Solving Local Reflections: a Direct Methodology

P. Jeremias[1], O. Chavarria[1], O. Garcia[1], X. Carrillo[2] and A. Cuñado[2]

[1]Audiovisual Technologies Department, La Salle School of Engineering-URL, Spain
[2]Digital Legends Entertainment, Spain

Abstract

Reflection and refraction are key processes intended for realism within simulation platforms. There are several parameters related to those techniques whose values have been traditionally selected by using subjective criteria. This paper presents an interactive water rendering model where the adjustment of the local perturbation parameters has been revised, ensuring physical correctness. Therefore we present new tuning paradigms, based on physical, optical and, besides that, objective constraints.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

1. Introduction

One can find several approaches regarding the interactive simulation of the water phenomena. The differences are related to what is being simulated: waves, reflection and refraction maps, foam, spray and turbulences among other realistic effects. However it seems difficult to find an approach that clarifies the procedures involved in the appropriated perturbation of the texture coordinates, regarding reflection and refraction maps. Moreover, visual artifacts at intersection lines arise when combining arbitrary objects with water.

The animation and modeling of ocean waves has been traditionally related to the use of the 3D Navier-Stokes equations, like in [MMS04]. In those simulators, real-time is hardly accomplished although we can find some attempts like the interactive animation scheme presented in [HNC02], where the computations are restricted to the visible part of the ocean surface.

From another perspective, the particularities of fluids have been shown in recent papers like [HW04], where an hybrid method is applied to the simulation of turbulent water, with foam and spray effects. They achieved interactive speeds for simple scenarios. The authors in [MCG03] proposed a new particle-based method for the simulation process besides a marching cubes reconstruction algorithm for the final surface. [SSK05] presents another approach regarding the simulation of fluids. There, physically-based algorithms are used for the modeling of small-scale fluids and their interactions within the simulated environment.

The reflection and refraction paradigms have been intensively tested [IDN03] although there are several key topics to be refined. [Kry05] presents a water model built from height maps and render-to-texture techniques, like in our approach. However, the mechanism used for the tuning of the local perturbation parameters should be revised for the sake of physical correctness. Typical schemes sustain that the eye ray and the surface normals take an important role for the perturbation of the texture coordinates. Therefore some kind of methodology, related to physical and/or optical constraints should be defined.

Our technique finds solutions to the presented drawbacks at interactive rates. It is intended for videogames and real-time applications. The algorithm runs under pixel shader 2.0 (or upper) and within any vertex shader version.

2. Local Reflections

Typical paradigms apply constant distortion to their respective reflection implementations. Due to that, reflections break at the object's bases. In fact, less-distorted reflections don't show this behavior although minimizing them doesn't improve the final result.

If we apply distortions with arbitrary factors, which are essentially subjective, geometry dimensions and distortion sizes are not taken into account.

We present two distance-based solutions when computing reflections for geometries outside from the skybox. The algorithms avoid empty spaces at the intersection lines.

Figure 1 shows our first approach, the *Distance-Based Sphere (DBS)* method, which is intended for any types of shapes and geometries. **E** is the eye vector; **N** is the plane (water) normal; **N'** stands for the normal vector taken from the normal map; **P** is the point in the mesh that we are working with; **R** is the "underwater" mirroring of the reflection vector, around the plane normal; **R'** is the reflection vector associated to **N'**; **AP** is the approximated point, in **R'; RP** is the real geometry hit, whose projection should be placed at **P**; **ε** is the estimated error term; **d** is the distance from **P** to the first hit, along the **R** direction.

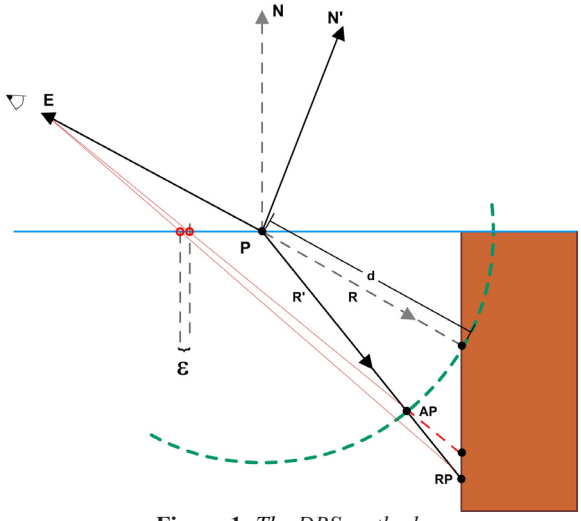

Figure 1: *The DBS method.*

We suppose that **d** (distance from each surface point (**P**) to the underwater geometries when using the correct reflection (**R**) associated to **N**) remains constant, even if using a modified normal like **N'** and a new reflection vector (**R'**).

The distances from each pixel to the underlying geometries are computed while rendering those reflected geometries within an auxiliary render target. This computation is based on an easy ray-plane collision. After that, the distance is saved in the alpha component of the reflected image. It comes clear that the render target should be previously initialized with a value representing the distance to the sky.

Figure 1 shows that from a conceptual view, it is possible to describe a circle centered at **P**, while using its associated distance (saved in its alpha channel in the texture) as the radius that collides with **R'**, finding its corresponding modified point at the surface. The error can be estimated by using two projections onto the water surface.

The water is rendered as a previously defined mesh. For each pixel, we compute the new reflected vector by taking the normal map generated from a height map, in real-time. With this reflected vector and the distance saved in the reflection map, we compute the new point and its coordinates above the water surface (projective mapping).

When using vertical geometries, it might be less accurate to use the *DBS* paradigm. We take this into account with our second approach, which is still under consideration, the *Distance-Based Plane (DBP)* method, shown in figure 2. It computes a vertical plane for each of the pixels in the water surface. The plane is always orthogonal to the water surface and passes through the collision point (between **R** and the first hit in geometry). Therefore we find a new intersection between **R'** and this plane. It allows us to find the new surface point, which is used for the perturbation.

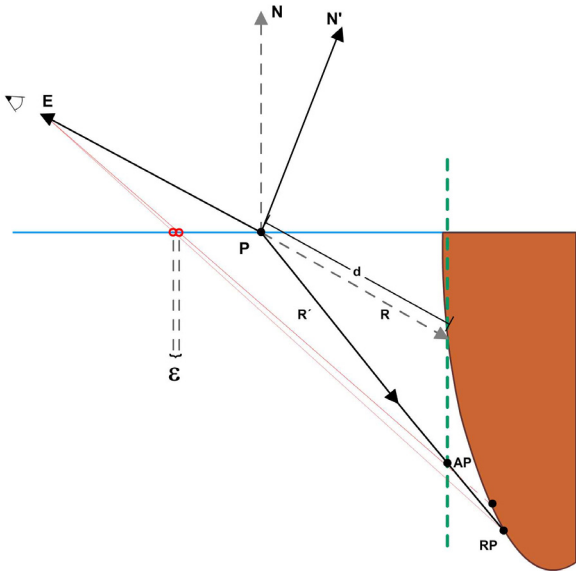

Figure 2: *The DBP method.*

3. Water Rendering Algorithm

Like other water rendering algorithms [Jen01], our render scheme uses mixed reflections and refractions (Fresnel's term) attenuated by a depth factor. However there are two special features: the local reflection approaches already presented and the use of two different Fresnel terms (check figures 3 and 4).

Where $\mathbf{n_1}$ is air and $\mathbf{n_2}$ stands for water; $\mathbf{N_1}$ and $\mathbf{N_2}$ are the normal vectors for each case; $\mathbf{L_1}$, $\mathbf{R_1}$, $\mathbf{T_1}$, $\mathbf{L_2}$, $\mathbf{R_2}$ and $\mathbf{T_2}$ depict the different eye, reflection and transmission vectors; α and θ are the angles related to the Snell's law [WP05].

In figure 3, the incidence angle α allows the calculation of the Fresnel value used for the mixing of the rays converging at the eye position, due to the reflection (rays along the $\mathbf{L_1}$ direction).

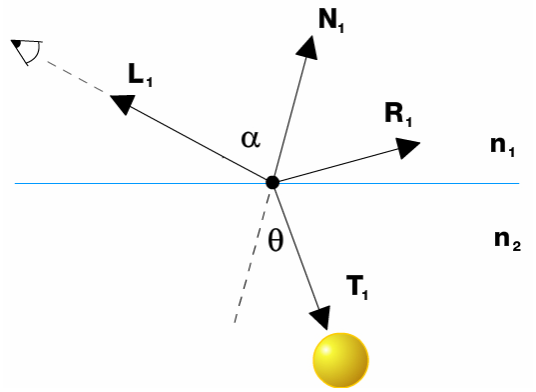

Figure 3: *first Fresnel term with rays from air to water.*

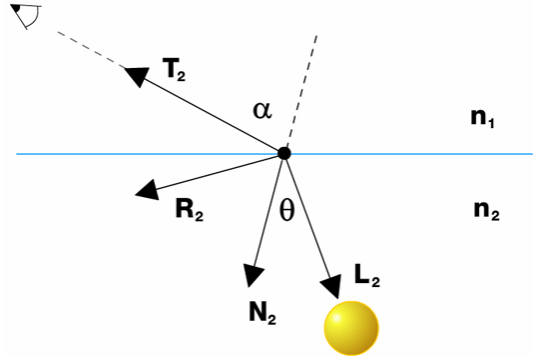

Figure 4: *second Fresnel term with rays from water to air.*

Figure 4 shows how the same concept arises but light travels from water to air. The final result receives some contribution from the underwater light coming from the $\mathbf{L_2}$ direction, with an incidence of θ degrees and refraction within the $\mathbf{T_2}$ direction. θ allow the evaluation of a second Fresnel coefficient. From the information inferred from the angles, we create a texture containing all the coefficients in different channels. Then we only need to read once for a particular angle of incidence (figure 5). This is because we need to reduce the amount of shader instructions.

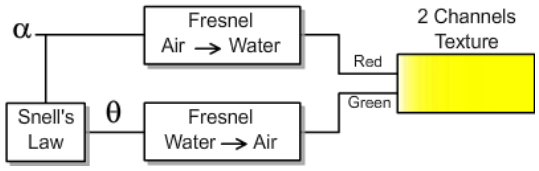

Figure 5: *Two Fresnel terms.*

4. The Algorithm: Step by Step

The following steps depict implementation details for the proposed solutions:

- Store the water clipping plane information as a matrix. The plane clips the geometry under, or above, the water surface (either in the reflection or in the refraction maps).
- Compute the reflection map using a render target.
  - Due to the normal vectors at the water surface, we need to find a projection matrix with a bigger field-of-view. The scene might reflect objects outside from the initial view.
  - Save the distance along the viewing direction between geometry and the water surface (as an alpha channel).
- Compute the refraction map as above.
- Render the water surface:
  - Evaluate the eye vector within the vertex shader.
  - In the pixel shader framework, find the perturbed reflected vector using a local reflection approach (either *DBS* or *DBP* methods).
  - Change texture coordinates according to the perturbation scheme and find the refraction color.
  - Scale the reflection color using the Fresnel term.
  - Attenuate the refraction color exponentially (fog equation and water density).
  - Scale the refraction color using a second Fresnel term.
  - Find the global water color by mixing the reflection and refraction colors previously found.

5. Results

Table 1 shows some comparison results for the *DBS* method, tested on two different stations: a 3.4 Ghz Intel Pentium 4 CPU with 1Gb of system RAM and a NVIDIA GeForce 6600GT PCIE with 256 Mb of video memory; and an equal PC except for the video card (NVIDIA 7800GT Dual GeForce with 512 Mb of video memory). The scene's resolution is 1024 x 768 (55175 polygons: 32768 for the water and 22407 for the geometry) and there are no post-processes running.

	Fps DBS	Fps Typical
6600 GT	86	94
7800 GT	215	221

Table 1: *Frame rate statistics: DBS vs. Typical* [Lom04].

Figures 6 and 7 show some rendering results (using the second graphics card above).

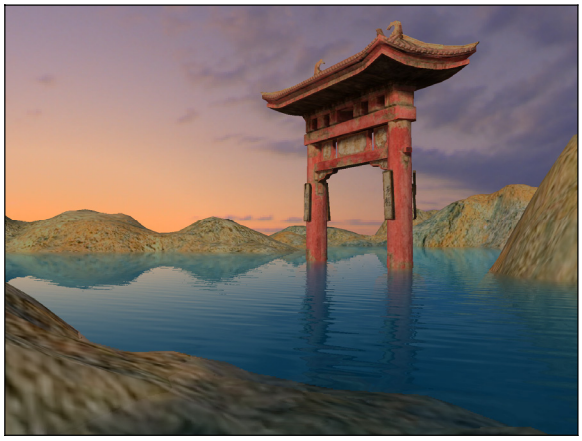

Figure 6: *The DBS method in real-time.*

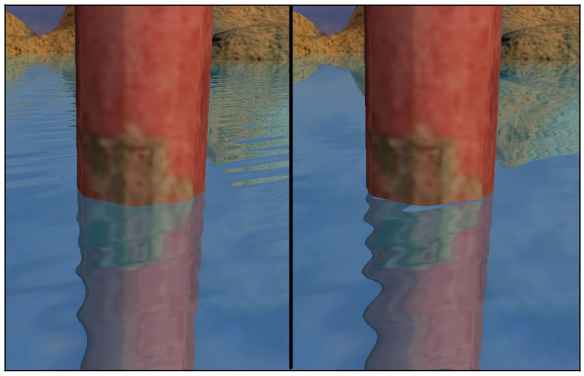

Figure 7: *Comparison results: DBS (left) vs. typical (right)*

6. Conclusions and Future work

Our method provides with a direct methodology that solves distortion problems regarding water reflection in real-time besides providing with an objective approach for the parameterization.

Our experiments show that it is possible to implement an iterative version of the *DBS* method. Nevertheless, the associated GPU cost doesn't compensate.

Future guidelines include an intensive testing of the *DBP* method and improving the water rendering. We'd like to adapt the presented schemes to the water refraction process; use shader model 3; apply displacement mapping and finally, clip the scene using the geometry and not only a plane.

7. Acknowledgements

We'd like to thank the Digital Legends Entertainment artists for helping us with their nice 3D models.

References

[HNC02] HINSINGER D., NEYRET F., CANI M.P.: Interactive animation of ocean waves.
Symposium on Computer Animation (July 2002),
Pages 161–166. (Proc. Eurographics Symposium on Computer Animation'02).
Session: Natural phenomena
ISBN: 1-58113-573-4

[HW04] HOLMBERG N., WÜNSCHE B.C.: Efficient modeling and rendering of turbulent water over natural terrain.
Computer graphics and interactive techniques in Australasia and South East Asia (June 2004)
Pages 15–22. (Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia'04).
Session: Rendering
ISBN: 1-58113-883-0

[IDN03] IWASAKI K., DOBASHI Y., NISHITA T.: A Fast Rendering Method for Refractive and Reflective Caustics Due to Water Surfaces.
Annual Conference of the European Association for Computer Graphics - Eurographics (September 2003),
Volume 22 , Issue 3.

[Jen01] JENSEN L.: Deep Water Animation and Rendering.
In
http://www.gamasutra.com/gdce/2001/jensen/jensen_01.htm (September 26, 2001)

[Kry05] KRYACHKO Y.: Using Vertex Texture Displacement for Realistic Water Rendering.
In GPU Gems 2 (March 2005),
Pages 283–294.
Topic: Shading, Lighting and shadows
ISBN: 0-321-33559-7

[Lom04] LOMBARD Y.: Realistic Natural Effect Rendering: Water I.
In
http://www.gamedev.net/reference/articles/article2138.asp (September 7, 2004)

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications.
Symposium on Computer Animation (July 2003),
Pages 154–159. (Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation'03).
Session: Procedural modeling & animation
ISBN: ~ ISSN: 1727-5288 , 1-58113-659-5

[MMS04] MIHALEF V., METAXAS D., SUSSMAN M.: Animation and control of breaking waves.
Symposium on Computer Animation (August 2004),
Pages 315–324. (Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation'04).
Session: Natural phenomena
ISBN: ~ ISSN:1727-5288 , 3-905673-14-2

[SSK05] SONG O.Y., SHIN H., KO H.S.: Stable but non dissipative water.
ACM Transactions on Graphics (January 2005),
Volume 24 , Issue 1, Pages 81–97.
ISSN: 0730-0301

[WP05] Alan WATT A. And POLICARPO F.: Real-Time Animation.
In Advanced Development with programmable graphics hardware (2005),
Pages 229–237.
Topic: Reflection and refraction
ISBN: 1-56881-240-X