Realistic Real-time Underwater Caustics and Godrays

C. Papadopoulos [1,2], G. Papaioannou [1]
[1]Department of Informatics, Athens University of Economics and Business, Athens, Greece
[2]Department of Computer Science, State University of New York at Stony Brook, New York, USA

Abstract

Realistic rendering of underwater scenes has been a subject of increasing importance in modern real-time 3D applications, such as open-world 3D games, which constantly present the user with opportunities to submerge oneself in an underwater environment. Crucial to the accurate recreation of these environments are the effects of caustics and godrays. In this paper, we shall present a novel algorithm, for physically inspired real-time simulation of these phenomena, on commodity 3D graphics hardware, which can easily be integrated in a modern 3D engine.

Keywords: *I.3.7, Three-Dimensional Graphics and Realism, Color, shading, shadowing, and texture.*

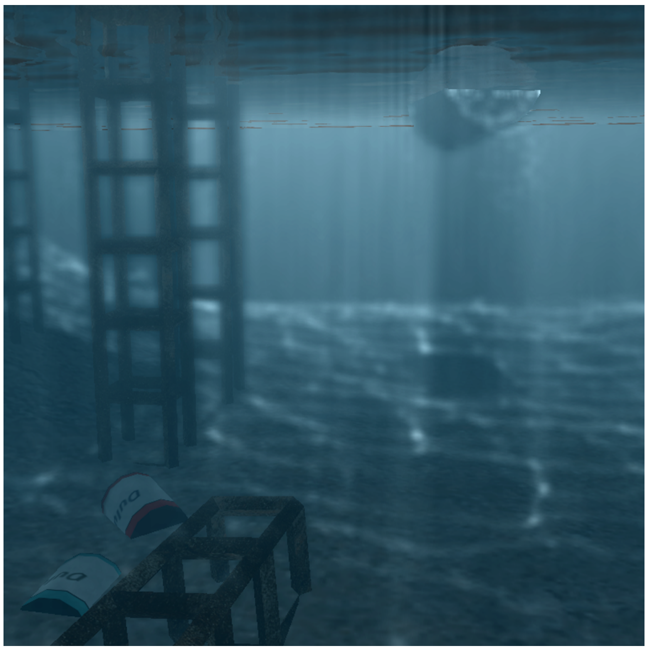


Figure 1: A screenshot of our caustic and godray creation algorithm, running at an excess of 100 fps, at a resolution of 800x800.

1. INTRODUCTION

Caustics are a result of the convergence of light at a single point. The phenomenon occurs when light interacts with a reflective or refractive surface, where rays deviate from their initial directions and focus on certain regions. *Crepuscular rays* or *godrays* are a result of photons outscattering from their path due to the presence of particles in a participating medium. In this paper we shall present a novel method for a physically inspired simulation of these two phenomena in real time (Figure 1).

2. PREVIOUS WORK

Even though the underwater illumination has been the subject of extensive research (mostly in the domain of particle and ray tracing algorithms), real-time solutions to the matter of realistic caustic formation are limited. Methods for the simulation of crepuscular rays (or godrays) are even sparser. In this section, a brief overview of some of the existing work in the area will be presented, which is mainly focused on offline rendering or interactive illumination simulation, but also includes real-time techniques for modeling and displaying caustics and godrays.

Offline solutions can create extremely realistic underwater caustics and godrays, at a significant computational cost. One of the first methods that allowed for caustic creation is forward ray tracing [1], which differs from conventional ray tracing in the sense that rays are cast from the light towards the scene. Rays that intersect the camera clipping plane contribute to the intensity of the respective pixel. An alternative (and faster) method is bidirectional ray tracing [2],[3], that traces rays both from the camera and the light source and 'connects' the ray paths to find the radiance contribution to the corresponding pixel. Photon mapping, a global illumination solution presented in [4], utilizes forward ray tracing along with probabilistic techniques, importance sampling and an illumination gathering step to achieve relatively high performance and high quality caustics. It also permits the creation of godrays in participating media (via ray marching). However, despite the improvements when compared to conventional forward ray tracing, photon mapping is also limited to off-line applications.

One of the first proposals for real-time caustic creation was by [5] and involved the projection of a precomputed texture onto the scene geometry using additive blending. Animation was achieved via texture coordinate perturbation. Obviously, while being fast, this method did not produce physically correct caustics that animated in tandem with the water surface. Another approach, by [6] assumes that caustics are formed by rays emanating from the water surface directly above the point of interest and uses Snell's law to produce refracted rays. It modulates intensity based on the direction of these rays and alternatively uses the vectors to index a light map. Obviously the caustics produced by this method are not physically correct, as a result of the above assumption.

Shah et al. [7] presented a novel method for creating refractive caustics in image-space. This method involves creating a vertex grid out of the refractive geometry and then splatting the vertices onto the scene using an image-space ray-scene intersection algorithm. However caustic quality is directly related to the tessellation of this vertex grid. Furthermore, this method does not provide a solution to the issue of godrays that is part of the same illumination problem.

Research concerning godray creation falls into two main categories, the first being the modeling of the rays as light shafts. [8] presented a method that modeled lightshafts as the front-facing surfaces of 'illumination volumes'. However, the computational cost for this method was very high and, at its time, it did not provide a real time solution. Lanza [9] proposed a similar method that used parallelepipeds, which are transformed via a vertex shader program so as to animate in tandem with the water surface. This method, while producing very 'dramatic' results, requires a high density parallelepiped grid in order for the effect to be realistic, which in turn results in a very high fill rate cost. Furthermore, the final intensity computation for the godrays does not account for individual ray attenuation, as it is done in a post-processing pass, during which, information for each individual ray is not available.

The second category of godray creation algorithms focuses around the sampling of the visible distance in front of the viewer by rasterizing planar surfaces parallel to the near clipping plane. Dobashi et al. [10] uses precomputed integral values (saved in 2D textures) in order to compute the intensity value at each plane, which are then accumulated in the frame buffer. Jensen [11] performs a per-fragment projective texture read from a caustics map, for each fragment on a sampling plane. Both of these methods present one main drawback, since a large number of sampling planes is required in order to avoid artifacting. As these planes are represented by screen-space quads, there is a significant fill-rate premium to pay for their rasterization. Furthermore, in the case of Dobashi et al., there is no direct way for the algorithm to control animation of the light shafts (since it would require recomputation of the 2D lookup textures), while as far as Jensen's method is concerned, the result is not physically accurate. In addition, the non-linear nature of these projection transformations on the view-oriented planes produces non-uniformly spaced samples and may result in curved godrays.

Another type of method that can handle caustics and scattering due to complex refractive objects involves ray-marching through a voxel grid of the refractive geometry. 'Eikonal rendering' [12] is such a technique, and while it allows for real-time framerates during viewport changes, if the refractive object or the light position are altered, the lighting distribution has to be recalculated (a process that can take several seconds) making it unsuitable for simulating underwater effects due to their constantly shifting nature. Another technique in this area is presented by Sun et al.[13] who use an oct-tree data structure to store voxel data along with an adaptive photon-tracing step to recompute the radiance volumes at interactive frame rates (approximately 8 fps). However, while successfully dealing with arbitrary and non-uniform refractive geometries, this method also is too computationaly intensive to be used in a real-time generic 3D application.

Finally, Krüger et al. [14] proposed a method for generic caustic simulation that utilizes the rasterization of lines compacted as texture rows to compute the intersection points of photon rays with scene geometry. While this method can be quite accurate when all occluders lie inside the camera frustum and can also handle multiple bounces via depth peeling, it fails to calculate intersection points for off-frustum occluders, which is frequently the case in underwater scenes. Furthermore, the brute-force texture-space intersection algorithm imposes a high fill-rate cost and requires multiple passes to calculate an adequate number of particle collisions, as the data for each particle intersection estimation effectively occupies one texture row (texture space scan-line).

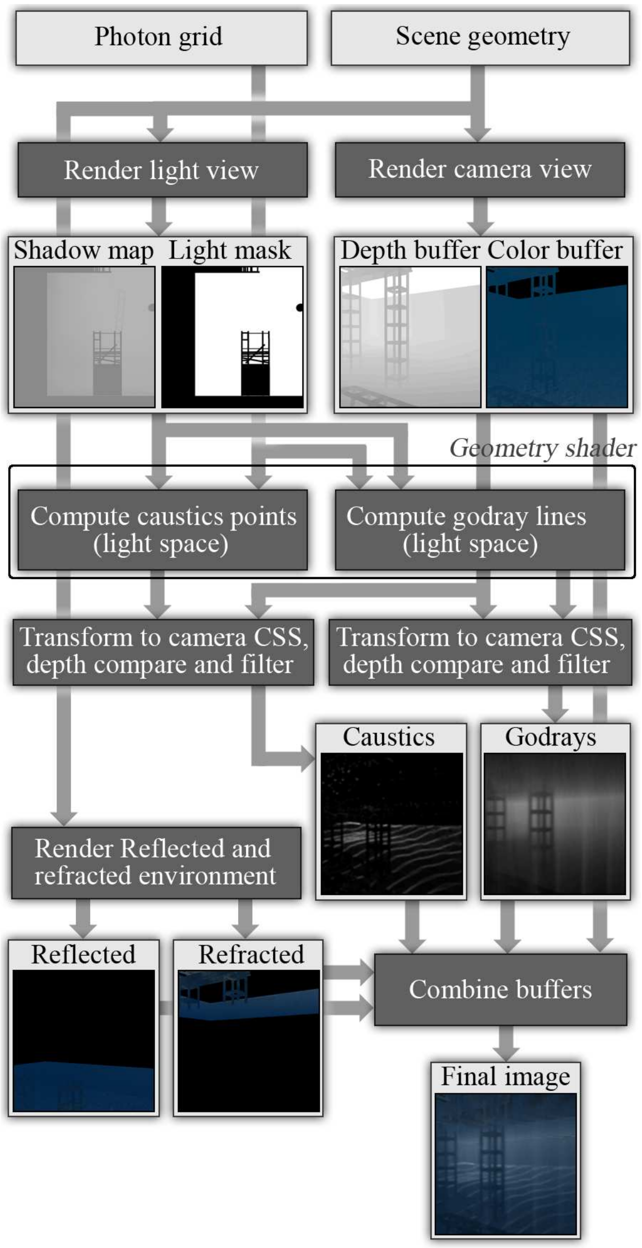
Our method is a real-time approximation of the photon mapping algorithm for underwater caustics (and volumetric caustics) generation. It utilizes the image-space ray-scene intersection method by Shah et al. [7], but in contrast to the original technique, it decouples the effect accuracy from the refractive geometry by tracing uniformly distributed photons from the light source towards the scene. Caustics are modeled as point primitives of variable size created on the intersection points, while scattering due to the participating medium is simulated using line rasterization between the water surface and these points. Intensity calculations are made on a per-photon basis (allowing for realistic attenuation and outscattering). Finally the equivalent of the photon mapping gathering step is a filtering pass that spreads the particle contribution over a certain area and reduces aliasing. Our technique achieves very high framerates on commodity graphics accelerators and can be easily integrated within any modern 3D engine.

3. METHOD OVERVIEW

Our method realistically approximates caustic creation, by casting photons from the light source evenly distributed over a grid. These photons are intersected against the scene geometry, and point primitives of variable size are created at the intersection points. For godrays, the intersection points are discovered in a similar way, but line primitives are spawned instead. Our method utilizes a deferred rendering approach and makes extensive use of render-to-texture and programmable shader capabilities of modern graphics hardware (vertex/fragment and geometry shaders). Following is a high-level overview of the rendering pipeline:

1. Frame preparation
   a. Calculate the new position and look-at vector for the rendering pass from the light's perspective.
2. Shadow map creation
   a. *(In light-space)* Render the shadowmap.
   b. *(In light-space)* Optionally, render a mask representing the water surface into the shadowmap's color attachment, allowing for the algorithm to test photons against the water's surface only and therefore eliminate those that do not enter the water mass.
3. Render scene geometry
   a. *(In camera-space)* Render the scene geometry into a render target, from the camera's point of view.
4. Optional reflection/refraction rendering passes
5. Caustics Rendering
   a. *(In light-space)* Calculate intersection points of rays with the scene geometry.
   b. *(In camera-space)* Emit point primitives and rasterize them with additive blend.
   c. *(In camera-space)* Filter caustics.
6. Godray Rendering
   a. *(In light-space)* Calculate intersection points of rays with the scene geometry.
   b. *(In camera-space)* Emit line primitives and rasterize them with additive blend.
   c. *(In camera-space)* Filter godrays.
7. Compose final image
   a. *(In camera-space)* Use additive blending to compose the final image using input from steps 3,4,5 and 6.
   b. *(In camera-space)* Optionally apply other screen-space post processing effects like ambient occlusion and motion blurring.

Since our goal was to focus around underwater caustics and godrays, the algorithm supports the tracing of a single refraction. It can however be extended to support multiple refractive interfaces via depth peeling ([7], [14]). A visual representation of the algorithm process that demonstrates how results from each step are combined, is presented in Figure 2.



Photon grid
Scene geometry
Render light view
Render camera view
Shadow map
Light mask
Depth buffer
Color buffer
Geometry shader
Compute caustics points (light space)
Compute godray lines (light space)
Transform to camera CSS, depth compare and filter
Transform to camera CSS, depth compare and filter
Caustics
Godrays
Render Reflected and refracted environment
Reflected
Refracted
Combine buffers
Final image

Figure 2: Overview of the godray and caustic creation algorithm.

3.1 Frame Preparation

Since we cast a concentrated but small number of photons onto the geometry, we must ensure that most of them intersect the visible portion of the scene. Therefore, the light frustum (and the photon grid) is bound to follow the camera frustum.

For directional (infinite) light sources, their position $\mathbf{p}_{light}$ tracks the camera frustum, whereas for point lights, the position remains fixed. In both cases though, the look-at vector $\vec{\mathbf{l}}$ points at the middle of the camera view frustum $\mathbf{p}_{mid}$.

$\vec{\mathbf{l}} = \mathbf{p}_{mid} - \mathbf{p}_{light}$

$\mathbf{p}_{mid} = \mathbf{p}_{viewer} + \frac{z_{near}+z_{far}}{2} \cdot \vec{\mathbf{l}}_{viewer}$

where z_{near}, z_{far} are the camera's near and far clipping distances and $\vec{\mathbf{l}}_{viewer}$ is its look-at vector.

3.2 Render Scene Geometry

In this step we render the scene geometry into an off-screen buffer. If necessary, fog calculations can also be applied here. Furthermore, the Z-buffer information from this pass is captured, as it is required in the following steps to depth-test the caustics and godrays.

3.3 Caustics Rendering

We have mentioned that our process involved casting a photon grid onto the scene. In this way, the emission of photons from the light source towards the scene is simulated. The grid is modeled in the light's canonical screen space as an array of points, with the required tessellation. During this step of the algorithm, a low resolution grid is sent to the GPU for rendering. Then, using a geometry program, each grid cell is subdivided to produce the desired number of points, ensuring a dense photon distribution. An alternative adaptive subdivision scheme has been proposed by Wyman et al. in [15]. However, usage of this scheme is not justified in our method, since it requires feedback during each subdivision step and is better suited for generic caustics simulations in which the refractor has limited screen-space coverage and possesses an arbitrary shape. The geometry shader also performs the refraction, point splatting and godray modeling operations. The grid point currently being calculated is unprojected from light canonical screen space coordinates into world coordinates and modeled as a ray ($\vec{r_i}$), which is then intersected against the water surface. With the surface intersection point $\mathbf{p_s}$ and normal $\vec{\mathbf{n_s}}$ of the water surface known, $\vec{\mathbf{r_i}}$ is refracted to produce the refraction direction $\vec{\mathbf{r_t}}$. Both $\vec{\mathbf{r_t}}$ and $\mathbf{p_s}$ are passed to the image space intersection algorithm by Shah et al. [7], along with the shadow map texture. The algorithm uses the Newton-Rhapson (NR) derived iterative method to approximate the solution to a function $f(d)$ with d being the distance to intersection point $\mathbf{p_i}$ from $\mathbf{p_s}$ along $\vec{\mathbf{r_r}}$. In order to do so, an estimated intersection point $\mathbf{p_e} = \mathbf{p_s} + d_{estimate} * \vec{\mathbf{r_t}}$ is assumed, along with its projection into light-space $\mathbf{p_{proj_{p_e}}}$ and the algorithm approximates the solution to $f(d) = \mathbf{p_e} - \mathbf{p_{proj_{p_e}}}$. In [7], it is shown that this NR-derived estimator tends to converge to a value of d so that $\mathbf{p_s} + d * \vec{\mathbf{r_t}}$ is the surface intersection point $\mathbf{p_i}$. At $\mathbf{p_i}$ we then emit a point primitive. A schematic overview of the photon casting process can be seen in Figure 3.

These primitives are transformed to camera space and rendered using additive blending with their size corresponding to their screen-space coverage. Consiquently, point size is actually a significant factor in computation of the final caustics intensity. If all points emitted have a constant size then the projection of the distant point primitives would overlap on the view plane, resulting in much brighter caustics than the ones close to the camera. On the other hand, if the point size is not large enough, photons rasterized close to the viewer do not superimpose each other, leading to inadequate splatting that cannot simulate the gathering stage of particle tracing and produces noise artifacts. The solution to this issue is to regulate the size of the points based on their distance from the camera (see Figure 4). The final point size is calculated as follows:

$s_{final} = a + b/d_{pointFromViewer}$

with $a = s_{max} - \frac{z_{far}\cdot(s_{max}-s_{min})}{z_{far}-z_{near}}$

and $b = \frac{z_{near}\cdot z_{far}\cdot(s_{max}-s_{min})}{z_{far}-z_{near}}$

S_{max} and S_{min} are minimum and maximum point sizes.

The emitted point primitives are then rasterized (with additive blending) in a high-accuracy render target and Z-tested against the camera's depth buffer (already available from step 3). The final

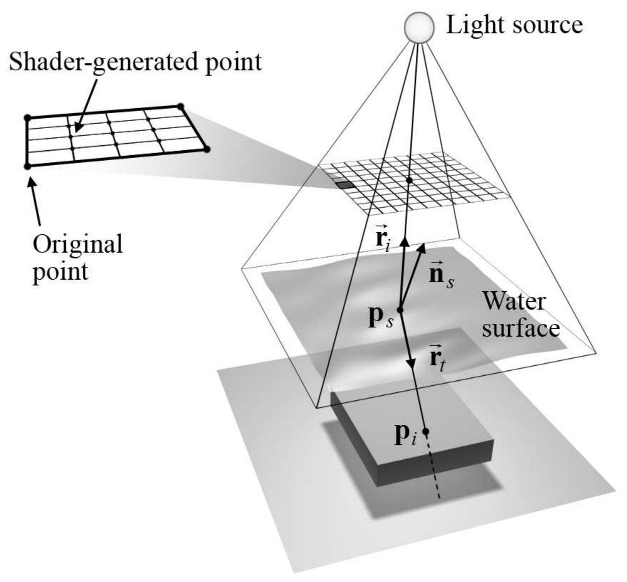


Figure 3: The photon casting process.

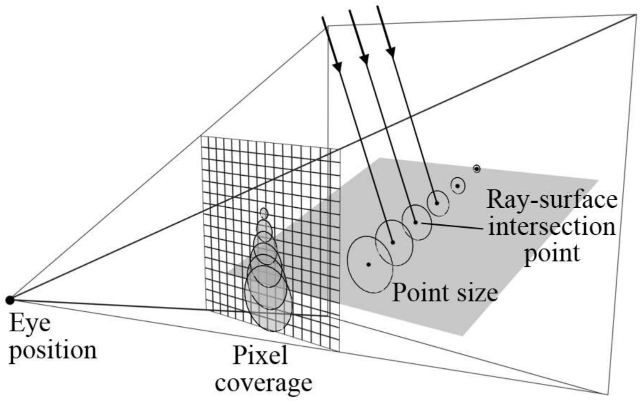


Figure 4: Distance-regulated point size to account for the non-projective hardware-based drawing of point primitives.

intensity value is produced from this formula:

$I_{final} = I_{photon} \cdot e^{-\gamma \cdot d_{fromSurface}}$

where γ is the medium attenuation parameter and $d_{fromSurface}$ is the distance of the photon from the water surface. This formula only takes $d_{fromSurface}$ into account, since the attenuation based on the distance from the viewer is intrinsically handled via the point size regulation described above.

3.4 Godray Rendering

In this step, a grid of rays is cast and intersected with the scene (similar to step 5). The main difference lies in that, instead of emitting a single point primitive at the ray-scene intersection point $\mathbf{p_i}$, a line primitive that starts at water surface $\mathbf{p_s}$ and ends at $\mathbf{p_i}$ is emitted. The line primitives are transformed to camera-space, then rasterized with additive blending in a high-accuracy render target and Z-tested against the camera's depth buffer. The formula that produces the final intensity value per godray fragment is the following:

$I_{final} = I_{photon} \cdot Mie(\theta) \cdot e^{-\gamma \cdot d_{fromViewer}} \cdot e^{-\gamma \cdot d_{fromSurface}}$

$Mie(\theta)$ is a Mie scattering phase function with θ being the angle between the ray direction and the vector from the viewer to the fragment. $d_{fromSurface}$ is the distance of the fragment from the water surface.

3.5 Filtering and Composition

In both of the above steps, the result can display some aliasing in the low intensity areas. To counter this issue, after calculating the effects, we apply a multi-tap low pass filter with a rotating sampling kernel that reduces noise. Comparison between the filtered and unfiltered results can be see in Figure 5.

Finally, the intermediate images (color/reflection/refraction/filtered godray/filtered caustics buffers) from the above steps are combined into the end result. At this point, the water surface geometry is rendered as well. The final product of our algorithm can be seen in Figures 6,7,8.

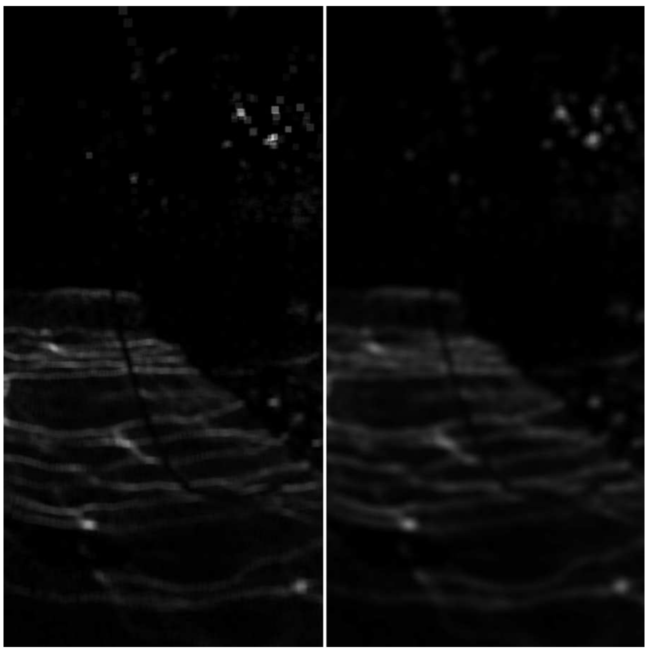


Figure 5: Detail demonstrating the differences between unfiltered (left) and filtered caustics (right).

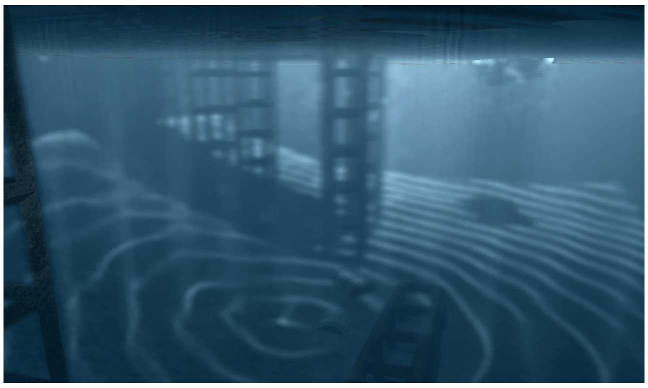


Figure 6: Our algorithm, rendering a small port, running at a resolution of 1440×850, with a framerate of 60+ fps.

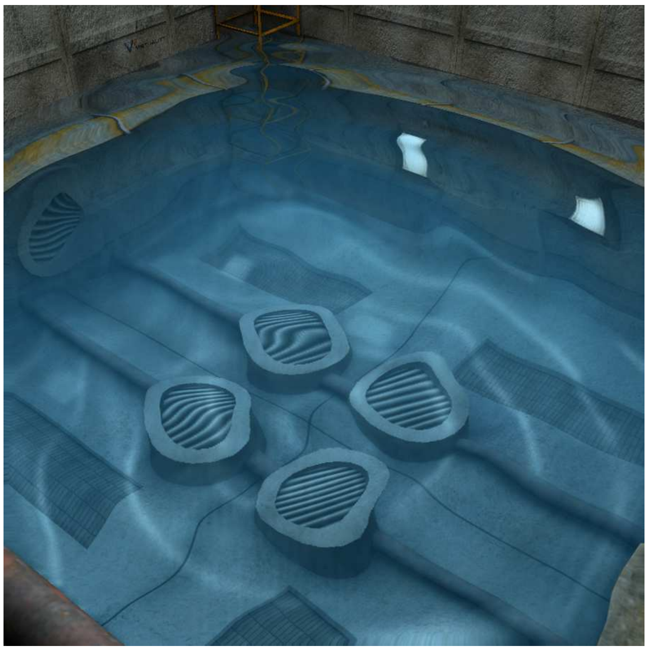


Figure 7: A water tank scene rendered with our algorithm. The scene runs at more than 120 fps in a window of 800×800 pixels.

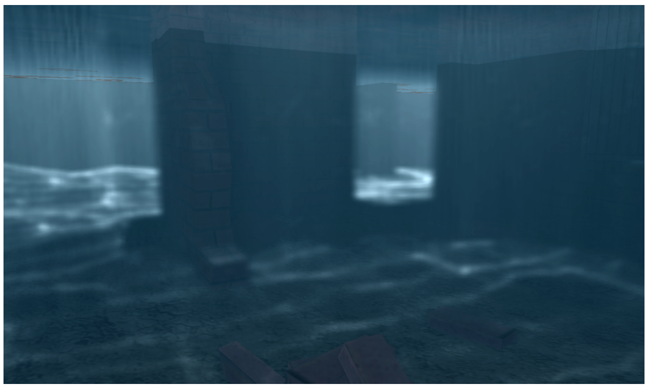


Figure 8: A flooded storage area scene rendered with our algorithm in a 1440×850 resolution. The scene runs at more than 60 fps.

4. IMPLEMENTATION AND RESULTS

4.1 Implementation Details

We have implemented the caustic and godray formation algorithm using OpenGL and GLSL. The render targets are implemented as OpenGL Frame Buffer Objects with textures of various sizes bound to the respective attachment points.

In order to improve performance, we regulate the sizes of the render targets to a fraction of the final frame buffer resolution. Specifically, in our demo application the caustic render target resolution is 512×512 (compared to the 800×800 default resolution for the main viewport), whereas the godray render target is 128×128 pixels. This results in a significant performance boost (since the godray rendering process is fill-rate intensive) with only a small cost in accuracy. If the application viewport is resized, the back-buffers are also resized to preserve this scale factor.

In our implementation, the photon grids are initialized on application startup and rendered during each frame. The volumetric nature of the godray effect requires a smaller amount of cast photons when compared to the caustic effect in order to achieve satisfactory detail. Thus, two separate photon grids are created, with the first one being 200×200 points large, and the second one being 100×100 points. Inside the geometry shader, these points can be further tessellated in order to improve detail. Dynamic tesselation is possible but greatly impacts peformance (by as much as 50%) as the shader compiler is unable to optimize the shader code by unwinding the primitive generation loop.

For our test cases (and the respective demo application) we have created a single light positioned at a very large heigth above the scene in order to simulate a directional light source with an intensity of $9.2\frac{w}{sr}$. The exponential attenuation factor γ used is $0.13m^{-1}$. Also, we have utilized a simplified version of the Henyey-Greenstein phase function [16] in order to compute the $Mie()$ term.

We mentioned in the previous section that a crucial part of caustic production is the regulation of the size of the generated points. In our implementation we have set a minimum point size of 5 and a maximum of 20. These numbers can be adjusted according to the resolution of the cast photon grid (with lower resolutions requiring larger point sizes in order to compensate). Similarly, the width of the lines spawned in the godray portion of the algorithm has to be modified to compensate for a reduction in grid resolution.

In our tests, we provide two different models for the procedural generation of water surface elevation and normals. The first one is a simple circular cosine function with a small noise contribution that is read off a Perlin Noise [17] texture, that provides a familiar and easily comparable effect. The second model consists of a cosine value along one of the world axes with a significant Perlin Noise contribution, approximating the turbulent waves of a water surface. In both models, the amplitude of the water surface can be regulated by the user during run time.

4.2 Results / Tests

We conducted our tests on a system with an Intel Core 2 Duo E4500 processor running at 2.2 Ghz, with 2 gigabytes of ram and an NVidia GTX260 GPU with 216 stream processors. With a viewport resolution of 800×800 for the main window, with a 512×512 caustics buffer, a 128×128 godray buffer and grids for the caustics and godrays of $200 \times 200 \times 4$ and $100 \times 100 \times 4$ photons respectively, the frame rate exceeds 110 fps. At a viewport resolution of 1440×850, while maintaining the same grid resolutions, the frame rate still remains highly interactive, exceeding 60 fps.

If the grid resolutions are increased to $500 \times 500 \times 4$ (for a total of 1000000 photons) for the caustics portion and to $300 \times 300 \times 4$ (for a total of 360000 photons) for the godrays portion, the framerate still remains interactive on our test system (ranging from 20 to 30 fps). At these resolutions, aliasing effects are no longer noticeable, making the filtering passes unnecessary (as can be seen for the godray portion of the algorithm in Figure 9).

In order to test how the algorithm scales with respect to back-buffer resolution, we performed measurements with a viewport resolution of 1440×850, while increasing the godray back-buffer to 460×272 (a 4-fold increase in resolution compared to the default setting). To compensate, we also increased the screen-coverage of the line primitives to 12 pixels. Finally, we augmented the grid resolutions to $400 \times 400 \times 4$ for caustics generation and $200 \times 200 \times 4$ for godrays and regulated the point primitive size (smallest 2 pixels, largest 10 pixels) to sharpen the caustics. Despite the increase in back buffer resolution and fill-rate cost, the frame-rate still remains highly interactive (20 frames per second). Again, with this increase in back-buffer size and grid resolutions, filtering of the effects becomes unnecessary. Results of this configuration can be seen in Figure 10.

The screen captures presented in this paper have had their color balance and gamma slightly enhanced in order to improve legibility on printed media.

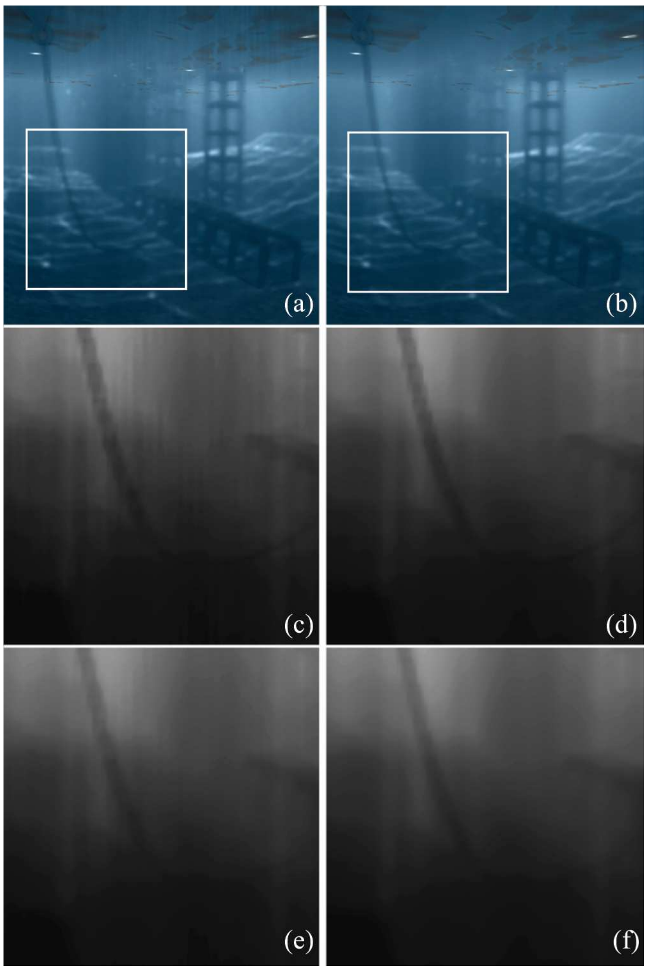


Figure 9: Figure demonstrating the results of the godray portion of the algorithm with different grid resolutions. Left column: (a) Final result with 40000 photons. (c) unfiltered godrays. (e) post-filtered godrays. Right column: (b) Final result with 360000 photons. (d) unfiltered godrays. (f) post-filtered godrays.

5. CONCLUSIONS

We have presented a novel algorithm for the creation of real-time underwater caustics and godrays. The algorithm achieves realistic results at high framerates on consumer graphics hardware. Due to the utilization of rendering stages and buffers encountered in modern graphics engine implementations, our method can be easily integrated into any 3D engine.

6. ACKNOWLEDGEMENTS

The work presented in this paper is funded by the Athens University of Economics and Business Special Account for Research Grants (EP-1600-10/00-1).

7. REFERENCES

[1] J. Arvo, "Backwards ray tracing," in *SIGGRAPH Course Notes*, 1986, vol. 12, p. 100.

[2] E. P. Lafortune, "A theoretical framework for physically based rendering," *Computer Graphics Forum*, vol. 13, pp. 97–107, 1994.

[3] E. Veach and L. J. Guibas, "Bidirectional estimators for light transport," in *Eurographics Rendering Workshop 1994*, 1994, pp. 147–162.

[4] H. W. Jensen, *Realistic image synthesis using photon mapping*, A. K. Peters, Ltd., Natick, MA, USA, 2001.

[5] J. Stam, "Random caustics: Wave theory and natural textures revisited," in *Visual Proceedings of SIGGRAPH 1996*, 1996, p. 151, Available at: `http://www.dgp.toronto.edu/people/stam/INRIA/caustics.html`.

[6] J. Guardado and D. Sanchez-Crespo, "Rendering water caustics," in *NVidia GPU Gems*, pp. 31–44. Addison-Wesley, 2004.

[7] M. A. Shah and J. Konttinen, "Caustics mapping: An image-space technique for real-time caustics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 2, pp. 272–280, 2007, Member-Pattanaik, S.

[8] K. Iwasaki, T. Dobashi, and T. Nishita, "An efficient method for rendering underwater optical effects using graphics hardware," in *COMPUTER GRAPHICS forum*, 2002, vol. 21, pp. 701–711.

[9] S. Lanza, "Animation and rendering of underwater godrays," in *SHADERX 5*, pp. 315–327. Charles River Media, 2007.

[10] Y. Dobashi, T. Yamamoto, and T. Nishita, "Interactive rendering of atmospheric scattering effects using graphics hardware," in *Graphics Hardware*, 2002, pp. 99 – 108.

[11] L. Jensen, "Deep-water animation and rendering," 2001, Available at: `http://www.gamasutra.com/view/feature/3036/deep_water_animation_and_rendering.php`.

[12] I. Ihrke, G. Ziegler, A. Tevs, C. Theobalt, M. Magnor, and H. P. Seidel, "Eikonal rendering: efficient light transport in refractive objects," in *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, New York, NY, USA, 2007, p. 59, ACM.

[13] X. Sun, K. Zhou, E. Stollnitz, J. Shi, and B. Guo, "Interactive relighting of dynamic refractive objects," in *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, New York, NY, USA, 2008, pp. 1–9, ACM.

[14] J. Krüger, K. Bürger, and R. Westermann, "Interactive screen-space accurate photon tracing on gpus," in *Rendering Techniques (Eurographics Symposium on Rendering - EGSR)*, June 2006, pp. 319–329.

[15] C. Wyman and G. Nichols, "Adaptive caustic maps using deferred shading," *Computer Graphics Forum*, vol. 28, no. 2, pp. 309–318, 2009.

[16] L. G. Henyey and J. L. Greenstein, "Diffuse radiation in the Galaxy," *Annales d'Astrophysique*, vol. 3, pp. 117, 1940.

[17] K. Perlin, "Improving noise," in *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 2002, pp. 681–682, ACM.

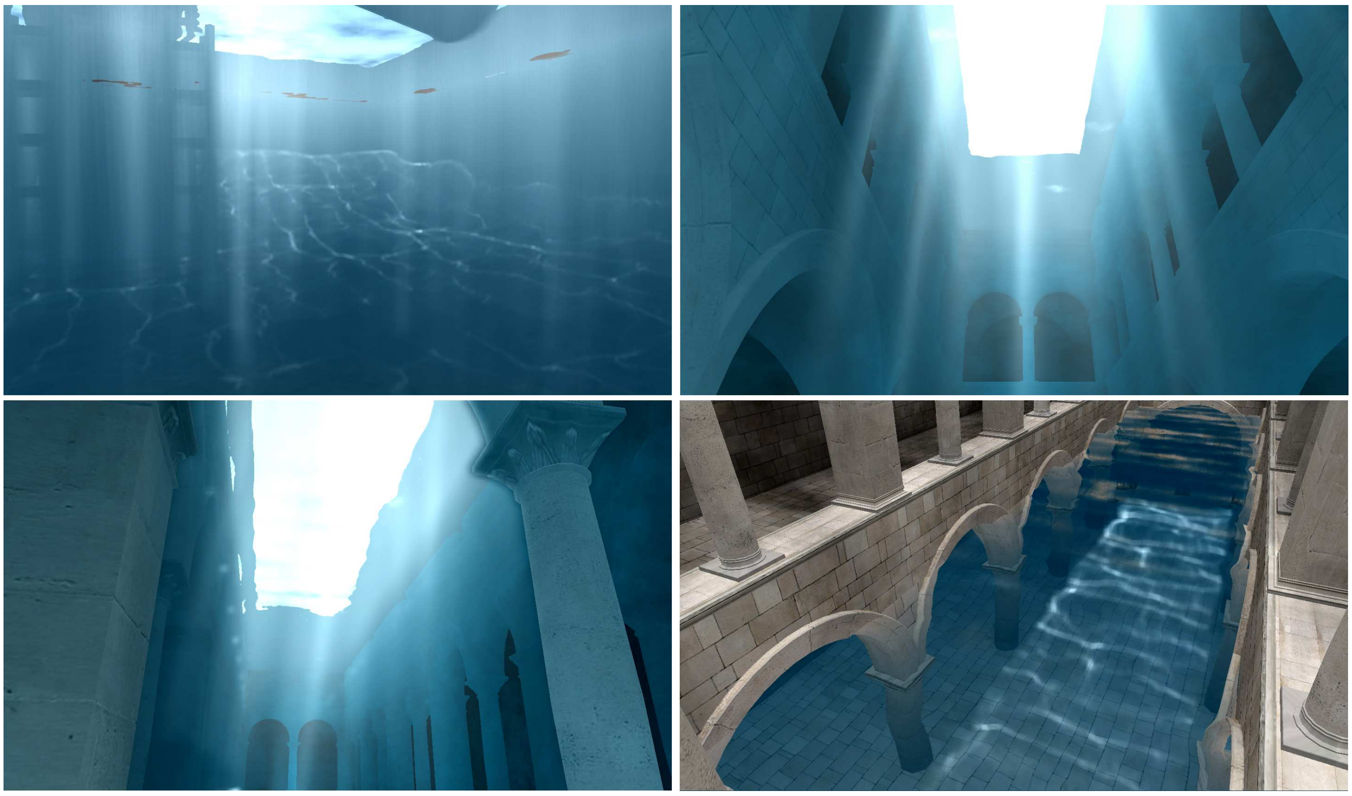


Figure 10: Final results of our algorithm with high grid resolutions, an increased godray back-buffer resolution, regulated primitive sizes and screen-space ambient occlusion approximation. For all these test cases, the method still allows for interactive frame-rates.

ABOUT THE AUTHORS

Charilaos Papadopoulos received a BSc in Informatics from the Athens University of Economics and Business and commenced his PhD studies in Computer Science at the State University of New York at Stony Brook in the Fall of 2009. His research and teaching interests focus around computer graphics, real-time photo-realistic rendering and visualization. His contact email is `papado@aueb.gr` and his personal website can be found at `http://graphics.cs.aueb.gr/users/papado`.

Georgios Papaioannou received a BSc in Computer Science in 1996 and a PhD degree in Computer Graphics and Pattern Recognition in 2001, both from the University of Athens, Greece. From 2002 till 2007 he has been a virtual reality systems engineer and developer at the Foundation of the Hellenic World. Dr. Papaioannou has been teaching elementary and advanced computer graphics, programming and human-computer interaction courses since 2002. He is currently a lecturer at the Department of Computer Science of the Athens University of Economics and Business and his research is focused on real-time computer graphics algorithms, photorealistic rendering, virtual reality and three-dimensional shape analysis. He is a member of IEEE, ACM and SIGGRAPH. His contact email is `gepap@aueb.gr`.