

Fast Real-time Caustics from Height Fields

Cem Yuksel · John Keyser
Texas A&M University

Received: date / Accepted: date

Abstract Caustics are crucial in water rendering, yet they are often neglected in real-time applications due to the demanding computational requirements of the general purpose caustics computation methods. In this paper we present a two-pass algorithm for caustics computation that is extremely fast and produces high quality results. Our algorithm is targeted for commonly used height field representations of water and a planar caustic-receiving surface. The underlying theory of our approach is presented along with implementation details and pseudo codes.

Keywords Caustics · Real-time water rendering · GPU algorithms · height fields

1 Introduction

In water rendering, caustics are not only interesting visual elements that enhance the quality and realism of generated images, but also extremely important in providing necessary cues to perceive the shape of the rendered water surface. Despite their importance, the caustics component of light behavior is often neglected in real-time applications due to performance limitations. Most existing real-time caustics computation techniques are efficient enough for tech demos with relatively simple scenes on recent graphics hardware. However, in the presence of other visual elements that build up the complicated virtual environments of today's real-time applications, the full power of the graphics hardware cannot be devoted to a single visual element. Therefore,

caustics are often totally neglected or replaced by fixed animated textures that do not reflect the shape of the water surface.

We present a new real-time caustics computation technique that works well on both current and earlier graphics hardware. In this method, rather than providing a solution for all possible cases of caustics computation, we target the common case of the water surface being a height field and a flat ground surface underneath receiving caustics. This setting allows significant simplifications in the caustics computation, which result in it becoming one of the cheaper visual elements.

In Figure 1 two images of the same water surface are seen from above with and without caustics. Comparing the two images, one can easily see that caustics are essential in visually defining the shape of the water surface.

Unlike previous techniques that aim to capture photon trajectories by rendering point primitives, we compute caustics starting from the receiving surface and examining the illumination contributions of the neighboring points on the water surface. We perform this operation efficiently with a new two-pass algorithm that uses multiple render targets and a technique that resembles separable convolution filtering, which minimizes texture lookups per fragment. The most significant contribution of this paper is this two-pass algorithm that is introduced in Section 4.

In the next section we overview earlier caustics computation techniques. Section 3 explains the theory of our method and Section 4 provides the details of our two-pass algorithm. We present our implementation details and results in Section 5. Finally, we conclude in Section 6 with a brief discussion about the limitations and advantages of our approach.

C. Yuksel
E-mail: cem@cemyuksel.com

J. Keyser
E-mail: keyser@cs.tamu.edu



Fig. 1 The image on the left shows a water surface defined by a height field rendered with reflections and refractions. The image on the right shows the same water surface with caustics computed using our method.

2 Previous Work

Rendering of caustics has been recognized as an important aspect of creating realistic images for decades [1–3]. A wide variety of approaches for rendering caustics have been explored over the years, including (like ours) methods adapted to particular environments or particular hardware considerations.

Some of the earliest work on generating caustic effects used Monte Carlo path tracing [1] or analytic approaches [3]. However, the most common approaches to project light from a source into the scene, then represent in some fashion the places where the light “hits” the surface. An example of this is wavefront propagation, where the entire light wavefront is propagated through a scene [4].

Many of the more practical methods are derived from backward raytracing, first proposed by Arvo [2]. In this approach, rays are traced from the light source to create an illumination map, including caustic effects. Photon mapping extended this idea, tracing photons from a light source and storing them in a map based on where they hit [5]. A major push of recent techniques has been the extension of this idea to create “caustic maps” [6–9]. Caustic maps are created by tracing photons into the environment, but instead of storing the final photons in a photon map, creating from them a caustic map that can be projected (like a shadow map) into the final rendered scene.

Other methods have focused particularly on rendering caustics in water. Stam presented a method for generating inaccurate but visually compelling “random” caustic images, such as those on the base of a pool [10]. Most of the water rendering methods use beam tracing [11] in order to better capture volumetric scattering effects under the water. In contrast, our approach does

not attempt to capture such volumetric effects. One of the earliest such methods was Watt’s approach, which used beam tracing to trace the path of light through each triangle on the water’s surface and determine how that light is projected onto the underwater surface [12]. Later work continued the beam tracing approach [13] including extending it to use graphics hardware [14] and achieve interactive results [15].

The method proposed by Guardado and Sanchez-Crespo [16] computes caustics starting from the receiving surface as in our approach. However, unlike ours, this method computes caustics on each receiving point from a single point on the water surface. Therefore, this method is highly inaccurate and can only roughly approximate caustics for sun light coming from directly above the water surface. On the other hand, in our method we use a two-pass algorithm to capture caustics from an area on the water surface efficiently and accurately for arbitrary lighting directions.

3 Caustics Computation

In our caustics computations we follow light paths starting from the caustic-receiving surface instead of the light source. To simplify the computation, we assume that the caustic-receiving surface is a flat finite plane. The final result of our caustics computation is a caustics map that is mapped onto this plane. Figure 2 shows the caustics map of the frame in Figure 1 computed using our method. The grayscale value of each caustics map pixel represents the incoming light intensity of the corresponding pixel area on the caustic-receiving plane.

To produce this caustics map, we consider the refracted radiance from the height field water surface towards the caustic-receiving plane. For each pixel of the

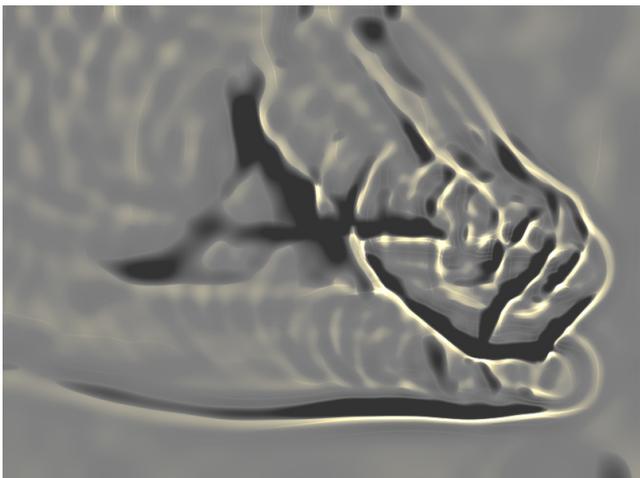


Fig. 2 The final result of our caustics computation is this caustics map that includes bright and dark areas corresponding to caustics. This is the caustics map of the frame in Figure 1.

caustics map, we sum the refracted radiances towards the pixel from all points within a rectangular region R on the water surface. We refer to the center of this rectangular region as the *illumination center*.

Let $z = 0$ be the ground plane underneath the water surface and \mathbf{P}_G be a point on this plane (Figure 3). The *rest state*¹ of the height field is represented by the plane $z = h$, where h is the *rest depth* that corresponds to the distance between the ground plane and the water surface. The illumination center \mathbf{P}_C that corresponds to the ground point \mathbf{P}_G can be found by

$$\mathbf{P}_C = \mathbf{P}_G + h\mathbf{L}', \quad (1)$$

where \mathbf{L}' is the refracted light direction \mathbf{L} from the rest surface with normal \hat{z} in positive z -direction.

The size of the rectangular region R limits the part of the height field surface region from which we can capture the illumination contribution to \mathbf{P}_G . In other words, our computation is accurate as long as the incoming radiance towards \mathbf{P}_G through the water surface is confined within this rectangular region. The required size of R to capture 100% of the illumination is a complicated function of \mathbf{L} , h , and maximum surface normal deviation. When the height field water surface has high frequency deformations with large magnitudes, this size can be arbitrarily large and even cover the whole height field surface. However, real-time water simulations using height fields often yield smooth surfaces with low frequency deformations. Therefore, even a very small rectangular region can cover a significant portion of the incoming light, regardless of the magnitudes of the deformations.

¹ In the *rest state*, the water surface is flat and all the values of the height field are equal to a constant value.

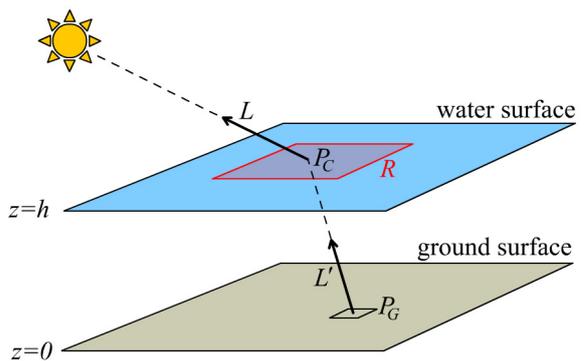


Fig. 3 The illumination on point \mathbf{P}_G comes from the refractions through the rectangular area on the water surface.

Let A_G denote the area of a caustics map pixel on the caustic-receiving plane. Assuming that the rectangular region R is sufficiently large, the average light intensity I_G over the area A_G can be written as

$$I_G = I_r(A_R, A_G) \frac{A_R}{A_G}, \quad (2)$$

where $I_r(A_R, A_G)$ is the reflected light intensity through the rectangular region R towards A_G , and A_R is the area of R . This equation can be written as an area integral over R ,

$$I_G = \int_R \frac{I_r(A_w, A_G) dA_w}{A_G}, \quad (3)$$

where A_w is an infinitesimal area on the water surface within R . To compute this integral we discretize this equation as

$$I_G = \sum_i I_r(A_i, A_G) \frac{A_i}{A_G}, \quad (4)$$

where A_i is the i^{th} sample area within R .

We approximate the reflected intensity I_r for the i^{th} sample by assuming that the surface normal is constant within the area A_i . Thus, $I_r(A_i, A_G) \approx \alpha I_r(A_i)$, where $I_r(A_i)$ is the average refracted light intensity through A_i and α is the fraction of the refracted area of A_i that intersects with A_G .

In the next section we introduce a two-pass algorithm to perform these operations efficiently.

4 The Two-Pass Algorithm

To generate the caustics map efficiently we use a two-pass approach that minimizes the number of texture lookups (Figure 4). In the first pass we read the height field texture at multiple points along one direction (x -axis) storing the illumination contributions of these points in multiple textures. In the second pass we read the textures generated in the first pass along the perpendicular direction (y -axis) yielding the final caustics map.



Fig. 4 The two-pass algorithm: the first pass reads the height field texture and generates multiple outputs; the second pass reads the result of the first pass and produces the final caustics map.

In the first pass, for each pixel $\mathbf{P}_G^{i,j}$ of the caustics map, we find the corresponding illumination center on the height field and read N samples along the x -axis on either side of the illumination center. We place these samples on the height field such that the distance between two consecutive samples is equal to the width of a caustics map pixel, such that each sample represents an area on the height field surface that is equal to the area of a caustics map pixel. Note that the resolution or the orientation of the height field does not have to match the caustics map, since we base our sampling density and orientation only on the caustics map.

The aim of this first pass is not only to find the illumination contributions of these N samples on the pixel $\mathbf{P}_G^{i,j}$, but also on the neighboring M pixels of the caustics map along the y -axis, from $\mathbf{P}_G^{i,j-M/2}$ through $\mathbf{P}_G^{i,j+M/2}$. Therefore, the output of the first pass needs $M + 1$ color channels, each of which correspond to a different pixel on the caustics map. On modern graphics hardware we can output up to 64 channels using multiple render targets (8 render targets with RGBA channels). However, in practice we found that as few as 8 channels can be sufficient since most of the illumination contribution comes from points close to the illumination center.

To compute the values of these $M + 1$ channels, we calculate the refracted ray directions of each one of the N samples on the height field and find where these rays intersect the ground plane. Assuming the surface normal is constant within each sample area, a pixel sized square centered on each intersection point indicates the area illuminated by the refracted light through that sample. For each one of these square areas, we find the nearest two pixels between $\mathbf{P}_G^{i,j-M/2}$ and $\mathbf{P}_G^{i,j+M/2}$, then we compute the fraction of the square that overlaps with each one of these two pixels. The sum of all these ratios yields the total ratio of the refracted refracted intensity through these N samples on these $M + 1$ pixels.

In the second pass, for each pixel $\mathbf{P}_G^{i,j}$, we simply sum the values from the previous pass that correspond to this pixel. These values are stored in different output channels of the first pass at the pixels $\mathbf{P}_G^{i,j-M/2}$ through $\mathbf{P}_G^{i,j+M/2}$. The resulting total values yield the fraction of incoming light at each pixel of the caustics map.

```

#define N      7
#define N_HALF 3

struct Pass1Out {
    float4 color0 : COLOR0;
    float4 color1 : COLOR1;
}

void Pass1( out Pass1Out Out
           , in float2 P_G : TEXCOORD0
           , in float2 P_C : TEXCOORD1
           , uniform sampler2D heightField )
{
    // initialize output intensities
    float intensity[N];
    for ( int i=0; i<N; i++ ) intensity[i] = 0;
    // initialize caustic-receiving pixel positions
    float P_Gy[N];
    for ( int i=-N_HALF; i<=N_HALF; i++ ) P_Gy[i] = P_G.y + i;
    // for each sample on the height field
    for ( int i=0; i<N; i++ ) {
        // find the intersection with the ground plane
        float3 pN = P_C + ( i - N_HALF ) * xDirection;
        float2 intersection = GetIntersection( heightField, pN );
        // ax is the overlapping distance along x-direction
        float ax = max(0, 1 - abs(P_G.x - intersection.x));
        // for each caustic-receiving pixel position
        for ( int j=0; j<N; j++ ) {
            // ay is the overlapping distance along y-direction
            float ay = max(0, 1 - abs(P_Gy[j] - intersection.y));
            // increase the intensity by the overlapping area
            intensity[j] += ax*ay;
        }
    }
    // copy the output intensities to the color channels
    Out.color0 = float4( intensity[0], intensity[1],
                       intensity[2], intensity[3] );
    Out.color1 = float4( intensity[4], intensity[5],
                       intensity[6] );
}

```

Fig. 5 The pixel shader pseudo code for the first pass.

5 Implementation and Results

The two-pass algorithm explained in the previous section enables efficient computation of caustics. Most of the computation is carried out in the first pass and the second pass merely combines the outputs of the first pass to produce the final caustics map. The computations of both of these passes take place in the pixel shader on the graphics hardware.

```

void Pass2( out float4 color : COLOR
           , in float2 P_G : TEXCOORD0
           , uniform sampler2D inColor0
           , uniform sampler2D inColor1 )
{
    float val = 0;
    val += tex2D( inColor0, P_G + float2( 0, -3 ) ).r;
    val += tex2D( inColor0, P_G + float2( 0, -2 ) ).g;
    val += tex2D( inColor0, P_G + float2( 0, -1 ) ).b;
    val += tex2D( inColor0, P_G ).a;
    val += tex2D( inColor1, P_G + float2( 0, 1 ) ).r;
    val += tex2D( inColor1, P_G + float2( 0, 2 ) ).g;
    val += tex2D( inColor1, P_G + float2( 0, 3 ) ).b;
    color = val;
}

```

Fig. 6 The pixel shader pseudo code for the second pass.

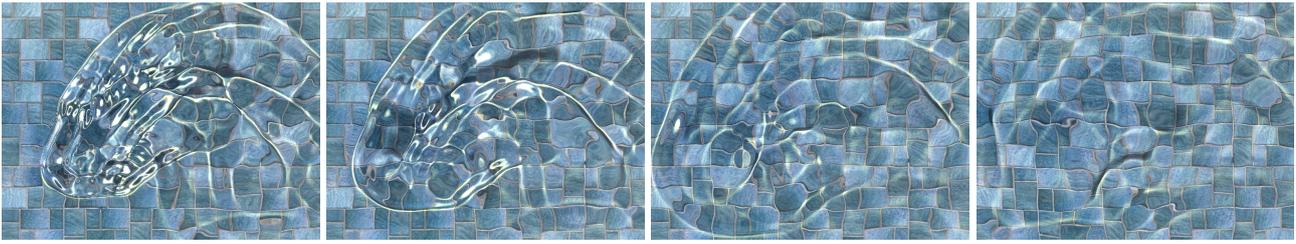


Fig. 7 Frames from an animated sequence captured from our real-time water simulation and rendering system.

With a little further analysis one can easily see that a large portion of the computation in the first pass is repeated by multiple neighboring pixels. The computation of the refracted ray directions and their intersections with the ground plane are repeated multiple times. In our implementation we introduce an additional pass before the first pass to compute the intersection positions of the refracted light rays with the ground plane. The first pass reads the output of this additional pass, rather than the height field itself to reduce its computation load.

We provide the pseudo codes of the pixel shaders of the first and second passes in figures 5 and 6 respectively for the case of $N = 7$.

Figure 7 shows sample frames captured from our real-time water simulation and rendering system. The caustics map size in these frames is 400×300 , $N = 7$, and $M = 6$. On a GeForce 9600 GT graphics card, the caustics computation per frame takes about 1.04 milliseconds, which corresponds to 960 fps. When the caustics resolution is 800×600 as in Figure 1, the computation time becomes 3.3 milliseconds, which is 303 fps.

6 Discussion and Conclusion

We present a two-pass algorithm for computing caustics from a height field water surface onto a flat plane. The efficiency of the algorithm comes from the fact that it does not require a high resolution water surface or a large number of point primitives to be rendered. The whole computation takes place in the pixel shader. It has a sequential texture access pattern, which highly utilizes the texture cache on the graphics hardware.

Our backwards caustics computation gives physically-based results as long as the caustics receiving surface is a flat finite plane. Therefore, using our method caustics on non-flat surfaces can only be approximated. Yet, this is a much better approximation than neglecting caustics or using constant animating textures, which are the two most common methods used in the current real-time applications.

The obvious limitation of this approach is that it can only handle height field surfaces and planar caustic-receivers. A useful future direction would be to extend this approach to accurately handle non-planar caustic-receivers.

References

1. Kajiya, J.T.: The rendering equation. *SIGGRAPH Comput. Graph.* **20**(4), 143–150 (1986)
2. Arvo, J.: Backward ray tracing. In: *ACM SIGGRAPH 86 Course Notes - Developments in Ray Tracing* (1986)
3. Inakage, M.: Reflection and refraction model for ray tracing. In: *ACM SIGGRAPH 86 Course Notes - Developments in Ray Tracing* (1986)
4. Mitchell, D., Hanrahan, P.: Illumination from curved reflectors. In: *Proceedings of SIGGRAPH '92*, pp. 283–291 (1992)
5. Jensen, H.W.: *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA (2001)
6. Szirmay-Kalos, L., Aszdi, B., Laznyí, I., Premecz, M.: Approximate ray-tracing on the gpu with distance impostors. *Computer Graphics Forum* **24**(3), 695–704 (2005)
7. Wyman, C., Davis, S.: Interactive image-space techniques for approximating caustics. In: *Proceedings of I3D '06*, pp. 153–160 (2006)
8. Shah, M.A., Konttinen, J., Pattanaik, S.: Caustics mapping: An image-space technique for real-time caustics. *IEEE Trans. on Vis. and Computer Graphics* **13**(2), 272–280 (2007)
9. Wyman, C.: Hierarchical caustic maps. In: *Proceedings of SI3D '08*, pp. 163–171 (2008)
10. Stam, J.: Random caustics: natural textures and wave theory revisited. In: *Proceedings of SIGGRAPH '96*, p. 150 (1996)
11. Heckbert, P.S., Hanrahan, P.: Beam tracing polygonal objects. In: *Proceedings of SIGGRAPH '84*, pp. 119–127 (1984)
12. Watt, M.: Light-water interaction using backward beam tracing. In: *Proceedings of SIGGRAPH '90*, pp. 377–385 (1990)
13. Nishita, T., Nakamae, E.: Method of displaying optical effects within water using accumulation buffer. In: *Proceedings of SIGGRAPH '94*, pp. 373–379 (1994)
14. Iwasaki, K., Nishita, T., Dobashi, Y.: Efficient rendering of optical effects within water using graphics hardware. In: *Proceedings of Pacific Graphics '01*, p. 374 (2001)
15. Ernst, M., Akenine-Möller, T., Jensen, H.W.: Interactive rendering of caustics using interpolated warped volumes. In: *GI '05: Proceedings of Graphics Interface 2005*, pp. 87–96 (2005)
16. Guardado, J., Sanchez-Crespo, D.: Rendering water caustics. In: R. Fernando (ed.) *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education (2004). Chapter 2