GPU (AGP). In most test scenes, graphics driver overhead and pixel fill rate were the most significant factors in the rendering performance.

4.1 Clamping and Culling Performance

Table 1 shows the frame rates achieved by our test application with beam rendering calls disabled. In these tests, all of the usual beam clamping and culling computations are performed, but the state changes and rendering commands required to actually draw the caustic beams are skipped. The rest of the scene is still rendered, but the frame time is dominated by the beam processing.

These results clearly demonstrate that our hierarchical algorithm can perform clamping and culling much more efficiently than a non-hierarchical technique.

The performance gains for each test scene appear to be correlated with the fraction of the caustic beams which are visible in each scene. In these test scenes, the percentages of visible beams are 50%, 35%, 100%, and 25% (from left to right). We attribute this to the fact that, in our implementation, beam-box intersection and clamping are much more expensive than frustum culling, and these operations are performed only on visible beams. In addition, the first and third scenes have a larger number of visible underwater objects, which also increases the clamping workload. Scenes with a larger clamping workload will derive much more benefit from our hierarchical data structure.

4.2 Caustic Rendering Performance

Table 2 shows the frame rates achieved when rendering the caustics. The results are graphed in Figure 6. The data in these figures indicate that our hierarchical algorithm can yield a performance improvement over non-hierarchical rendering, but the improvement appears to degrade as the mesh size and scene complexity increase.

We attribute this to the fact that driver overhead causes graphics hardware to perform much less efficiently when a large number of small rendering commands are issued, with interleaved state changes. When our mesh size is increased, more beams are rendered, and our frame rate is dominated by the cost of repeatedly changing the stencil state and issuing rendering commands. This eventually negates any performance improvement that is obtained from the hierarchical data structure. Future changes to the DirectX driver model may lower the cost of repeated state changes and reduce the effect of driver overhead on our rendering performance [Boyd 2003].

Shadow culling consistently yields an additional performance boost by reducing the number of beams which need to be rendered. In our test scenes, the number of beams that were culled in the shadow culling phase was typically about ten times the number of occlusion tests that were performed.

Vector accumulation lighting, because it consumes more frame buffer bandwidth, incurs a performance loss ranging from 3% to 25%. The smaller performance loss at higher mesh sizes is due to the fact that driver overhead starts to become a more significant bottleneck than pixel fill-rate. Although it will never completely disappear, we expect this performance cost to be reduced as GPUs continue to evolve and floating point blending becomes more widely used.

5 Summary, Suggestions, and Future Work

We have presented an improved caustic rendering algorithm which supports fully dynamic environments, and can project caustics onto any type of object. Our implementation also takes into account shadows in the caustics from objects above the water, and is able to exploit these shadows for improved performance by using shadow culling.

Our performance analysis allows us to conclude that hierarchical clamping and culling is more efficient than a non-hierarchical implementation, and that our hierarchical shadow culling can result in a significant performance boost by eliminating the rendering of shadowed beams. In addition, we have shown how floating point blending can be used to achieve a more accurate implementation of the caustic lighting model.

There are a number of potential improvements and optimizations that could be attempted to make our caustic rendering system more robust. Although shadow culling can provide a performance boost in heavily shadowed scenes, it is possible that the occlusion testing will result in a net performance drop in scenes with little or no shadowing. A more robust implementation of our technique should check for this case and avoid shadow culling in situations where the performance gains would be minimal. A simple bounding box overlap test between the projections of the water and the occluders in light space should be sufficient. Depending on the scene and the viewpoint, occlusion culling of the underwater objects may also be desirable as a means of reducing the beam clamping workload. It may also be possible to implement beam-box testing and clamping in the vertex shader, by packing the visible bounding boxes into shader constants and using data-dependent looping. Another possibility, which will require further research, is to attempt to estimate the contribution of each visible, non-shadowed beam to the final image, and cull beams which contribute little or no energy.

Another idea for future research is to apply our hierarchical method to shadow volume rendering. Although this work has concentrated on caustic rendering, our hierarchical clamping and culling was, in part, inspired by a recent work on shadow volumes [Lloyd et al. 2004]. We believe that a hierarchical organization like ours could substantially improve the performance of their techniques in complex scenarios, such as shadows cast by a picket fence or the leaves on a tree.

Other possibilities for future work include extending the algorithm to efficiently render reflective caustics from arbitrary objects, such as the canonical metal ring on a table, or developing a new algorithm to allow for caustics caused by multiple scattering events, such as by refraction through a glass ball.

6 Acknowledgements

Removed for review.

7 References

ARVO, J. Backward Ray Tracing, *Developments in Ray Tracing*, SIGGRAPH `86 Course Notes, Volume 12, August 1986.

BOYD, CHAS. Future Features. *Presented at Microsoft Meltdown 2003*. Notes online: http://www.microsoft.com

ENGEL, K., KRAUS, M. ERTL, T., High Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. *Graphics Hardware 2001*, pp. 9 - 17.

HEIDMANN, T. Real Shadows, Real Time. In *Iris Universe*, vol. 18, 1991. Silicon Graphics, Inc., pp. 23 - 31.

IWASAKI, K., DOBASHI, Y., NISHITA, T. Efficient Rendering of Optical Effects within Water Using Graphics Hardware. *Pacific Graphics 2001*, pp. 374--383, 2001.

IWASAKI, K. DOBASHI, Y., NISHITA, T. A Fast Rendering Method for Reflective and Refractive Caustics due to Water Surfaces. *Computer Graphics Forum (Eurographics 2003)*, 25(3), September 2003.

JENSEN, H.W. Global Illumination using Photon Maps. In *Rendering Techniques '96*. pp. 21-30, Springer-Verlag, 1996.

JENSEN, L. S., GOLIAS, R. Deep Water Animation and Rendering. *Presented at Game Developer's Conference Europe 2001.*

LOVISARCH, J. Complex Water Effects at Interactive Frame Rates. *The 11th Annual Conference on Computer Graphics, Visualization, and Computer Vision*, 2003.

LLOYD B., WENDT J., GOVINDARAJU N.,MANOCHA D.: CC shadow volumes. *In Proceedings of the Eurographics Symposium on Rendering, 2004*.

NISHITA, T. NAKAMAE, E. Method of Displaying Optical Effects within Water using Accumulation Buffer," *Proc. SIGGRAPH' 94*, 1994, pp.373-380.

PURCELL, T.J. DONNER, C. CAMMARANO, M. JENSEN, H.W. Hanrahan, P. Photon Mapping on Programmable Graphics Hardware *Graphics Hardware 2003*, pages 41-50, San Diego, July 2003.

SLOAN, P.P, KAUTZ, J, SNYDER, J. Precomputed Radiance Transfer for Interactive Rendering in Dynamic, Low Frequency Lighting Environments. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques.* pp. 527-536, San Antonio, 2002.

STAM, J. Random caustics: Natural Textures and Wave theory revisited. *ACM SIGGRAPH 96 Visual Proceedings* pp. 150, 1996.

TRENDALL, C., STEWART, A.J.: General calculations using graphics hardware, with application to interactive caustics. In *Rendering Techniques 2000*, 287-298, Springer, 2000.

WAND, M. STABER, W. Real-Time Caustics. *Computer Graphics Forum (Eurographics 2003),* 25(3), September 2003

WATT, M. Light-Water Interaction using Backward Beam Tracing. *Proc. SIGGRAPH'90*, 1990, pp.377-385.

WYMAN, C. HANSEN, C. SHIRLEY, P. Interactive Ray-Traced Caustics. University of Utah Tech. Report, April 23, 2003.

WYMAN, C. HANSEN, C. SHIRLEY, P. Interactive Caustics Using Local Pre-Computed Irradiance. *Proceedings of the 2004 Pacific Conference on Computer Graphics and Applications*. 143-151.