Eurographics Workshop on Natural Phenomena (2006), pp. 1–8
E. Galin, N. Chiba (Editors)

Realistic Water Volumes in Real-Time

1022

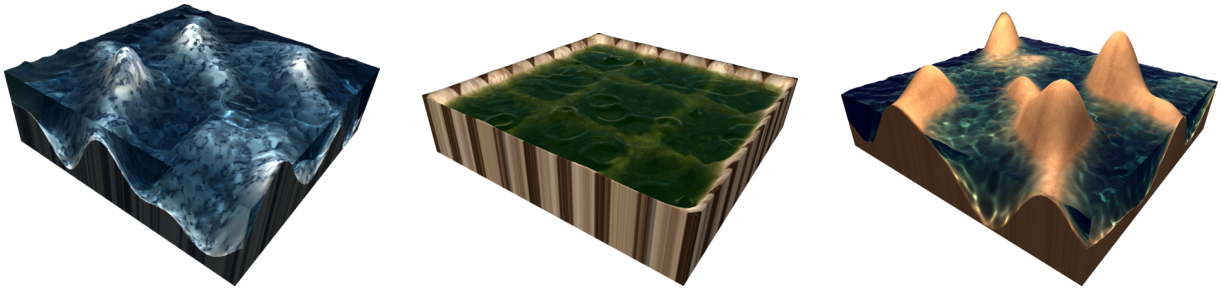


Figure 1: *Examples of rendering of water volumes. All images are* 800×600 *and are generated at about 30Hz.*

Abstract

We present a real-time technique to render realistic water volumes. Water volumes are represented as the space enclosed between a ground heightfield and an animable water surface heightfield. This representation allows the application of recent GPU-based heightfield rendering algorithms. Our method is a simplified raytracing approach which correctly handles reflections and refractions and allows us to render complex effects such as light absorption, refracted shadows and refracted caustics. It runs at high framerates by exploiting the power of the latest graphic cards, and could be used in real-time applications like video games, or interactive simulation.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Realistic rendering of water is one of those effects that can highly increase the perceived realism of virtual scenes. However, the appearance of water is caused by many physical phenomena that were typically not affordable in real-time until recently. Indeed, the increasing functionalities and power offered by graphics hardware allow the adaptation of well-known off-line techniques to real-time applications. In particular, ray-tracing is well adapted to the SIMD nature of fragment processors. However, porting generic ray-tracers to the GPU is quite involved [PBMH02]. Hierarchical acceleration data structures, or generic mesh-ray intersection routines, must be carrefuly rewritten to fit the GPU organisation. Fortunately, in specific situations, the computations can be greatly simplified and efficiently mapped to the GPU. In particular, this is the case for the ray-tracing of heightfields. The question is then to identify the class of objects that can be adequately represented with these simpler models.

The contribution of this paper is to show that water basins and similar structures can be quite realisically modelled by two heightfields, one for the ground, and one for the water surface. Using adaptations of recent techniques for heightfield rendering, complex effects like reflections, refractions, light absorption, refracted shadows and caustics can be efficiently simulated in real-time. Except for a part of the caustics computations, it runs entirely on the GPU. The amount of displayed geometry is negligible and the rendering time is proportional to the number of pixels occupied by the water on screen. The different features are mostly independent and can be enabled separately, allowing a tradeoff between realism and performance.

2. Previous work

Displaying water surfaces involves two orthogonal problems : simulating the movement and rendering it accurately. Our work focuses only on this latter part, and any animation method can be used. We briefly review existing techniques.

Water Simulation Fluids are governed by the Navier-Stokes equations which have been used at different levels of simplification. Applying these equations to a volumetric representation of water leads to physically accurate but computationally intensive simulations [FF01, EMF02]. They are required for complex movements like breaking waves or splashing water. The study of Navier-Stokes equations in the context of ocean surfaces modeling has led to the Gerstner and Biesel swell model which decomposes ocean waves into a sum of trochoidal wave trains. Realistic results [FR86, HNC02] have been obtained by combining it with models expressing waves amplitude, frequency and direction spectra derived from measurements [PM64, HDE80], and waves propagation properties like refraction near coasts [Pea86, TB87, Tes99, GM02]. These approaches generally allow one to compute a heightfield representation of the water surface in real-time. In the case of bounded water surfaces, the surface heightfield can be expressed as the solution of simple partial differential equations [KM90] which can be solved numerically by efficient local diffusion schemes [Gom00, Sta03]. The spectral approaches [MWM87, Tes99, PA00] first compute a Fourier spectrum of the water surface heightfield, using physically measured distributions, before transforming it into a heightfield with a Fast Fourier Transform. The resulting heightfield is bounded but periodic.

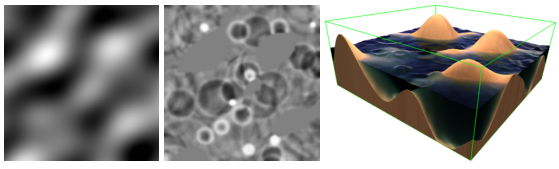
Water Rendering Rendering of water surfaces is difficult due to complex interactions with light. As explained by [PA00], the physical study of light-water interactions is a full-fledged research field with a vast litterature. Several works have focused on the transcription of these phenomena for computer generated images [Wat90, PA00, IDN03]. Although realistic, these approaches are computationally intensive. Realtime solutions exist, but they generally make important simplifying assumptions [NN94, HVT*04]. Reflection and refraction are important effects for the realism of virtual water. Until recently, reflection approaches would only address special configurations such as planar reflectors [FvDFH90] or infinitely far reflected objects, or would be only interactive [HNC02] or based on precomputations [HLCS99, YYM05]. Recently, real-time approaches using GPU-based ray-tracing and image based approximations of the reflected objects have been proposed [SKALP05, PMDS06]. Refraction algorithms usually consider one transparent object surrounded by an infinitely far environment represented by an environment map (see [GLD06] for a recent review of these methods). This assumption can not be used in the case of the refraction of a ground surface through a water volume. Computing the refraction of nearby objects can be done with memory costly light-field precomputations [HLCS99]. Caustics are caused by the convergence of light due to reflections and refractions. They are computationally expensive to render and are traditionally handled by global illumination algorithms like photon mapping [Jen01]. Interactive solutions exist based on different photon emitting and gathering strategies [EAMJ05, WS03, WD06, DS06] but they need to reduce the number of emitted photons to run in real-time, thereby producing noisy or blurred caustics. We use a method conceptually very similar to backward beam tracing with illumination maps [Arv86].

Heightfield rendering Our technique is based on efficient heightfield rendering. Efficient GPU-based implementations [POC05, Tat06, BD06] have made it an advantageous way to render geometric details with small computation and memory costs. These algorithms sample the heightfield texture along each viewing ray using different sampling strategies. [POC05] intersect the viewing ray with fixed horizontal planes before computing a precise intersection with few iterations of a binary search. This amounts to an even sampling in the vertical direction, which can cause stair-stepping artifacts at grazing angles. This dependence of the sampling on the view angle is reduced by [Tat06] which chooses the number of samples between fixed bounds according to the viewing ray direction. The horizontal sampling method described by [BD06] uses precomputed information to sample the ray optimally. This allows the accurate display of heightfields potentially containing fine features.

3. Our approacch

3.1. Modelling hypothesis

Our observation is that, except for big waves, water surfaces are mostly 2.5D. Similarly, because of gravity, most terrain can be represented as a displacement over a flat surface. Therefore, we choose to represent a water volume with two heightfields, the ground surface and the water surface, both encoded as 2D textures (Fig. 2). The texture for the water surface is the output of a simulation procedure which can be procedural (e.g. a sum of basis functions) or numerical (e.g. a physical simulation). To render the water volume, we ren-



(a) (b) (c)

Figure 2: *The ground (a) and water (b) surfaces heightmaps and the corresponding water volume (c).*

der its bounding box and use a fragment shader to perform

ray-tracing. During this, we only consider one level of recursivity : for each viewing ray, we find the intersection with the water surface, then the intersection of the refracted ray with the ground. Avoiding recursion is important because current fragment shaders do not support recursive function calls. This single level recursion assumption is correct if the

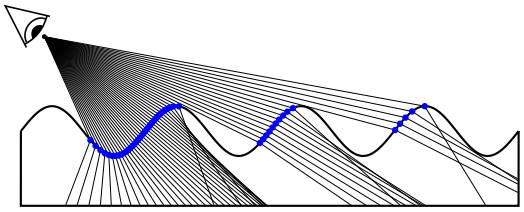


Figure 3: *The single level recursivity assumption generally holds, even for large waves.*

ground is non-reflective, and if no light-ray crosses the water surface more than once. Due to the leaning effect induced by refraction, even in presence of big waves, this hypothesis generally holds (Fig.3). When it is not the case, the visual difference induced by this simplification is hard to notice for the human eye. The rays we consider are shown on Figure 4. The algorithm outline is the following :

```
v ← viewpoint
for each screen pixel
    d ← direction of the corresponding viewing ray
    p_g ← intersection(ground, ray(v, d))
    p_w ← intersection(water, ray(v, d))
    if (p_g undefined and p_w undefined)
        do nothing (discard fragment)
    else if (p_g before p_w)
        pixel ← lightedGroundColor(p_g, d)
    else
        d_t ← refractedDirection(d, n(p_w), η)
        d_r ← reflectedDirection(d, n(p_w))
        p_g ← intersection(ground, ray(p_w, d_t))
        p_e ← intersection(ground, ray(p_w, d_r))
        C_t ← lightedGroundColor(p_g, d_t)
        C_r ← if (p_e undefined)
                  envMap(d_r)
              else
                  lightedGroundColor(p_e, d_r)
        F ← fresnelReflectivity(d, n(p_w), η)
        pixel ← (1−F) × C_t + F × C_r
```

The fresnelReflectivity procedure computes the ratio F between reflected and incident radiance with the Fresnel formula. For a fixed refraction index η, it is only a function of the incident angle, which can be efficiently computed with the approximation given by [Sch93] or simply sampled in a precomputed 1D texture. The lightedGroundColor procedure computes the color of a point on the ground from its coordinates, the viewing direction and the light position. True computations should integrate all rays coming from the light and

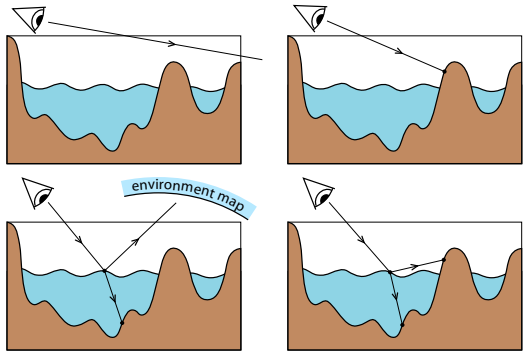


Figure 4: *Four possible situations are handled by the algorithm.*

arriving at p_w after refractions. Instead, we make a simplification. We use a simple Phong model involving the ray from p_w to the light, the normal, and the material at that point (the latter two being specified by texture maps). Only the amount of light is computed, taking into account the many possible light rays (see Section 4.3 on caustics).

Note that we implicitly supposed that each ray entering the water hits the ground further on. This condition is satisfied if the water is correctly enclosed by the ground surface, i.e. if on the borders the ground height values are higher than the water ones. This holds for a basin or a pond, but not for an aquarium (Fig.5). In that latter case, a dynamic environment map can be used to approximate what the ray "sees" when exiting the water.

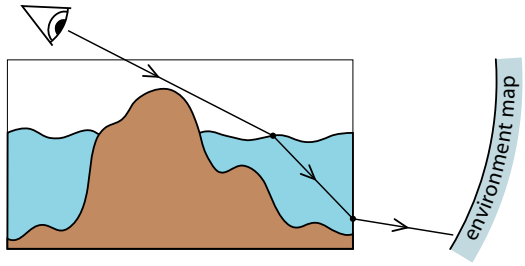


Figure 5: *Outgoing light rays from an unclosed water volume. We look up an environment map to find out what they "see".*

3.2. Optimized rendering

Our rendering algorithm is a tailored version of [BD06]. We refer the viewer to that paper for detailed description. In brief, it walks along the ray until a binary search can be safely run, i.e. it is guaranteed that the interval of that search contains at most one intersection with the heightfield. For that, it uses a pre-computed *safety radius texture* indicating for each texel the maximum neighborood in which a ray

above the texel can intersect the heightfield no more than once. The paper also describes a slower – but still exact – method that does not use the safety radius texture.

In the rendering of water surface, we perform intersections with two heightfields, the ground and the water surface. As the ground is static, precomputing the safety texture can be done off-line. This is not the case for the water texture which is dynamic. Fortunately, compared to the ground, it generally has a small amplitude. We thus estimate the average amplitude and size of the features, and compute a constant safety radius accordingly. In the worst case, it gives a radius of 1, which boils down to using the exact intersection described in [BD06]. To further optimize the number of steps required to find the intersection, we intersect the viewing ray with the bounding box of the water surface instead of that of the ground and water altogether.

Conceptually, we run the ray intersection algorithm several times (see pseudo-code in Section 3). However, in practice, we can perform the search along the ray for the two heightfields at the same time, thereby saving the computations of the successive samples taken on the light ray. It allows to stop as soon as the first of the two surfaces is hit. Moreover, if the two heightfields are packed in a single texture, this divides by two the number of texture lookups performed. Packing the ground heights with the water ones can be done during the precomputing of the water texture at no additional cost.

We want to emphasize that various strategies can be adopted here. Although we use the artifact-free approach of [BD06], any other method such as [POC05, Tat06] can be used. One can also choose to use different methods (binary search, horizontal or vertical fixed steps, Amanatides traversal, constant or texture-based safety radius, etc.) depending on the desired tradeoff between speed and quality, and depending on possible constraints on particular viewing conditions (distant or nearby water, grazing angles, large or small amplitudes of the heightfields, etc.). The only limitation is that the joint intersection described in previous paragraph is applicable only if the same method is used for intersecting with the ground and the water. Our goal in this paper was to show that the availability of these techniques and the heightfield representation for water basin and alike makes the rendering of complex effects achievable in real-time. We provide images and timings of what the programmers can expect, but the tuning for a particular application remains their responsibility.

Finally, we would like to say a few words about the precision used for heightfield values. For most situations, using one 8 bit channel per heightfield is sufficient. But if the ground or water surfaces are to be viewed from very close, or if they have large amplitudes, it will not be sufficient. Even if the ground is smooth and is correctly sampled, the bilinear interpolations performed on 8 bits numbers will produce stairs artifacts. This can be avoided by using 16 bits textures (the two height maps can be packed in a 32 bits texture with two 16 bits channels). Note also that as water is likely to have a small amplitude, it is more appropriate to store it in a normalized form by passing two additional parameters to the shaders : a height offset (the mean water level) and an amplitude. Doing so allows us to fully use the precision offered by the texture.

3.3. Integration with other objects

For our method to be integrated within a classically modelled scene, we must handle correctly the interactions with other objects. The first thing to notice is that for each fragment, the exact 3D intersection point of the viewing ray with the first encountered surface is known. Transforming it into the camera frame, we can trivially compute an appropriate z-value (instead of that corresponding to the bounding box we are actually drawing). Furthermore, for rays that do not intersect any of the two surfaces, the corresponding fragment is discarded, so it does not interfere with any past or future value in the framebuffer. Finally, the current depth in the z-buffer can easily be taken into account during the intersection procedure. We just stop the walk along the ray when this value is exceeded (if it corresponds to a point v_z between the segment $[v_0, v_1]$, the only modification to do is to restrict this segment to $[v_0, v_z]$). For all these reasons, our water volumes seamlessly integrate with the z-buffer, yielding correct occlusions with other rendered primitives.

This statement is however true only for objects that do not enter totally (like fishes) or partially (like the legs of a character) the water volume. Treating these cases is difficult because after undergoing the refraction, the viewing rays can be scrambled in a way which is difficult to predict (Fig.6). Having objects occluding viewing rays between the water surface and the ground breaks the organization into heightfields of this specific volume which made the raytracing feasible. Note that this effect is typically faked with ad-hoc hacks in games.

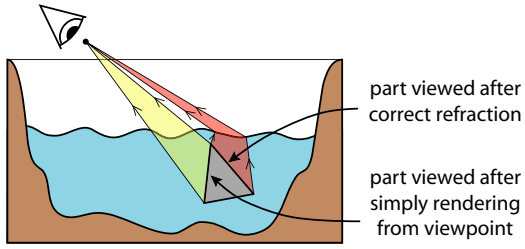


Figure 6: *Objects in water are difficult to correctly display because of refraction.*

4. Simulated effects

We now discuss a list of effects that can be straightforwardly simulated with our approach. Each effect can be turned on

or off in the shaders, allowing for a tradeoff between realism and speed.

4.1. Underwater light absorption

Careful attention has to be paid to lighting phenomena occuring in water. As explained in [PA00], some of these phenomena, like light scattering, are difficult to handle. Thus lighting equations have to be simplified. We use the lighting model given by [PA00] : for a given wavelength λ, the radiance transmitted from a point p_g under water to a point p_w on the surface is given by :

$$L_\lambda(p_w,\vec{\omega}) = \alpha_\lambda(d,0)L_\lambda(p_g,\vec{\omega}) + (1-\alpha_\lambda(d,z))L_{d\lambda} \quad (1)$$

where $\vec{\omega}$ is the direction from p_g to p_w, $L_\lambda(p,\vec{\omega})$ is the outgoing radiance from the point p in the direction ω, d and z are respectiveley the distance and the depth difference between p_g and p_w, $L_{d\lambda}$ is a constant diffuse radiance computed from scattering measurements involving sky and sun colors, and $\alpha_\lambda(d,z)$ describes an exponential attenuation depending on traveled distance and depth :

$$\alpha_\lambda(d,z) = e^{-a_\lambda d - b_\lambda z} \quad (2)$$

where a_λ and b_λ are attenuations coefficients depending on water properties. The first term of the equation equates radiance coming from p_g attenuated with traveled distance, the second one equates the contribution of diffuse scattering, depending also on depth.

Remember that during the rendering process we compute the precise coordinates of p_g and p_w so traveled depth and distance are easy to compute. This makes the implementation of this lighting equation costless relative to other computations.

Light attenuation can vary with wavelength, producing visually compelling chromatic scales. Ideally the color spectrum has to be discretized and computations must be done per wavelength. Using only the three components of the RGB color space gives an appropriate approximation. This is done by computing the attenuation color :

$$\alpha(d,z) = (\alpha_R(d,z),\alpha_G(d,z),\alpha_B(d,z)) \quad (3)$$

and combining it component-wise with the incoming and diffuse colors. Precisely desired color variations (e.g. physically measured) can be obtained by using a precomputed 2D color texture containing sampled values for $\alpha(d,z)$.

In practice, the influence of depth (the $b_\lambda z$ term) is subtle and considering only traveled distance can be sufficient. Doing so simplifies the attenuation coefficient to :

$$\alpha_\lambda(d) = e^{-a_\lambda d} \quad (4)$$

which can now be sampled in a 1D color texture (Fig.7). With this simplification, $L(p_w,\vec{\omega})$ is the simple linear blending between $L(p_g,\vec{\omega})$ and L_d with respect to $\alpha(d)$.

Notice that a_λ can be seen as a distance scaling factor, so it can be scaled according to the scale of the water volume. If this value is large, it means that light will be quickly absorbed so the deep parts of the ground surface will not be visible. In that case it is useless to keep running on the refracted ray after a certain limit distance d_{max}. This distance can be defined by fixing an attenuation threshold ε under which the incoming ground color is considered to have a negligible contribution :

$$\forall\lambda, \quad e^{-a_\lambda d_{max}} < \varepsilon \quad (5)$$

hence simply computable as :

$$d_{max} = \max_\lambda\left(-\frac{\log(\varepsilon)}{a_\lambda}\right) = \frac{\log(\varepsilon)}{\min a_\lambda} \quad (6)$$

Figure 7: *An example of color variations induced by the exponential attenuation.*

4.2. Self-reflections

Reflections on the water surface can be separated into three classes : reflections of distant objects, reflections of near objects and reflections of the parts of the ground emerging from water (Fig. 8). The first class can be easily handled using an environment map whereas only heuristics exist for the second one. We call the third class *self-reflection*. This case is more specific : emerging parts of the ground are represented in the ground heightfield, which allows us to perform a rapid raytracing of the reflected ray, as we do for refracted rays. So to take these reflections into account, the reflected ray is intersected with the ground surface. If an intersection exists, the corresponding lighted ground color replaces the one that would be otherwise looked up in the environment map.

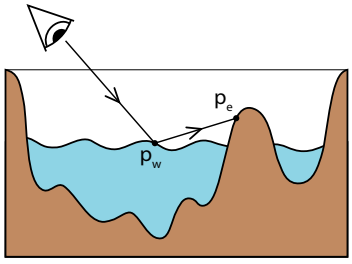


Figure 8: *Reflected ray hitting an emerging relief.*

One can notice that, as with refracted rays, we simplified the exact raytracing algorithm : the reflected ray could actually hit the water surface before reaching the ground or exiting the box, thus splitting in two new secondary reflected and refracted rays. But it is an indirect, difficult to witness, effect which doesn't affect realism, making this simplification reasonable.

4.3. Caustics and shadows

Computing caustics is hard to carry out in a forward raytracing approach because it requires us to count the amount of incoming light at each rendered point. That is why most existing caustics algorithms are based on backward raytracing : rays are cast from the light source instead of the viewpoint. We use a two-pass photon-mapping-like algorithm, involving GPU/CPU transfers of textures. Fortunately, we only need small textures and this does not impede the performance too much. The first pass renders the water surface from the light source into a *photon texture*. The texels of this texture record the coordinates of where the corresponding light rays hits the ground. Since the ground is a heightfield, we need only recording the (x, y) coordinates, which leaves the third channel of the texture available to store the photon contribution, based on the Fresnel transmittance, the traveled distance, the incident angle and a ray sampling weight. We apply a random jittering to light rays to reduce aliasing artifacts (see Figure 9). We then gather the photons recorded in

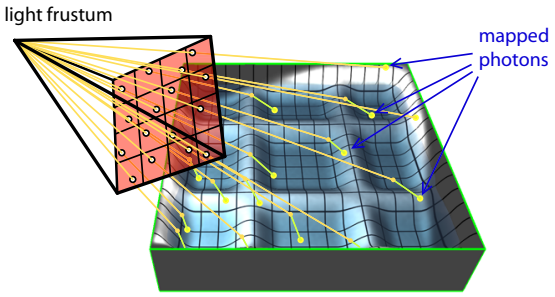


Figure 9: *Photons are emitted from jittered pixel positions and mapped on the ground surface.*

the texture. This boils down to computing a two-dimensional histogram of this texture. Unfortunately, it can not be done efficiently on the GPU because of the lack of *forward* mapping, so the photon texture is transfered to CPU where it is processed to construct an *illumination texture* (by analogy to illumination maps of [Arv86]). We traverse the texels of the photon texture, retrieve the position and intensity of the corresponding photon, and add this intensity to that stored at that position in the illumination texture. The illumination texture is then transferred to the GPU where it is used for lighting in the final render pass.

The illumination texture can be very noisy due to the crude sampling of our approach. Applying a simple gaussian blur would remove noise but would also blur caustics patterns which, by nature, present high frequencies. That is why edge preserving filters like anisotropic diffusion or bilateral filtering [TM98, DD02, WMM*04] are traditionally used to filter density maps in photon mapping techniques. We apply a GPU-based bilateral filter which gives good results once correctly tuned. The whole process is summarized in Figure 10.

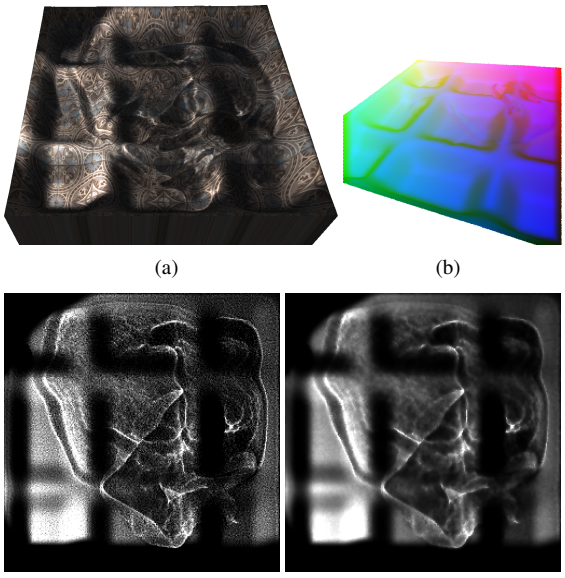


Figure 10: *To compute caustics and shadows (a), we render a photon texture from the light (b) and "invert it" into an illumination texture (c) which is bilaterally filtered (d).*

The resolutions for the two intermediate textures must be adjusted to balance the desired quality and framerate. The resolution of the photon texture directly determines the number of photons. The more photons are used, the more accurate the caustics will be. The illumination map can be seen as a lighting texture directly mapped on the ground, so its resolution affects the fineness of the reproduced caustic patterns. Note however that these two resolutions are linked : if too few photons are emitted, a large caustics map will be noisy. The resolutions are limited by the cost of their transfer between GPU and CPU memory. Using a 256×256 photon texture and a 128×128 illumination texture gives good results in practice.

This approach for computing caustics also accounts for shadows cast by the ground on itself, with refractions handled. Shadows of other objects in the scene onto the ground can not be cast that easily. Once again, the difficulty is to handle the refraction of rays. However, one could use a shadow map approach as a coarse approximation. It would integrate directly with our algorithm by performing a projective texture lookup into the shadow map, using the world position where the ray hits the ground.

5. Results and discussion

We implemented the proposed algorithm using GLSL shading language, and ran the example on a GeForce 7800FX. We implemented animation using [Gom00] with forces specified by random rain-drops or by mouse interactions

(see accompanying videos). Table 1 indicates performance measurments. The cost is highly dependent on the number

coverage	refraction	+ auto-reflection	+ caustics
100%	26 Hz	20 Hz	15 Hz
50%	45 Hz	36 Hz	23 Hz
25%	87 Hz	69 Hz	34 Hz

Table 1: *Framerates obtained at* 800×600 *for different screen coverages and with various effects turned on successively (rightmost column combines all effects).*

of fragments rasterized. For our tests, the displayed volume covers a large part of a 800×600 window, yet it renders in real-time. Another interest of the method is that its memory requirements are low: only a couple of textures are required. In all examples shown, we used 128×128 textures for the ground and water surfaces, yet they look appealing.

Figure 1 and Figure 11 show the kind of effects we can simulate. We believe the accompanying videos should also convince that a high level of realism is achieved with this technique. In particular, relevant effects are simulated, and the illusion of water is convincing.

6. Conclusion and future work

In this paper, we show how the encoding of water volumes with two heightfields allows for real-time rendering of realistic water. Using an efficient ray-tracing approach for such a representation, we show that many effects can be combined : single bound refractions and reflections, fresnel effect, light absorption, caustics and shadows. None of these effects are new; the contribution of this paper is to validate the amenability and suitability of heightfields in this context. Each effect can be turned on or off depending on the desired tradeoff between speed and quality. The method is based on simplifications of the underlying physical phenomena, yet gives very appealing results. The rendering cost is directly proportional to the screen occupancy of the displayed water. Thus, we believe it can be of great interest for rendering elements of a virtual world such as aquariums, ponds, muddy puddles, fountains, etc. in a much more realistic way that previously. Yet, even at large resolutions, it is still fast and may also prove useful for sea and lake rendering.

In the future, we would like to investigate a couple of optimizations (we plan to release the shaders and a detailed description of the formulae used). We also want to generalize the approach to other objects that combine a transparent layer in front of an opaque one, such as windows in a city walkthrough. We also want to tackle the problem of geometry-based objects that penetrate the water, for which we investigate approaches that dynamically "project" the geometry onto the heightfields.

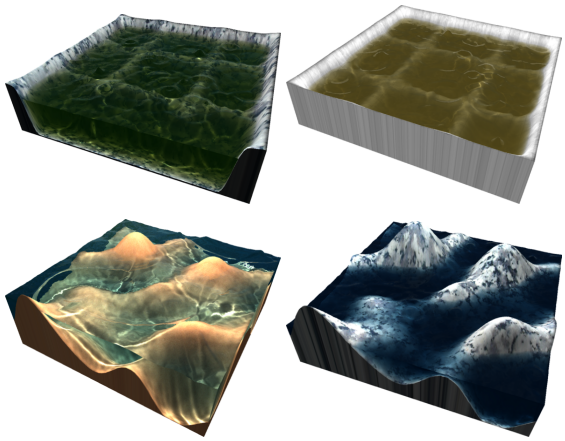


Figure 11: *Examples of renderings obtained with our technique.*

References

[Arv86] ARVO J.: Backward ray tracing. In *Developments in Ray Tracing (SIGGRAPH '86 Course Notes)* (Aug. 1986).

[BD06] BABOUD L., DÉCORET X.: Rendering geometry with relief textures. In *Graphics Interface '06* (2006).

[DD02] DURAND F., DORSEY J.: Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2002) 21*, 3 (July 2002), 257–266.

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2006), ACM Press, pp. 93–100.

[EAMJ05] ERNST M., AKENINE-MÖLLER T., JENSEN H. W.: Interactive rendering of caustics using interpolated warped volumes. In *Graphics Interface 2005* (2005), pp. 87–96.

[EMF02] ENRIGHT D., MARSCHNER S., FEDKIW R.: Animation and rendering of complex water surfaces. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2002) 21*, 3 (July 2002), 736–744.

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *SIGGRAPH 2001* (2001), pp. 23–30.

[FR86] FOURNIER A., REEVES W. T.: A simple model of ocean waves. *Computer Graphics (Proc. of SIGGRAPH '86) 20*, 4 (Aug. 1986), 75–84.

[FvDFH90] FOLEY J. D., VAN DAM A., FEINER S. K., HUGHES J. F.: *Computer Graphics: Principles and Practice*, 2nd ed. Addison-Wesley, 1990, ch. Illumination and shading, pp. 758–759.

[GLD06] GÉNEVAUX O., LARUE F., DISCHLER J.-M.:

Interactive refraction on complex static geometry using spherical harmonics. In *Symposium on Interactive 3D Graphics and Games* (2006), pp. 145–152.

[GM02] GAMITO M. N., MUSGRAVE F. K.: An accurate model of wave refraction over shallow water. *Computers & Graphics 26*, 2 (2002), 291–307.

[Gom00] GOMEZ M.: *Game Programming Gems*. Charles River Media, Inc., 2000, ch. Interactive simulation of water surfaces, pp. 187–194.

[HDE80] HASSELMANN D., DUNCKEL M., EWING J.: Directional wave spectra observed during JONSWAP 1973. *Journal of Physical Oceanography 10* (Aug. 1980), 1264–1280.

[HLCS99] HEIDRICH W., LENSCH H., COHEN M. F., SEIDEL H.-P.: Light field techniques for reflections and refractions. In *Rendering Techniques '99 (Proc. EG Workshop on Rendering)* (June 1999), pp. 187–196.

[HNC02] HINSINGER D., NEYRET F., CANI M.-P.: Interactive animation of ocean waves. In *Symposium on Computer Animation* (2002), pp. 161–166.

[HVT*04] HU Y., VELHO L., TONG X., GUO B., SHUM H.: Realistic, real-time rendering of ocean waves. *Computer Animation and Virtual Worlds* (2004). Special Issue on Game Technologies.

[IDN03] IWASAKI K., DOBASHI Y., NISHITA T.: A volume rendering approach for sea surfaces taking into account second order scattering using scattering maps. In *2003 Workshop on Volume Graphics* (2003), pp. 129–136.

[Jen01] JENSEN H. W.: *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., 2001.

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. *Computer Graphics (Proc. of SIGGRAPH '90) 24*, 4 (Aug. 1990), 49–57.

[MWM87] MASTIN G. A., WATTERBERG P. A., MAREDA J. F.: Fourier synthesis of ocean scenes. *IEEE Computer Graphics & Applications 7*, 3 (1987), 16–23.

[NN94] NISHITA T., NAKAMAE E.: Method of displaying optical effects within water using accumulation buffer. In *SIGGRAPH '94* (1994), pp. 373–379.

[PA00] PREMOË S., ASHIKHMIN M.: Rendering natural waters. In *Pacific Conference on Computer Graphics and Applications* (2000), p. 23.

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P.: Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics 21*, 3 (July 2002), 703–712. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[Pea86] PEACHEY D. R.: Modeling waves and surf. *Computer Graphics (Proc. of SIGGRAPH '86) 20*, 4 (Aug. 1986), 65–74.

[PM64] PIERSON W. J., MOSKOWITZ L.: A proposed spectral form for fully developed wind seas based on the similarity theory of S. A. Kitaigorodskii. *Journal of Geophysical Research 69* (Dec. 1964), 5181–5191.

[PMDS06] POPESCU V., MEI C., DAUBLE J., SACKS E.: Reflected-scene impostors for realistic reflections at interactive rates. *Computer Graphics Forum (Proc. of Eurographics 2006) 25*, 3 (2006).

[POC05] POLICARPO F., OLIVEIRA M. M., COMBA J. L. D.: Real-time relief mapping on arbitrary polygonal surfaces. In *Symposium on Interactive 3D graphics and games* (2005), pp. 155–162.

[Sch93] SCHLICK C.: A customizable reflectance model for everyday rendering. In *Fourth Eurographics Workshop on Rendering* (1993), pp. 73–84.

[SKALP05] SZIRMAY-KALOS L., ASZÓDI B., LAZÁNYI I., PREMECZ M.: Approximate ray-tracing on the gpu with distance impostors. *Computer Graphics Forum (Proc. of Eurographics 2005) 24*, 3 (2005).

[Sta03] STAM J.: Real-time fluid dynamics for games. In *Game Developer Conference* (Mar. 2003).

[Tat06] TATARCHUK N.: Dynamic parallax occlusion mapping with approximate soft shadows. In *Symposium on Interactive 3D graphics and games* (2006), pp. 63–69.

[TB87] TS'O P. Y., BARSKY B. A.: Modeling and rendering waves: wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Transactions on Graphics 6*, 3 (1987), 191–214.

[Tes99] TESSENDORF J.: Simulating ocean water. In *Siggraph Course Notes* (1999), ACM Press.

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *International Conference on Computer Vision* (1998), p. 839.

[Wat90] WATT M.: Light-water interaction using backward beam tracing. In *SIGGRAPH '90* (1990), pp. 377–385.

[WD06] WYMAN C., DAVIS S.: Interactive image-space techniques for approximating caustics. In *Symposium on Interactive 3D graphics and games* (2006), pp. 153–160.

[WMM*04] WEBER M., MILCH M., MYSZKOWSKI K., DMITRIEV K., ROKITA P., SEIDEL H.-P.: Spatio-temporal photon density estimation using bilateral filtering. In *Computer Graphics International* (2004), pp. 120–127.

[WS03] WAND M., STRASSER W.: Real-time caustics. *Computer Graphics Forum (Proc. of Eurographics 2003) 22*, 3 (2003).

[YYM05] YU J., YANG J., MCMILLAN L.: Real-time reflection mapping with parallax. In *Symposium on Interactive 3D graphics and games* (2005), pp. 133–138.