Vol.2017-CG-166 No.6
2017/3/13

Real-time caustics rendering in large marine environments

Killian Guerin[1,a)] Masao Izumi[1,b)]

Abstract: Simulating caustics is a challenging and computationally expensive task. This paper introduces a real-time method to simulate caustics in large marine environments, depending on the surface producing these light effects. Inspired by existing techniques, it tries to improve the quality without adding too much constraint on the GPU.
Caustics are first approximated using a lookup at the water surface's informations. For a given underwater point, we deduce a light strength coefficient by comparing a refraction ray with the global light's direction.
Past techniques does the refraction ray computing by assuming the underwater surface is a plane. By modifying the way we seek surface informations in our textures, we can generalize this technique to more complex surfaces. Then, by refining formulas in our computing, we tend to improve the results.
Another aspect we added to this algorithm is the god rays generation. By using the caustic rendering method on sample points projected unto a grid defined by planes, it is possible to have a stable volumetric alike rendering underwater.

1. Introduction

Caustics are fascinating light effects. Present in our everyday life because of very reflective or refractive materials, it has a great effect on how a scene will appear to us. Although this effect can sometimes be very subtle, one can notice when it is absent from an image, even if they cannot say what is missing. Caustics are simply areas where rays of light are focused.

Simulating this effect in computer graphics is a necessity to reach realistic images. The most comfortable way of simulating this effect in computer graphics is to recompute the light path to know where rays end after reflection or refraction. However, even though the origin of this effect can be quite simple to understand, it is really hard and expensive to do so and images can take a lot of time to be computed.

Real-time techniques need heavy approximations to be able to run at sufficient speed. By ignoring multi-bounces or trying to minimize the number of samples, these algorithms are really simple mimics of what light should do. They sometimes even ignore spatial data as we will see later. The aim here is to simulate caustics in real-time for large environments. Even though we proceed by doing severe approximations for efficiency, we want to include as much data as possible to respect the actual scene being rendered.

In this paper we will first describe the phenomenon behind caustics. It will allow to understand the idea behind the algorithms used nowadays and we will present ours afterward. We will then analyze the results obtained and discuss further possibilities in the implementation.

[1] Osaka Prefecture University
[a)] killian.guerin.92@gmail.com
[b)] izumi@las.osakafu-u.ac.jp

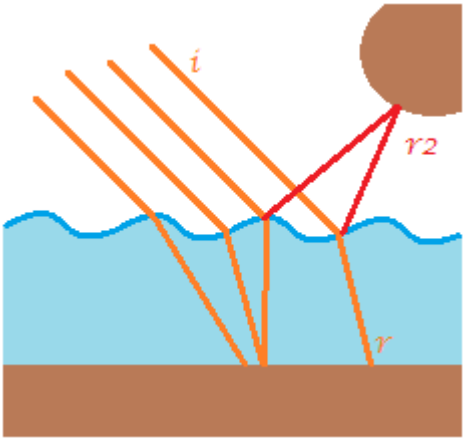

Fig. 1 Caustic's origin

2. Phenomenon origin

2.1 Caustics

To understand this effect, let's consider a ray of light coming from the sun and entering the ocean. Water is translucent and will allow light to pass. However, it has a different constitution from our atmosphere and light will progress through it in a different manner, with a different speed for instance. This change in speed makes the light beam change its trajectory at the boundary of the two materials. This is called refraction.

Caustics can be generated by such phenomenon. Water is not a perfectly flat surface. There are disturbances on it, thus modifying the refracted rays directions. Those rays can then concentrate on some areas, rising the light strength there. Moreover, not all light rays enter water. Some are also reflected, creating in the same manner areas where concentrated light meet. Those areas are what we call caustics. Figure 1 illustrates the effect. Red

Fig. 2 A real caustics formation example

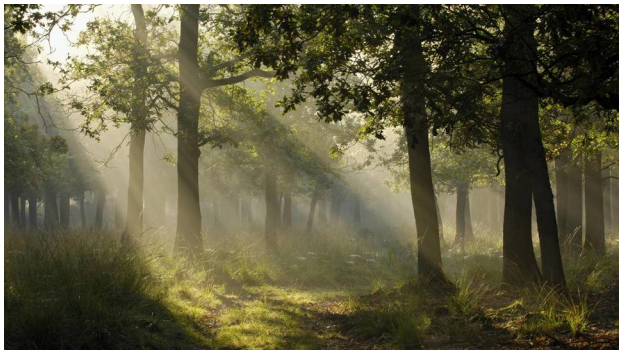

Fig. 3 Crepuscular rays example

rays are reflections whereas yellow ones are refractions. You can notice the way they gather depending on water surface's shape.

This effect can be noticed on image 2 these effects. It represents caustics created by the water on the sand beyond it. The water surface's complexity create the seemingly random shapes of the caustics.

2.2 God rays

When light goes into a non-empty environment, it will be scattered as it encounters the small particles inhabiting it. This will give the feeling we can see the light ray, like when throwing sand in front of a laser beam. However, there can be some objects casting shadows in this environment. If we imagine being in a forest, pollen and such particles can make the light appear to our eyes. But there will be trees casting shadows everywhere, giving the feeling of having rays of light going around in the air.

The image 3 illustrates this effect in the forest scenery. Notice how the trees are hiding the light, thus casting shadows not only on the ground, but also in the air.

The same thing is happening in the water, but with a different mechanism. Shadows are cast by the refractions. When focusing beams on a place, some other areas are put in the shadows, thus disturbing the uniformity of light distribution. This is how we can have god rays in the water. Of course, you can also have objects casting shadows in the water itself, adding to the effect.

3. Related works

Many researchers have been investigating caustics. Fascinating effect adding a lot to realism, it is however extremely hard and time consuming to compute in a realistic way. We will however

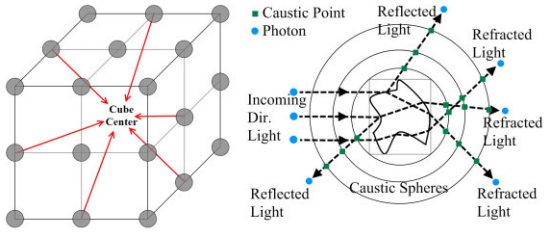

Fig. 4 [1]'s technique logic

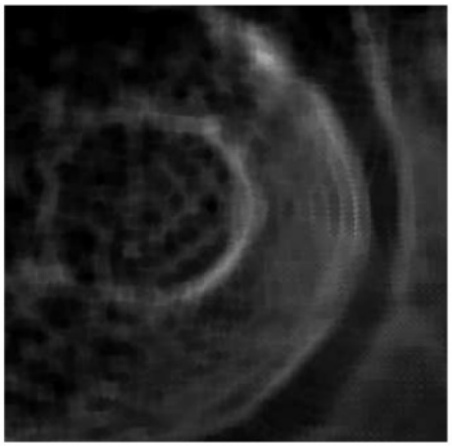

Fig. 5 A caustic map generated by [3]

focus on real-time techniques trying to compute environment dependents caustics, and exclude ray-tracing or simple texture projections techniques.

Caustic rendering is done according to multiple techniques compatible with different situations. Some of them focus on particular objects, other on global informations. But all use heavy approximations or pre-computing in order to ensure speed in their computations.

For instance, one technique try to compute beforehand the behavior light should have with an object. This information is stored in spheres including the object to be reused later when projecting the scene [1].

By computing beforehand how a directional light will behave on a specific object, they can then use these informations to project light once refracted and reflected on the scene being rendered. Moreover, by having a sphere distribution like in the image 4, they can use interpolation to produce volumetric caustics.

However, once precomputed, it can only be used with the settings used to produce the data. Although they can rotate the spheres to cope with directional lights changing their facing directions in some cases, the mesh cannot be distorted or animated.

Another technique produces a photon map by refracting the light direction using the refractive objects vertex informations. Then, as in shadow mapping, these informations are projected on the scene to know which areas on screen are brighter than the others [3].

This technique can be applied on animated mesh, as it can be updated before caustic map projection. It can also adapt to light

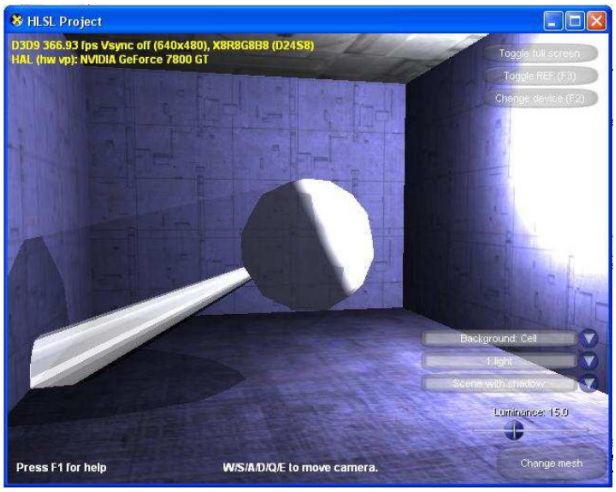

Fig. 6 Even if still work in progress, [4] reaches convincing results

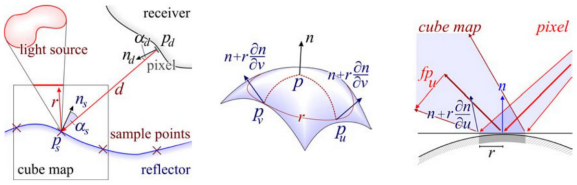

Fig. 7 [3]'s object subdivision

Fig. 8 [5]'s results

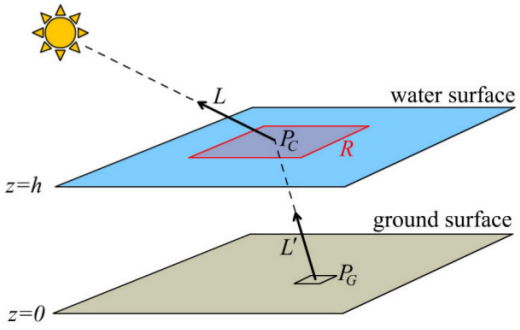

Fig. 9 How [6] pushes the logic

settings changes. Being an image space technique like shadow mapping, it suits very well to real-time rendering. However, it inherits all disadvantages shadow-mapping has : map textures size quality dependence, aliasing and projection resolution.

Pushing further the similarity with shadow computing techniques, [4] uses the shadow volume logic to add a volumetric side to the caustic mapping technique. By extruding the silhouette and making it thinner on the end, the volume generation allows to get not only projections on neighboring objects, but also the light effect in the air.

In another approach, [2] tries to emulate a disco ball effect by dividing the refractive or reflective casting surface.

Objects are subdivided so that each cell will be treated as a facet of the disco ball. Each facet will be reflecting the light source, and as such it will be possible to sum the contribution of every object by averaging the facets reflections. To ensure reflection continuity, the cells also have some parameters like curvature. This allow to compute sharper reflections without discontinuities.

The results are encouraging, but this method is heavy. At the time, it needed a lot of samples before getting convincing results. However, the more samples, the slower it gets as its computational cost increases. On simple scenes they hardly reach 20fps, and it could drop to 5fps. Although is was in 2003, those rates were obtained when computing 400x400 images on a good GPU.

When talking about water, a lot of assumptions can be made like in [5]. They consider the surface underwater as an horizontal plane. Then, they strip the caustics rendering as being only done by one ray, refracted by the surface above itself and compared to the original light's direction.

This technique makes some heavy assumptions making it applicable only in a small number of cases. Moreover, as it simplifies a lot the caustics logic, it is a more artistic approach rather than a physically correct rendering. However, it is very light to compute and reaches convincing results.

In order to be closer to physically correct techniques, [6] tries to push further [5]'s method. The logic is the same, appart from the fact that the aim is to sample the surface not only once, but also consider neighboring positions. Caustics are then an average of different contributions coming from different places.

By calculating the participation of one entire region, it tries to improve the caustics computing logic. However, to get better results one needs to increase the neighboring region's size, thus putting more load on the GPU. Although results are better, we need more computations and it is especially intensive on texture accesses. To avoid this, [6] computes the contribution over multiple passes to minimize the texture look-up number. This is a choice to do as it then needs memory to store the temporary informations computed by each pass.

4. Our implementation

Our method is inspired by current lightweight algorithms specialized for water rendering. The aim being to render large environments, we need it to be fast to compute.

We wanted to be independent from geometry complexity. This method acts in post-process by reconstructing pixels' positions and using informations stored by a standard deferred rendering pipeline. By doing so, we can process the right amount of data without having to manage level of detail stages. Moreover, as it is a post-process method, it adapts to everything on screen without

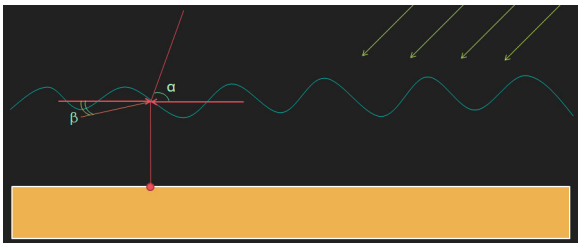

Fig. 10 We compute the 'α' coefficient and combine it with the 'β' one

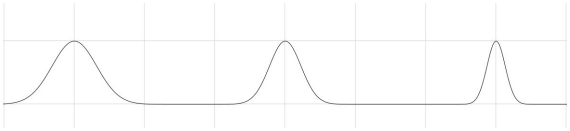

Fig. 11 τ's influence as it increases (from left to right, values are 5, 10, 30)

having to change anything in the standard scene rendering.

The method begins by considering one ray departing from the point we want to light as in [5]. This ray goes in a straightforward up direction, assuming the given point belongs to a plane. From the water surface informations up this point, we refract the direction vector and compute the dot product of it with a perpendicular x direction. This gives us a coefficient we will use to assign a light strength factor to this point.

Along with this, we also compute the reflection ray coefficient using the same surface informations and method. This newly generated ray will be incorporated in the formula. The idea behind this is to consider one bounce the light could do on the material. This simple component adds to the random side of the generated caustics.

Considering the example 10, we compute the cosine of the angle α to get an alignment ratio between the reference vector and the refracted ray. It is also done for the reflection ray. These two components are blended to get the final caustic coefficient.

However, we apply a filter to process our values. We define a translation formula, to "tighten" the caustics generated by the algorithm, described like this :

$$\omega = e^{-\tau \times a^2} \tag{1}$$

Where ω is the final assigned caustic strength, τ is the factor of tightening, and a our cosine computed beforehand, being the refracted or reflected ones. τ should be positive. The higher τ is, the more compressed will be our result.

Notice this exponential curve bell is centered on 0, meaning the light coefficient attached will be 1 if the dot product is 0, and 0 if the product is 1. This is why we compare our rays with a reference vector perpendicular to the incident ray, so that we can measure the degree of divergence from the original ray according to our formula.

This allows us to get more natural caustics, more compressed even on calm waters, as we can notice on example 2.

Getting the water surface information is done using UV tri-planar texture projection. This allows us to work on not-only planar underwater surfaces, as we project the informations in 3D. Complex shapes are possible with this technique.

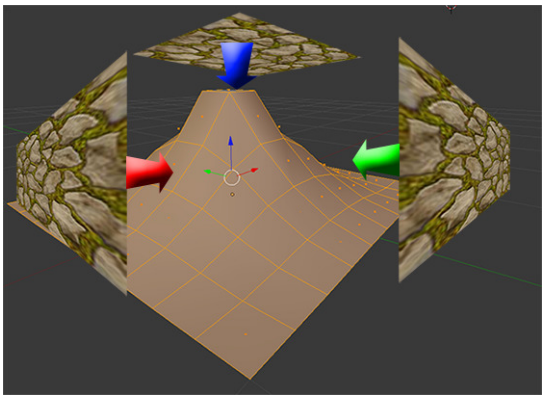

Fig. 12 The idea behind tri-planar projection

As we work in world space by reconstructing a given pixel's position, we use this 3 dimensional (x, y and z) information to sample the water surface textures. As these textures are defined in 2 dimensions (u and v), we have to choose wisely which component has to be ignored. Tri-planar projection uses the surface normal to define which components are the most important.

Informations will be computed 3 times. One time using as (u, v) the (x, y) of our position, and then two other times using (x, z) and (y, z).

On the example 12, we understand why informations are computed using these components alternatively. Each data will be representative of one plane, one defined by (x, y), the other one by (x, z), and the last one by (y, z). By using the normal assigned to the surface point we want to process, we can know in which direction it points, thus knowing which plane it faces the most. By blending between the 3 informations according to this, we reach a natural way of projecting informations on 3 dimensional surfaces.

By using this method, we lift a limitation of older similar algorithms that were only compatible with planar surfaces, allowing us to work with more natural and complex surfaces.

Our next step is to compute the crepuscular rays that can be present in the water. For simplicity, their computing was done using the presented caustic rendering method without tri-planar UV projection. The aim was to sample n different positions along the direction vector of one pixel, and sum up their light contributions obtained to average them later.

The challenging part here is to ensure continuity when the camera moves. If we simply split the line between the pixel and the seen surface, then when changing the position of the camera the sampling points will move too, making the result differ as different surface informations will be used. To avoid that we consider the rays shape.

Indeed, the rays will be considered straight. By doing so, we can imagine a grid of interleaved planes centered in the world. Each plane will be disposed at different values on each axis, allowing us to know our position in this imaginary grid.

The grid will consider only the x and z aligned planes to follow the rays shape. Being snapped to world coordinates, it doesn't move with the camera, allowing each sampling position to be constant on time.

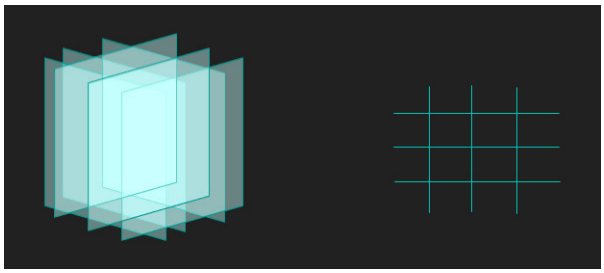

Fig. 13 The grid as imagined. Left is seen in 3D, right is from up

For a given pixel, the aim is to know on which plane the next sampling point is. If we consider the x and z dimensions, by doing simple computing, we can figure out which cell the camera is in. It becomes easy to compute the next plane which will be hit by our ray and deduce the intersection point coordinates.

We chose not to consider the grid as a set of cubes, but rather as a set of planes crossing each other. This allows to simply reconstruct the grid and step into it from image space.

It is then possible to compute the sum of every coefficient and divide it by the total distance done during the sampling process, allowing to smooth the result. The smoothing goes further as we also use the depth to fade the light strength. It simulates the out-scattering part in a simple way. We also define a fading factor computed according to the distance of one sample from the surface. The closer, the less visible it should be to avoid a tiling effect due to the grid's nature. Here is the thought expressed as a formula :

$$\gamma = \frac{\sum_{0}^{n} \rho_i \omega_i}{D_n} \tag{2}$$

With γ being our pixel's light factor, n the number of samples taken, i the sample point being processed, ρ the participation coefficient based on the distance it is from the surface, ω the caustic's participation computed using our method, and D_n the total distance crossed when advancing toward our sample points.

We also orient the ray along the light direction given. To do so, we simply rotate the grid so that it has its up vector following the light's direction. By using a translation matrix to go from the world's coordinates to the grid's rotated coordinates, we can easily compute our values and retrieve the real sample position to apply effects according to depth.

5. Results

Results were produced using DirectX11 on Windows. Measured times were taken on a MSI laptop under Windows 7, having an Intel i7 CPU, a 8Go RAM and a Nvidia Geforce GTX960M.

5.1 Images

You can notice on image 14 the blending of only the caustics compared to the original image. Then on figure 15, we blend only the god rays to notice them easily. Finally, we show the whole effect compared to the image we were modifying in 16.

We can see how the effect enlighten the image, like normal caustics would do in a real situation. Image 17 shows the total result seen from another point of view underwater, whereas 18 shows the caustics as seen above water.

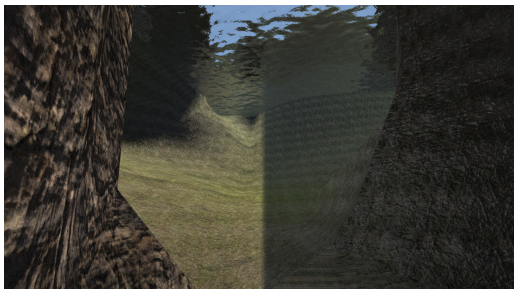

Fig. 14 A comparison when adding caustics

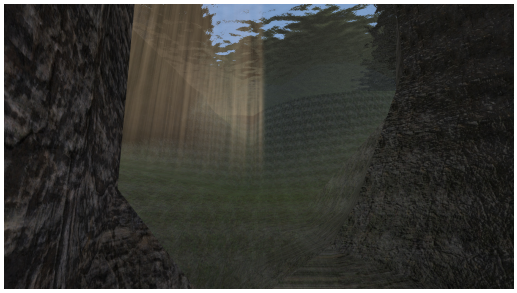

Fig. 15 A comparison when adding god rays

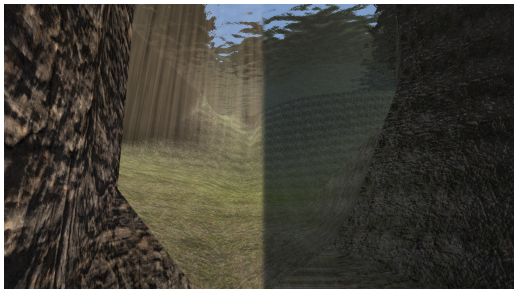

Fig. 16 The full comparison

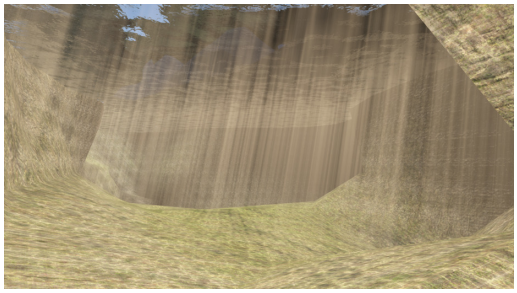

Fig. 17 Underwater total effect

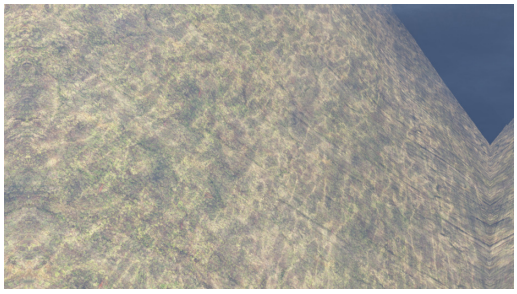

Fig. 18 The effect as seen from above the water

5.2 Time

Rendering times were taken using the DirectX11 GPU query system's capabilities. Time is measured in milliseconds.

We took these times when having the worst case scenario. As we have a post-process method, the number of pixels to proceed can have an influence on the rendering time. We took images that needed a whole frame processing, so that we had to process every pixel in it.

Caustics	Time (ms)
M_1	1.5

Caustics rendering time is under 2ms. If we aim for 60fps, it takes about 12% of the rendering time.

Sample Number	Time (ms)
4	0.2
8	1.0
16	1.9
32	6.6

What we can see is that from 16 to 32 samples, we lose a lot in performance. This may be due to the fact that we are hitting a bound when accessing the textures for surface informations.

6. Future work

On this algorithm, we worked a lot on the shape of the caustics rendered. We tried to improve their generation, but did not have time to insert them into a more complex lighting system.

Future works could use shadow mapping techniques to improve the rendered images' quality. This could help insert the caustics further into the scene they take place in.

Another point would be to improve the computing formula to be closer to how light scatter in the water environment. This could alleviate the need of having parameters describing the color and strength of the caustics to improve results.

Another important point not taken into account are the caustics projected on upper water surfaces due to the reflections. At this time, we do not have complete ideas on how to tackle this problem.

Performances could also be improved by changing the way we fetch data. We are doing it without especially considering performances by reconstructing for every point, every time and dynamically the informations. By being able to compute beforehand some informations or improve the reconstructing method, it could be possible to improve the rendering times.

7. Conclusion

Caustics are fascinating light effects. Due to the random path a light can take, it focuses on particular zones, enlightening them. By nature, it is really hard to compute correctly in a short time and we have to make severe approximations as we did in our algorithm.

We however tried to improve the number of cases in which this kind of solution is acceptable by changing the way we seek water surface informations. We also tried to add a new effect, using a volumetric-like approach, to improve the visual quality. All these computing are done keeping the scene shape in mind so that it scales well with what is being seen. Another important point is that we do this in a full post-process pass, allowing the algorithm to be easily inserted into a pipeline.

To summarize our method's functionalities :

- We increased the number of suitable situations for this kind of algorithms by using tri-planar UV mapping.
- The caustic's shapes randomness was improved by introducing a bounce in the light's computation.
- Crepuscular rays are simulated using a static grid setting when searching for sample points.
- All of this can be rendered for a large marine environment even on standard consumer computers.

There is however still a lot to do to keep increasing the correlation with the scene. Computing shadows, more complex reflections and scattering could help to closer the gap between the current approximation and the real behavior.

References

[1] Budianto Tandianus, Henry Johan and Hock Soon Seah: *Real-Time Caustics in Dynamic Scenes with Multiple Directional Lights*, Entertainment Computing - ICEC 2010, 9th International Conference, Seoul, South Korea, September 8-11, 2010. Proceedings, Seoul, South Korea, 2013.

[2] M. Wand and W. Straer: *Real-time Caustics*, EUROGRAPHICS, University of Tbingen, 2003.

[3] Musawir Shah and Sumanta Pattanaik: Caustics Mapping: An Image-space Technique for Real-time Caustics School of Engineering and Computer Science, 2007.

[4] Andrei Diakonov: Real-Time Caustics Rendering Institute of Mathematical Modelling, Technical University of Denmark, 2009.

[5] Juan Guardado and Daniel Snchez-Crespo: Rendering Water Caustics *GPU Gems*, Vol. 1, Nvidia, Universitat popeu Fabra and Novarama Technology, 2007.

[6] Cem Yuksel and John Keyser, Fast Real-time Caustics From Height Field, Texas A&M University, 2009.