Real-time Rendering of Enhanced Shallow Water Fluid Simulations

Jesús Ojeda[a], Antonio Susín[b]

[a]*Dept. LSI, Universitat Politècnica de Catalunya*
[b]*Dept. MA1, Universitat Politècnica de Catalunya*

Abstract

The visualization of simulated fluids is critical to understand their motion, with certain light effects restricted or with added computational complexity in the implementation if real-time simulation is required. We propose some techniques that improve the rendering quality of an enhanced shallow waters simulation. To improve the overall appeal of the fluid representation, lower scale details are added to the fluid, coupling external non-physical simulations, and advecting generated surface foam. We simulate caustics by raytracing photons in light and screen-space, and apply refraction and reflections also in screen-space, through a number of render passes. Finally, it is shown how a reasonably sized fluid simulation is executed and rendered at interactive framerates with consumer-level hardware.

Keywords: real-time reflections and refractions, real-time caustics, fluid rendering

1. Introduction

Photorealistic rendering is still quite demanding for interactive applications due to its computational complexity. Otherwise, given enough time, offline renderers can easily generate this kind of imagery, usually using some algorithm of the raytracing family.

In the real-time field, however, GPUs are used which implement rasterization algorithms. These algorithms rely on high coherency for the operations executed, which impose some constraints to simulate light as a raytracer could do, by simulating each separate light beam. For this reason, photorealistic rendering is achieved at interactive framerates by simplifying the algorithms used or even using tricks that are perceptually feasible.

This simulation of light behaviour is required if a realistic fluid visualization is pursued. Liquids, in their vast majority, exhibit reflection and refraction effects, which in turn may also result in caustics. Assuming a visualization over the fluid, for great volumes of water, as open sea scenes, the major visual effects one could expect may be the refraction of the underlying terrain with projected caustics, as well as reflected scenery from above the fluid.

With present fluid simulations being performed in GPUs at interactive framerates, we also need realistic visualizations which reproduce these effects of the light. In our case, we start from a heightfield fluid simulation enhanced with particles for the simulation of splashes in breaking wave conditions, like the ones proposed in [1] or [2], which are also fully coupled with dynamic objects. From there, we aim to provide these expected, light-based effects, namely refractions, reflections and caustics. As the fluid simulation mesh may have lower resolution than that expected for high quality results, it is also improved with other techniques as lower scale details and surface foam advection which effectively increase the general appeal of the rendered scenes. The key contributions we propose are

- An extension of the technique from [3] to screen-space, following an initial photon search in light-space, as well as some other modifications for the simulation of caustics.
- A screen-space technique to simulate refractions and reflections, based in raycasting through depth-maps.
- Texture-based techniques for additional surface effects using FFT ocean simulation or Perlin noise, as well as the advection of surface foam generated at the splash particles reintroduction.

The result of these contributions is exemplified in Figure 1, and how they are interlaced as an overall algorithm can be seen in Figure 2.

1.1. Related Work

There are two common approaches to simulate fluids: eulerian and laplacian. The first simulate the fluid inside a grid. In the second, the fluid is implicitly represented by a particle system. For a full 3D fluid simulation we can find many references of both approaches but for brevity reasons we refer the reader to [4], [5] and references therein for greater fluid overviews.

In the specific case of eulerian fluid simulation, the fluid is usually represented as a scalar field and its visualization is done by raycasting the volume or by using mesh-extracting techniques like marching cubes for further use. Nevertheless, 3D full simulation can be still quite costly, so other solutions as

Email addresses: jojeda@lsi.upc.edu (Jesús Ojeda), toni.susin@upc.edu (Antonio Susín)

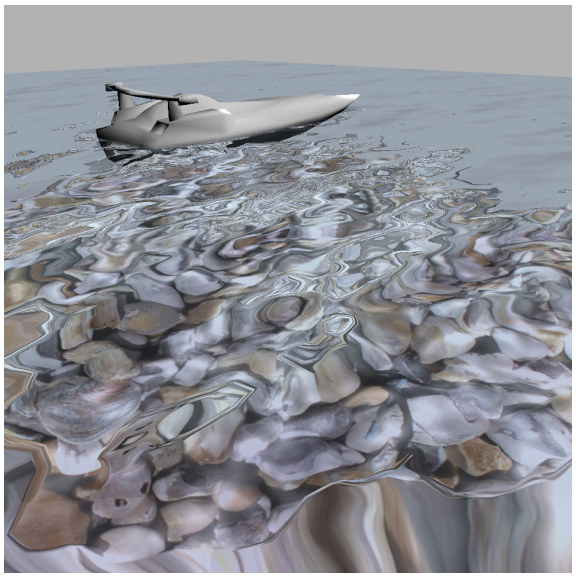


Figure 1: Caustics on the underlying terrain can be seen through the refractive surface of the fluid.

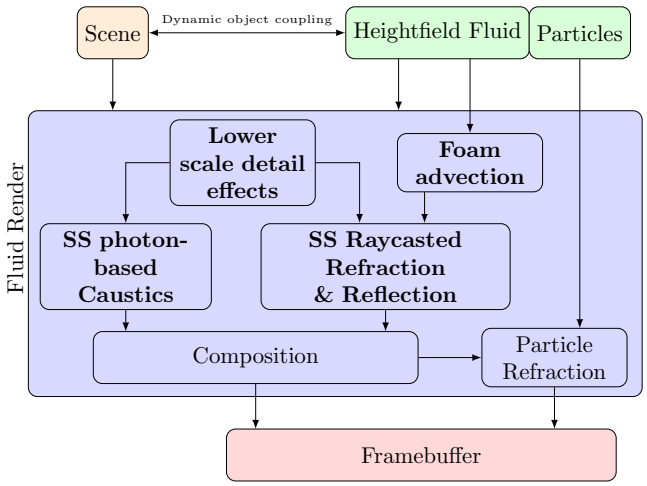


Figure 2: Pipeline of the different parts involved in the rendering of enhanced shallow water simulation with particles, providing our screen space photon-based caustics, screen-space raycasted refraction and reflection, as well as other surface effects as lower scale details and foam advection.

heightfield representations are more commonly used in the industry of real-time applications. Such simulations can come from procedural methods like the FFT ocean simulation [6], wave trains [7] or even physical frameworks as the Shallow Water equations [8, 2, 1]. Their result, can be easily represented as a triangle mesh, where vertex heights are provided by the own simulation.

As these grid-based approaches have a fixed resolution, in order to increase the perceived level of detail, other techniques have been applied as the advection of additional textures to simulate flow [9], coupled with normal mapping as in [2].

To finally visualize the fluid, several light-induced effects have to be considered. One of these effects are caustics, which are a very distinguishable effect from any refractive or reflective surface. Starting from Kajiya's work [10], caustics have been traditionally implemented with global illumination techniques like pathtracing, the metropolis light transport method [11] or photon mapping [12]. These techniques require a high count of rays or photons to achieve soft caustics, which relegate them to off-line rendering, although there are already GPU implementations of some of them like, e.g., [13, 14].

In the real-time domain, [15] was the first to explore caustics using synthetic texture maps; although inaccurate, they were visually compelling. Nevertheless, to achieve physically realistic results, the more recent techniques are inspired in pathtracing methods and can be generally classified in two groups. In the first group, techniques like, e.g, [16, 17, 3, 18], render from light and create caustic maps, similar to photon maps, but used like shadow maps; reprojected in camera space in order to lit the visible pixels that receive caustics. In the second group, the caustics are traced back from the receiver object to the light through limited areas on the refractive surface as in [19, 20], which usually require the receiver to be planar as a simplification.

Similarly, for the simulation of refractive or reflective materials, the ground truth may be reached with the usual pathtracing techniques but in the real-time domain trick techniques, like [21] which apply a random offset to the refracted vector, are commonly used. [16] on the other hand relies on environment maps as distance impostors to achieve approximate refraction. However, for a more physically accurate approach, the more complete techniques involve tracing rays through depth maps. In this sense, [22] simulated refraction using front and back depth maps, while later [23] improved on the previous technique by repeating the search in between depth maps to simulate total internal refraction. [24] also worked upon [22], improving it with depth corrections, impostors and caustics.

In contrast, we provide a full system for caustics, reflections and refractions as well as other effects to complete the fluid rendering. For the caustics, we improve upon [3], adding a second raycast phase in screen space (from the camera) to the first one used in light space. Furthermore, we simplify their approach by not generating a caustics map, but splatting the photons on the receiving geometry. In the reflection and refraction case, we specialize the refraction approach of [22, 23] to our fluid scenes: we render the geometry over and below the fluid separately and apply the same raycast algorithm to both buffers,

combining the results using Fresnel terms.

2. Fluid simulation

In our case, we use the fluid solver from [1], the simulation algorithm is based on the the Lattice Boltzmann Method (LBM) for Shallow Waters. The fluid is simulated in a grid and the interactions between the fluid molecules (described as distribution functions f_i) are modeled as collisions. Using the D2Q9 model, the LBM with the popular BGK collision operator [25] can be defined with the following equation:

$$f_i(\mathbf{x} + \mathbf{e}_i\Delta t, t + \Delta t) = f_i(\mathbf{x}, t) - \omega(f_i - f_i^{eq}) + \mathcal{F}_i, \qquad (1)$$

where ω is the relaxation parameter related to the viscosity of the fluid, $\mathcal{F}_i$ are external forces and f_i^{eq} is the equilibrium distribution function defined as

$$f_i^{eq}(h, \mathbf{u}) = \begin{cases} h\left(1 - \frac{5}{6}gh - \frac{2}{3}\mathbf{u}^2\right), & i = 0, \\ \lambda_i h\left(\frac{gh}{6} + \frac{\mathbf{e}_i\cdot\mathbf{u}}{3} + \frac{(\mathbf{e}_i\cdot\mathbf{u})^2}{2} - \frac{\mathbf{u}^2}{6}\right), & i \neq 0, \end{cases} \qquad (2)$$

where $\lambda_i = 1$ for $i = 1..4$ and $\lambda_i = 1/4$ for $i = 5..8$. g is the gravity and h and $\mathbf{u}$ are the fluid properties: height level from the underlying terrain and velocity, respectively. They are calculated as

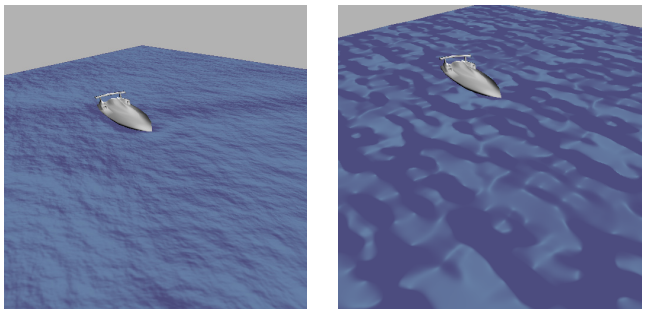
$$h(\mathbf{x}, t) = \sum_i f_i, \qquad (3)$$

$$\mathbf{u}(\mathbf{x}, t) = \frac{1}{h}\sum_i \mathbf{e}_i f_i. \qquad (4)$$

This basic model is enhanced by applying breaking wave conditions, an additional particle system and two-way object coupling similarly to [2]. Except the coupling with external objects, simulated with the Bullet Physics library, the whole simulation is executed in CUDA and achieves interactive timerates. We refer the reader to [1] for a full review on the simulated fluid system, the breaking wave example shown in Figure 3.

As the fluid is provided as a heightfield, its basic visualization can be a triangle mesh representing the whole domain, being the vertices equally displaced in the xz plane and their y coordinate the value of the heightfield at that point. We use this representation, and apply some techniques that enable more complex visual effects as caustics and refraction, which aren't restricted to this Shallow Waters simulation and may be applied to other refractive/reflective surfaces. These techniques are explained in the next sections.

For the particles, we render them as points, expanded to quadrilaterals and use depth and normal replacement, similarly to [26]. For refraction, methods like [21, 27] can be used. In our case we use the first one, where an arbitrary offset is applied to the refracted vector from the particle surface normal and used to look up at the framebuffer; although the results of this arbitrary offset on the refracted vectors are not physically correct, they are perceptually feasible and simpler to implement than the latter one, for example.



(a) FFT ocean simulation. (b) Noise, 4 octaves of a fractal sum.

Figure 4: Adding lower scale details to the fluid surface by normal mapping. Refraction and reflection are deactivated.

3. Additional surface detail

The heightfield simulation has a fixed size resolution which imposes a limit on the detail scale that can be achieved. We can add other simulations that improve the details by changing the surface normals locally. Furthermore, other advected properties as, e.g., surface foam, can also be simulated and applied to the final visualization. This section will introduce how these effects are accomplished.

3.1. Lower scale detail

From the heightfield of the fluid surface we can extract appropriate normals although they are restricted to the simulation resolution. We can increase the detail of the fluid just using normal mapping. For example, [2] applied a normal map texture generated from the FFT ocean simulation by [6] and advected it as in [9].

The FFT ocean simulation from [6] is based on the computation of the Fourier amplitudes of a wave field. The final heightfield is obtained from the inverse FFT to those amplitudes. In our case, we compute the FFT each frame and obtain a normal map from its heightfield which is then applied to the fluid surface, as can be seen in Figure 4a.

An alternative to the FFT approach is the use of noise textures with the same goal at mind. We can use gradient noise, being Perlin noise [28] the more popular, to obtain heightfields and compute normal maps from them to apply to the fluid surface. With 3D noise, we can create the illusion of animation moving through one of the dimensions. However, noise textures have some inherent problems: it is not clear how to create a good water-like function and if tiling is required, the pattern repetitions are quite obvious, as shown in Figure 4b.

3.2. Surface Foam

In the real life situation where splashes are generated, like in breaking waves, it is most probable that foam is generated when these splashes hit the fluid bulk again.

In contrast to [2], where diffuse disks are generated and advected with the fluid when particles fall into the surface fluid again, we simplify the idea. Using a floating-point single component texture mapped to the surface fluid, we detect where a

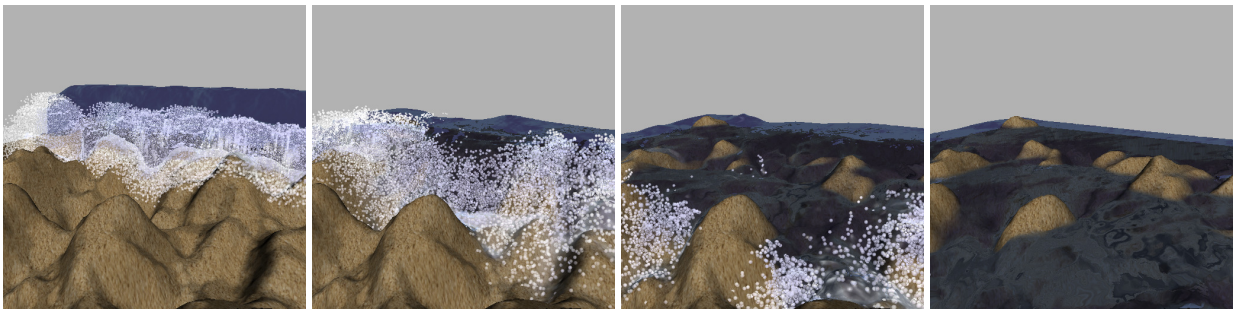


Figure 3: Breaking wave example from [1].

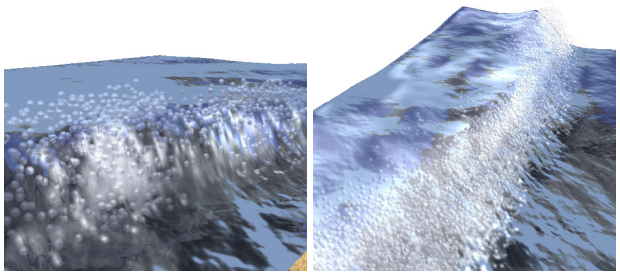


Figure 5: Foam is generated at particle-surface hit points and advected in successive frames using the fluid's velocity.

particle has fallen and initialize that texel to a certain maximum time-to-live (TTL) for the foam. This texture is then advected using the fluid's velocity field, tracing back as in [29]. Each frame, the values of the texture are decreased Δt until they become 0. These values are then multiplied with the desired foam color and mapped to the fluid mesh, resulting in the blended foam.

Using a texture for the foam introduces a constraint, however: its resolution should be dictated by the size of the particles as, it could happen that more than one texel should be initialized, depending on the particle to texel size ratio or, conversely, that the texels are too big for the particle size.

Overall, as seen in Figure 5, the results are convincing and the computations are faster due to the limited requirements, which make it ideal in real-time applications.

4. Photon-based Caustics

In order to add caustics to our real-time fluid simulation, we follow the same path of [3] and extend their work. They raycast a grid of photons, as points, through the scene in an orthographic space defined at the light source, which also allows them to easily add shadow mapping. One restriction they have is that the depth map used in the raycast phase should be continuous or, at least, with no great jumps. Other limitations this technique has are the same as image-based rendering: the results depend on the resolution of the textures used, which in this case restricts where the photons can end within the scene.

Our contributions to their algorithm imply extending the raycast of photons out of the light space to screen space, splatting them oriented with the surface of the receiving mesh. In contrast to [3], we do not generate a caustics map; the splats are blended with the scene, varying their intensity depending on the orientations of the caustic generating fluid position, as well as the distance the photon has travelled inside the fluid. As our fluid simulation is represented just by its surface, we restrict our approach to refracted photons which will fall in the underlying terrain and ignore reflected ones.

The multipass algorithm can be summarized in the following steps:

1. Render the objects of the scene (excluding the fluid) from the camera and store depth and normal maps.
2. Render the objects of the scene (excluding the fluid) from light with an orthographic projection and store the depth map.
3. Render the fluid from light with the same orthographic projection as before and store the world positions and refracted directions at each pixel.
4. Render the grid of photons. The primitives used are points which will be expanded to quadrilaterals when a final position is found.

This grid of points has the same resolution as the orthographic projection used previously in Step 2. In a vertex shader, the vertices will be raycast first in light space using the depth map from Step 2. If there is no intersection found, i.e., the photon exited through a wall of the frustum, the raycasting will be repeated in camera space. If there is not an intersection yet, the point is discarded (rendered out of frustum). Otherwise, if an intersection is found at light space, it is transformed to camera space and checked for correctness:

- If the point is occluded in camera space, it is discarded.
- Else, if the point is not occluded and the depth does not match between light and camera spaces, the raycasting continues from the actual point position in camera space.
- Else, the point is correct, that is, the depths match between light and camera spaces, thus the point is final.

When a final point is found, from the previous condition or from the camera raycasting, the normal is looked up in the normal map from Step 1. In a geometry shader, the points are expanded to quads oriented with their associated normal. Finally, in the fragment shader, the photons are textured with a

Gaussian splat, and their intensity is regulated depending on how they are facing the light and the distance they have travelled through the fluid until finally hit the receiving surface. At last, they are blended to the contents of the framebuffer.

We have not implemented shadows to keep the algorithm simple but, as suggested in [3], shadow mapping is easily added as the depth map from light is already stored for the raycasting.

As can be seen in Figure 6, the visual results are good enough for real-time rendering and the photons are not restricted to the light space. For a full physically-based render, the precise radiance of the photons should be computed. In order to make the algorithm more approachable, we just regulate the photons contributions with their orientation and user parameters as they are just blended with the framebuffer, which allow the technique to be faster in comparison, because we don't need the expensive operations for gathering the photons.

5. Screen-space Refraction and reflection

Similarly to the caustics approach, we implement refraction and reflection raycasting through depth maps, in the same way of [23].

For simplicity, we want to be able to use the same raycasting algorithm for both refractions and reflections, so we improve upon previous works by rendering in separate buffers what is above and below the fluid. This allows to, using the same code, just look for ray-depth intersection in the appropriate buffer to obtain the result and do a final composition with both refractions and reflection as needed.

In this case, rays are cast from camera and reflected or refracted (or both) when they hit the fluid, as shown in Figure 7.

Here, we also use a multipass algorithm that can be explained in the following steps, always rendering from camera:

1. Render the fluid mesh and store the depth buffer.
2. Using two render targets (RT) named 'over' and 'below', which will store color and depth, we render the objects of the scene (dynamic objects and ground in this case) and compare the depth with the previously stored. If the depth is greater, the fragment is stored in the 'below' RT, otherwise in the 'over' one. This pass can be thought as a stencil test, which separates what is above o below the fluid surface.
3. Render the fluid again, using the RTs. For each fragment of the fluid two rays are cast: one for refraction (using RT 'below'), one for reflection (using RT 'over'). As the rays start from the camera, if the reflected/refracted rays should come back, they are discarded. The results of both raycasts are combined using Schlick's approximation [30] to Fresnel terms for simplicity.
4. Finally, to avoid the repeated render of the other objects of the scene, we just use a screen-sized quadrilateral textured with the color buffer from the 'over' RT.

To reduce somewhat the need of the double raycasting, we can compute the Fresnel term from [30] prior to the raycastings at Step 3 as

$$F = F_0 + (1 - F_0)(1 - \theta)^5, \tag{5}$$

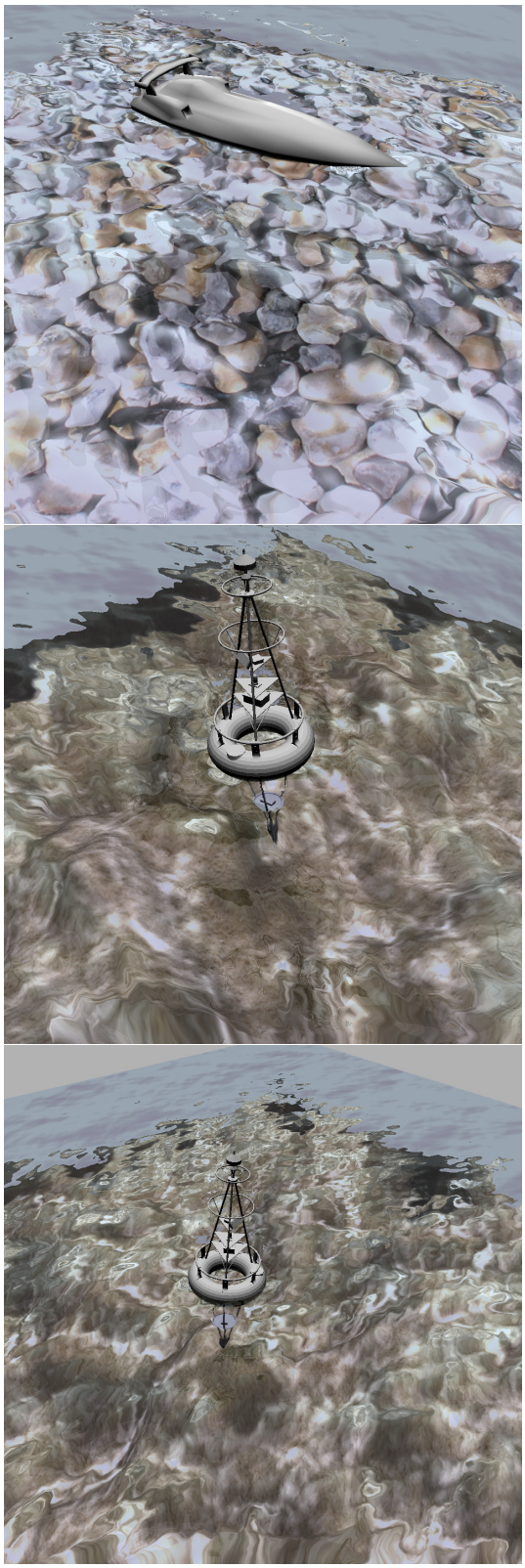


Figure 6: Caustics in the ground below the fluid surface with planar (boat) and noisy (buoy) terrain.

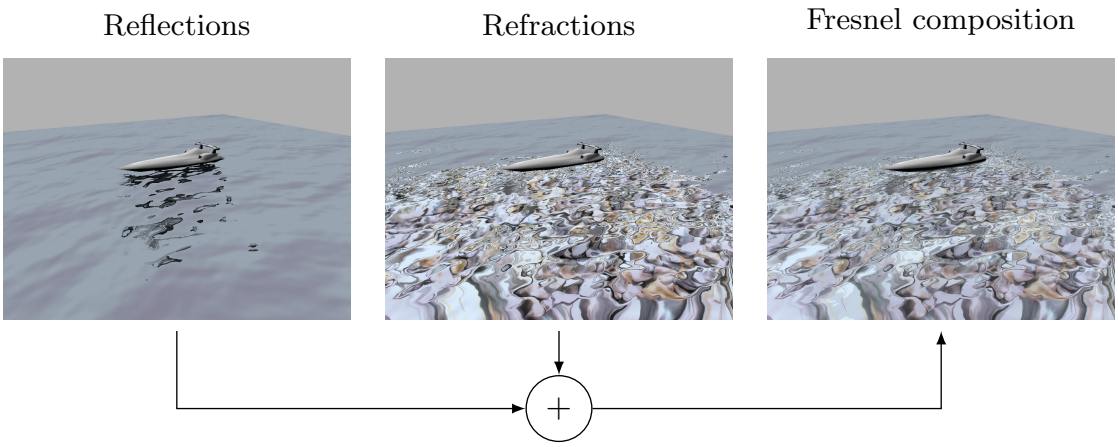


Figure 7: Reflections and refractions are found from the raycasting two different depth maps and finally composed using Fresnel for the final rendering.

being θ half the angle between the ingoing and outgoing light directions and F_0 the known value of F when $\theta = 0$, the reflectance at normal incidence. As we use the value F for a linear interpolation between the refracted and reflected colors, we can impose a threshold ϵ such as:

- If $F < \epsilon$, only the refraction raycasting is executed.
- If $1 - F < \epsilon$, only the reflection raycasting is done.
- Otherwise, both raycastings are done.

Additionally, for zones where the fluid height is quite low, controlled by an user parameter, we get the color from the direct view ray and interpolate from it to the combined color obtained from the previous algorithm, using the depth difference between the fluid and the ground below it. This alleviates some visible artifacts caused by the triangular mesh used for the fluid rendering, as shown in Figure 8.

Everything is done in screen-space, so there may be zones where there is not enough information, i.e., a ray should hit a point in space not visible; in those cases we detect the jump in the depth map and make use of the last pixel with information in the texture. Although this is really a problem due to lack of information, it may remain greatly unnoticed with the animated lower scale detail techniques of Section 3.1 and the own movement of the fluid surface.

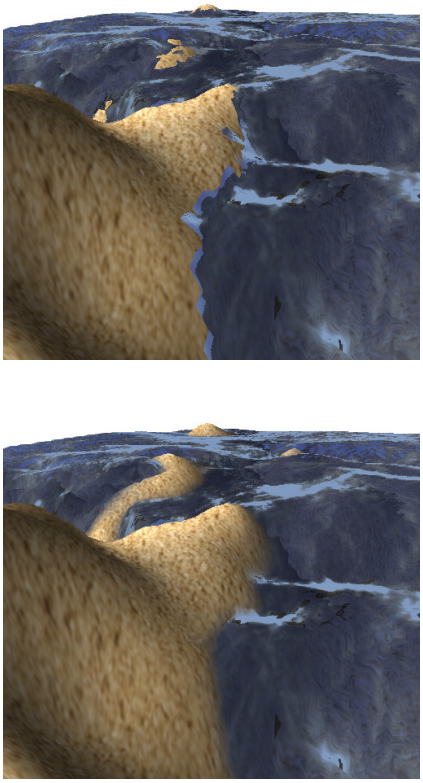


Figure 8: Artifacts from the fluid's triangular mesh on the left, alleviated on the right by interpolating the color value between the fluid's color and the ground color depending on the view distance from surface to ground.

6. Results and Discussion

We have tested the previous algorithms on an Intel Core2Duo E8400 with 4GB of RAM and a Nvidia GTX280 running Ubuntu 11.10. The resulting averaged timings of the caustics and refraction/reflection algorithms are shown in Table 1, as these are the ones that tax more on the GPU by the use of raycasting.

An improvement to the normal mapping for lower scale detail technique could be provided by also applying the technique from [9], in which multiple sets of texture coordinates are used and advected, already exploited in [2].

Viewport Size	Caustics Resolution	Caustics	Refraction & Reflection
512^2	128^2	1.0058	2.74639
	256^2	2.35144	2.79409
	512^2	7.92134	3.01034
	1024^2	51.3408	3.17178
1024^2	128^2	1.2585	5.79334
	256^2	3.33537	5.98238
	512^2	12.8643	6.21948
	1024^2	70.9195	6.53224

Table 1: Averaged timings in milliseconds for frame for the caustics and refraction/reflection algorithms. The Viewport column indicates the viewport resolution. Similarly, the Caustics Resolution column indicates the size of the viewport used for the orthographic camera, and thus, the number of photons traced.

The foam simulation from [2] could solve the fixed-size texture restrictions of our current solution, as they simulate foam directly with advected diffuse disks on the fluid surface, although this comes at the additional cost of generating and maintaining these disks on the fly. An alternative we believe would help our foam simulation is the use of a pyramidal texture approach; when particles fall to the fluid they initialize the correct level of the pyramid, being the other levels initialized extrapolating from that one.

Nevertheless, the timing results for both these techniques combined, the lower scale detail and the foam advection, never exceed the 2ms mark.

For the caustics, as shown in Table 1 and concluded in [3], the performance of the caustics algorithm depends primarily on the size of the grid of photons, but in our case also on the direction of the light, which can cause more photons to miss the light space raycast and use the second camera space one, thus increasing the number of computations and texture fetches needed to try to find a final position for them. Also, as the raycasting is done in the vertex shader it is further slowed down because of the increased penalty of texture fetches in that shader stage. Although a direct comparison with [3] is difficult because of the different hardware used, they reported to achieve about 200fps with a 128^2 photon grid, which is the same that saying that each frame costs 5ms to compute. With newer hardware but the dual light and camera space raycasts we propose, the cost of computing caustics is, in our case, below 2ms for the same configuration.

As the number of photons is limited, there may be zones over or undersampled; a hierarchical solution could help to solve this problem as shown in [18, 31], but we would require that it remains highly dynamic, as we are adressing the visualization of a moving fluid. Another thing worth researching would be to extend these caustics, if possible, to volumetric ones as those in [32].

The performance of the refraction/reflection algorithm is quite variable, it depends on the size of the viewport as well as the coverage of the fluid in screen: the more visible pixels, the more rays are cast. For fair comparison, the results in Table 1 were captured with the fluid covering the whole viewport, and even in this case, the whole algorithm does not cost more than 10ms for a reasonably sized viewport. In perspective, [23] made total internal refraction available although without surface reflection which, in the best case, reported 138fps, i.e., 7.24ms per frame on a Nvidia 8800 GTX, using only one bounce for internal refraction on a viewport of 512^2. Although our GPU is newer than theirs, in a similar scenario, we achieve less than half their time with both refraction and reflection.

Evidently, the restriction of the refraction/reflection algorithm being a screen-space technique limits how much information is available for such refractions and reflections. The simplest solution to this would be the use of environment maps, but, as the height of the fluid can be quite different across the domain, the position where the environment maps were generated would constraint, and even clip, possible geometry for correct refractions or reflections. Other alternatives should be considered to solve this limitation.

Both algorithms, caustics and refraction, may also suffer other performance penalties depending on the tessellation of the objects of the scene in question. This is due to the multipass character of the algorithms and the requirement of the rendering of the scene to obtain the depth maps for later raycasting. Although we have not encountered this problem in our tests due to low polygonal complexity, it should be worth having in mind. As a note, the boat model has 300 triangles, the buoy has 11k, the dolphin has 4k and the fluid and the ground have 32k triangles each.

Finally, the particles have just been rendered as billboards using depth and normal replacement with a sphere model. As they represent splashes, we want to maintain their crisp representation so, to improve their appeal, some additional tweaking could be done as applying some noise to their normals or deforming them in the direction they are moving to simulate some motion blur.

Finally, we have only taken into account the visualization of the surface of the fluid, as shown in Figure 9, in the future it should be also a key point to research water rendering as in, e.g., [33], in order to provide a full featured visualization.

7. Conclusions

In this paper we have presented a full pipeline of different algorithms for the rendering of heightfield-based fluid simulations coupled with particles, although the different parts can be applied to other situations as well.

The complex light-related effects like caustics, refractions and reflections have been adressed using raycasting techniques which ensure a more realistic simulation and the constraint of the algorithms to be in screen-space keeps the quantity of memory used low enough.

Additionally we have applied foam and lower-scale detail by applying textures to the fluid mesh; techniques which are very low demanding in comparison to the previous ones and really help to enhance the final result.

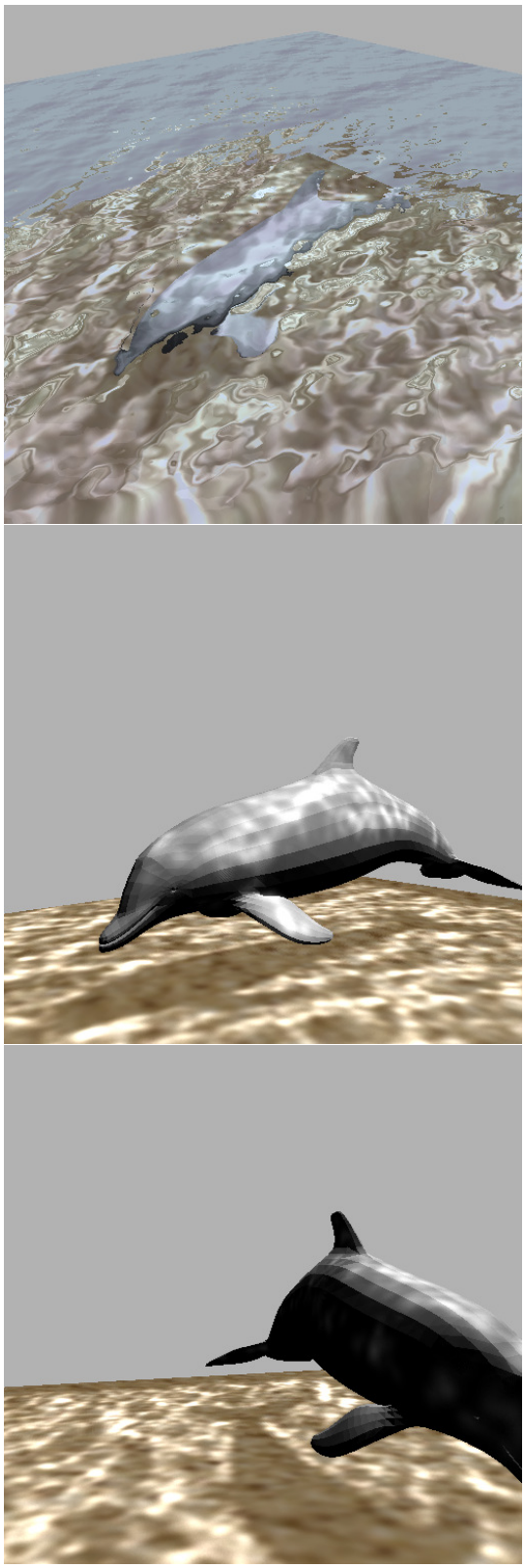


Figure 9: A dolphin underwater. Caustics are generated and projected on the dolphin and the terrain, visible from the surface.

Acknowledgements

We would like to thank Pere-Pau Vàzquez for his kind comments on the preparation of this work. With the support of the Research Project TIN2010-20590-C02-01 of the Spanish Government.

[1] Ojeda J, Susín A. Hybrid Particle Lattice Boltzmann Shallow Water for interactive fluid simulations. In: 8th International Conference on Computer Graphics Theory and Applications. GRAPP'13; 2013, p. 217–26.

[2] Chentanez N, Müller M. Real-time simulation of large bodies of water with small scale details. In: Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA). 2010, p. 197–206.

[3] Shah MA, Konttinen J, Pattanaik S. Caustics mapping: An image-space technique for real-time caustics. IEEE Transactions on Visualization and Computer Graphics 2007;13(2):272–80.

[4] Bridson R. Fluid Simulation for Computer Graphics. AK Peters; 2008.

[5] Solenthaler B, Pajarola R. Predictive-corrective incompressible sph. ACM Trans Graph 2009;28(3).

[6] Tessendorf J. Simulating ocean water. In: SIGGRAPH 2001 Course Notes. 2001,.

[7] Yuksel C, House DH, Keyser J. Wave particles. ACM Trans Graph 2007;26(3).

[8] Layton AT, van de Panne M. A numerically efficient and stable algorithm for animating water waves. The Visual Computer 2002;18(1):41–53.

[9] Max N, Becker B. Flow Visualization Using Moving Textures. In: Proceedings of the ICAS/LaRC Symposium on Visualizing Time-Varying Data. 1996, p. 77–87.

[10] Kajiya JT. The rendering equation. In: Proceedings of the 13th annual conference on Computer graphics and interactive techniques. SIGGRAPH '86; 1986, p. 143–50.

[11] Veach E, Guibas LJ. Metropolis light transport. In: Proceedings of the 24th annual conference on Computer graphics and interactive techniques. SIGGRAPH '97; 1997, p. 65–76.

[12] Jensen HW, Christensen PH, Kato T, Suykens F. A practical guide to global illumination using photon mapping. In: SIGGRAPH 2002 Course Notes. 2002,.

[13] Purcell TJ, Buck I, Mark WR, Hanrahan P. Ray tracing on programmable graphics hardware. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques. SIGGRAPH '02; 2002, p. 703–12.

[14] Nvidia . NVIDIA OptiX Application Acceleration Engine. `http://www.nvidia.com/object/optix.html`; 2012. [Online as of June-2013].

[15] Stam J. Random caustics: natural textures and wave theory revisited. In: ACM SIGGRAPH 96 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '96. SIGGRAPH '96; 1996,.

[16] Szirmay-Kalos L, Aszódi B, Lazányi I, Premecz M. Approximate ray-tracing on the gpu with distance impostors. Computer Graphics Forum 2005;24(3):695–704.

[17] Wyman C, Davis S. Interactive image-space techniques for approximating caustics. In: Proceedings of the 2006 symposium on Interactive 3D graphics and games. I3D '06; 2006, p. 153–60.

[18] Wyman C. Hierarchical caustic maps. In: Proceedings of the 2008 symposium on Interactive 3D graphics and games. I3D '08; 2008, p. 163–71.

[19] Guardado J, Sánchez-Crespo D. Rendering water caustics. In: GPU Gems. Addison-Wesley; 2004, p. 31–44.

[20] Yuksel C, Keyser J. Fast Real-time Caustics from Height Fields. The Visual Computer (Proceedings of CGI 2009) 2009;25(5-7):559–64.

[21] Sousa T. Generic refraction simulation. In: GPU Gems 2. Addison-Wesley; 2005, p. 295–305.

[22] Wyman C. An approximate image-space approach for interactive refraction. In: ACM SIGGRAPH 2005 Papers. SIGGRAPH '05; 2005, p. 1050–3.

[23] Davis ST, Wyman C. Interactive refractions with total internal reflection. In: Proceedings of Graphics Interface 2007. GI '07; 2007, p. 185–90.

[24] Hu W, Qin K. Interactive approximate rendering of reflections, refractions, and caustics. IEEE Transactions on Visualization and Computer Graphics 2007;13(1):46–57.

[25] Salmon R. The lattice boltzmann method as a basis for ocean circulation modeling. Journal of Marine Research 1999;57(3):503–35.

[26] Schaufler G. Nailboards: A rendering primitive for image caching in dynamic scenes. In: Proceedings of the Eurographics Workshop on Rendering Techniques '97. 1997, p. 151–62.
[27] van der Laan WJ, Green S, Sainz M. Screen space fluid rendering with curvature flow. In: Proceedings of the 2009 symposium on Interactive 3D graphics and games. I3D '09; 2009, p. 91–8.
[28] Perlin K. Improving noise. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques. SIGGRAPH '02; 2002, p. 681–2.
[29] Stam J. Stable fluids. In: Proceedings of the 26th annual conference on Computer graphics and interactive techniques. SIGGRAPH '99; 1999, p. 121–8.
[30] Schlick C. An Inexpensive BRDF Model for Physically-based Rendering. Computer Graphics Forum 1994;13(3):233–46.
[31] Wyman C, Nichols G. Adaptive caustic maps using deferred shading. Computer Graphics Forum 2009;28(2):309–18.
[32] Liktor G, Dachsbacher C. Real-time volume caustics with adaptive beam tracing. In: Symposium on Interactive 3D Graphics and Games. I3D '11; 2011, p. 47–54.
[33] Gutierrez D, Seron FJ, Munoz A, Anson O. Visualizing underwater ocean optics. Computer Graphics Forum 2008;27(2):547–56.