Realistic, Real–Time Rendering of Ocean Waves

Yaohua Hu[*] Microsoft Research
Luiz Velho[†] IMPA
Xin Tong[‡] Microsoft Research
Baining Guo[§] Microsoft Research
Harry Shum[¶] Microsoft Research

Abstract

In computer games and other real-time graphics applications, the ocean surface is typically modeled as a texture or bump-mapped plane with simple lighting effects. This paper describes a system for realistically rendering the water surface in real time. Our system can render calm ocean waves with sophisticated lighting effects at 100 fps on a 680 MHz Pentium III with a GeForce 3 graphics card. The wave geometry is represented view-dependently as a dynamic displacement map with surface detail described by a dynamic bump map. The illumination model includes reflection, refraction, and Fresnel effects which are critical for producing the look and feel of water.

CR Categories: I.3.7 [Three-Dimensional Graphics and Realism]—Color, shading, shadowing, and texture; I.3.3 [Picture/Image Generation]—Bitmap and framebuffer operations;

Keywords: Reflectance, Shading Models, Real-Time Rendering

1 Introduction

Visualization of ocean surfaces is an important topic in computer graphics because it is an element present in many natural scenes. Although many successful techniques have been developed for realistic ocean simulation and rendering in the film industry, these techniques are only appropriate for off-line rendering of animated sequences. Real-time applications such as games typically model the ocean surface as a texture-mapped plane with simple lighting effects, while realistic wave geometry and sophisticated lighting effects such as reflection, refraction and Fresnel effects are ignored.

The techniques described in this paper make it possible to render in real-time (> 100 fps) the ocean surface using dynamic displacement and bump maps with Fresnel reflection and refraction. These techniques would be very useful for interactive real-time applications including visual simulators and games.

Our system can be implemented with vertex and pixel shaders in PC graphics cards and game consoles. The system is based on two key ideas. The first is *Fresnel bump mapping*, which is a technique for efficiently rendering per-pixel Fresnel reflection and refraction on a dynamic bump map using graphics hardware. The accurate rendering of Fresnel effects on a dynamic bump map plays a critical role in recreating the look and feel of the ocean water. Our second idea is a *view-dependent* representation of wave geometry. This representation can realistically describe geometry of nearby waves, while efficiently handling distant waves.

[*]e-mail:i-yaohhu@microsoft.com
[†]e-mail:lvelho@impa.br
[‡]e-mail:xtong@microsoft.com
[§]e-mail:bainguo@microsoft.com
[¶]e-mail:harry@microsoft.com

Previous methods either generate very realistic renderings of ocean scenes as an off-line process, or produce a simulation of the ocean in real-time but lacking sophisticated lighting effects. Currently there is not a practical solution for fully realistic real-time ocean simulation. This problem is addressed by our technique which applies well for simulation of calm ocean waves.

1.1 Previous Work

The classic reference on ocean waves rendering is the the work of [Fournier and Reeves 1986]. In [Tessendorf 2001] the fundamental principles for realistic simulation of ocean waves are presented. These principles include a spectral method for modeling wave geometry and a sophisticated lighting model of the ocean water. Such techniques are now widely used in the film industry for off-line rendering of animation sequences.

[Premoze and Ashikhmin 2001] presented a method for wave generation on water surfaces. They also presented a light transport approach for computing the complex lighting effects of the ocean. However, their physically-based approach for wave simulation and rendering is time-consuming and impractical for real-time applications.

A survey of different methods for deep water animation can be found in [Jensen 2001]. The author also discusses hardware acceleration techniques for ocean rendering.

[Schneider and Westermann 2001] introduced a method for real-time visual simulation of a water surface in a container. They used NURBS to model the water surface and environment cube mapping for reflection and refraction. The Fresnel term was approximately evaluated using a formula from [Schlick 1994].

Fresnel reflection using image-based techniques is discussed in [Cabral et al. 1999].

Recently, [Hinsinger et al. 2002] proposed an algorithm to animate the ocean waves in real time. Combined with a simple lighting model, the ocean waves (with 50×50 mesh resolution) can be rendered at 20 fps on a 800 MHz Pentium III. Unfortunately, the performance of their system is still lower than the performance requirement in real-time applications (ideally ≥ 60 fps). Furthermore, they do not simulate Fresnel effects.

2 System Overview

In this section we give an overview of our real-time ocean simulation system. Our pipeline can be divided into two stages, preprocessing and rendering, as shown in Figure 1.

In the preprocessing stage, we perform an ocean wave simulation to generate a dynamic displacement map and a dynamic bump map. The displacement map is used to model the wave geometry, while the bump map is applied during the rendering stage to enhance the details of the wave surface. The bump map is augmented with a pre-computed Fresnel texture for fast evaluation of the Fresnel term on the bump map. We also construct a view-dependent representation of ocean wave geometry in the preprocessing stage. This representation consists of a near patch and a far patch.

For each view, the rendering stage takes four passes on a GeForce 3 graphics card. The first two passes generate the reflection and refraction maps. The third pass calculates the shadow map for the sunlight. This pass is very low cost. The final rendering pass draws the view dependent wave geometry with bump mapping, reflection, refraction, Fresnel effects, sunlight reflection and shadows. The fact that the final rendering can be done in a single pass is very important for rendering speed because of the high fill rate of water rendering. This efficiency is attributable to our system design which carefully moves much of the work load to the preprocessing stage.

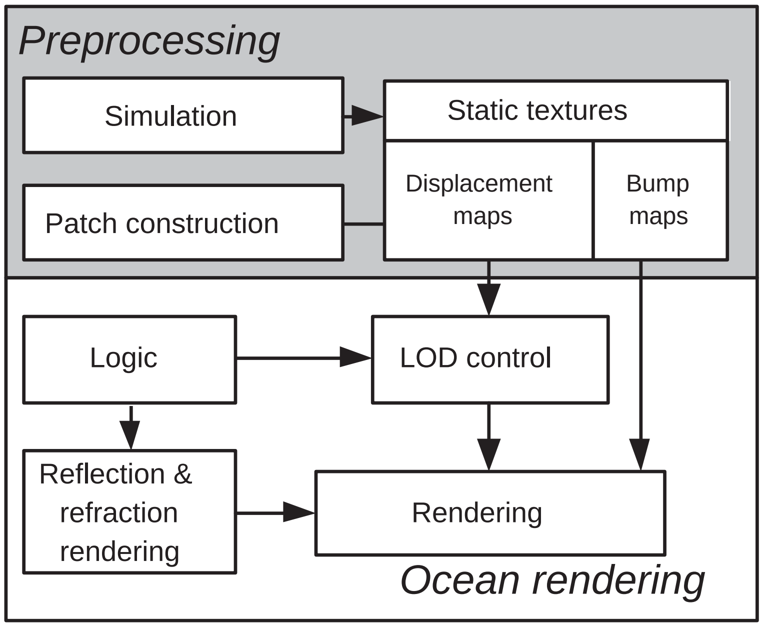


Figure 1: System overview.

2.1 Ocean Simulation and Patch Construction

We apply the spectral method of [Tessendorf 2001] to generate ocean wave shapes and dynamics. This is a statistical model based on experimental observations from the oceanographic literature. Specifically, we use the following formula to generate a height field representing the ocean waves over a rectangular region:

$$h(p,t) = \sum_k \tilde{h}(k,t)\exp(ik \cdot p) \tag{1}$$

where t is time and $k = (2\pi n/L_x, 2\pi m/L_z)$ is a two-dimensional vector, n,m are integers that belong to the intervals $n \in [-N/2, N/2]$, $m \in [-M/2, M/2]$, and L is a length parameter. The simulation results in a time-variant height field defined as a function $h(p,t)$ at discrete points $p = (nL_x/N, mL_z/M)$. Here, $L = (L_x, L_z)$ is a scale parameter and (M,N) is the grid resolution. More details of this method can be found in [Tessendorf 2001].

We sample this height field at different spatial resolutions to get representations of ocean waves at different scales. Basically, we need both low- and high-frequency sampling. The low-frequency grid is used to construct a displacement map that tiles the ocean surface. The high-frequency grid describes the fine details on wave surfaces through a bump map. For both displacement and bump maps, we appropriately sample above the Nyquist limit of the function $h(p,t)$ in space and time, by considering all compatible frequency components due to the discretization. These samples are played cyclically to simulate the wave dynamics during animation.

We should remark that our system can use any method that generates a ocean surface as a tileable height field. The method we currently use is the one adopted by the entertainment industry.

2.2 Reflection, Refraction, and Shadow Maps

We generate reflection / refraction maps by rendering all objects above / under the water plane to textures. In the open ocean, light is both scattered and absorbed by the volume of water. This water attenuation effect is approximated by shading effects during the refraction map generation. We use a new fog-from-depth shader. A shadow map is rendered into the alpha channel of the reflection map. Both reflection and refraction maps are used as projective texture maps during rendering.

To generate the reflection map, we first transform all objects above the water surface to its symmetric position under the ocean plane. Then we render these objects from the given viewpoint and save the result into the reflection texture map.

2.3 Final Rendering

After we obtain the reflection map, refraction map and shadow map, we can render the ocean surface using the GPU. In the vertex shader we first transform the ocean surface according to the current viewpoint. Then, we generate texture coordinates at each vertex for the bump map, reflection map and refraction map. In the pixel shader, we load the bump value. Then, we find the Fresnel term, reflection color, and refraction color for each pixel. The bump value is used in this step to perturb the texture coordinates. After computing the specular color caused by sunlight for each pixel, we finally composite all components together.

3 View-dependent Wave Geometry

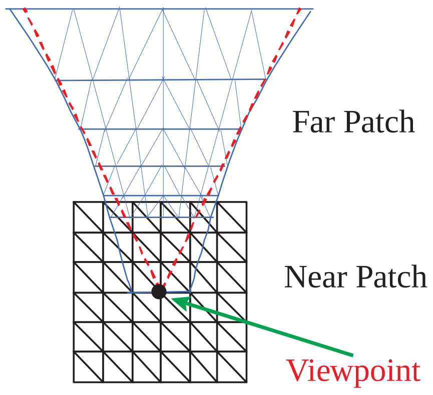
In this section we discuss the generation of view-dependent wave geometry for efficient rendering.

The height field computed by $h(p,t)$ is buit to tile seamlessly an arbitrarily large ocean plane. The purpose of the view-dependent representation of wave geometry is to compactly describe the waves in the viewing frustum. As shown in Figure 2(a), the view-dependent representation of wave geometry consists of a *near* and a *far* surface patch, which together cover an ocean surface region that is a slightly larger than the ocean surface in the current view frustum (shown as dotted red lines).

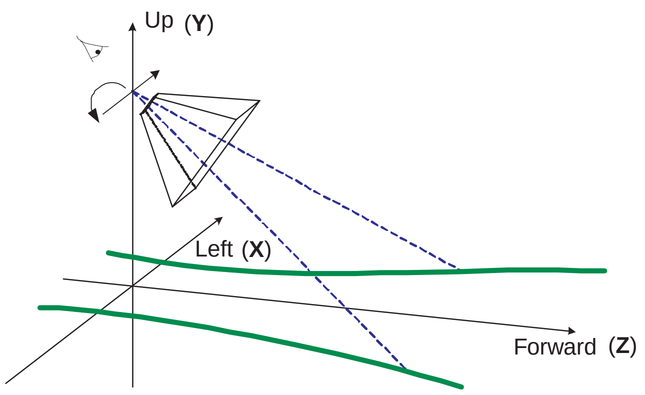
A good view-dependent wave geometry is vital to ocean rendering. Unlike terrains [Duchaineau et al. 1997], the ocean is largely planar and the far away waves are thus not occluded. We efficiently handle these waves by the far patch with a dynamic bump map. The far patch is pre-computed and can be used for any view point above the ocean surface, leading to big savings in the rendering stage.

To describe the geometric details of the ocean waves near the viewpoint, a fixed-size planar near patch is used to represent the height field around the viewpoint. The size of the near patch is large enough to cover all ocean waves with visible height variations for all viewing heights and directions. The height field of the near patch is sampled from the tiled displacement map. Note that the resolution of the near patch changes with the height of the view point. We maintain a continuous level of detail representation (LOD) for this purpose. This LOD structure is a binary tree similar to [Duchaineau et al. 1997]. This tree is generated by a procedure that determines the refinement level based on the viewpoint height.

The far patch is used to fill the visible ocean surface from the far-end of the near patch to the horizon (see Figure 2(b)). The far patch is planar and waves are represented by a bump map on the ocean plane. To avoid generating the far patch for each frame, we construct the far patch in the pre-processing stage. As shown in Figure 2(b), for a viewpoint located at the origin with the viewing direction rotating around the x-axis, the visible area on the ocean plane (at height $-H$ and for some constant field of view) is bounded by two conic curves.



(a)



(b)

Figure 2: (a) Top view of the view–dependent wave geometry. The blue wire-frame mesh represents the far patch. The black wire-frame mesh represents the near patch. (b) The conic curves (in green) bounding the far patch. The curves are defined by $y = -H$ and $a^{-2}x^2 = z^2 + 1$.

The far patch is tessellated based on the viewing distance using the two conic curves. Thus the far patch can be used for all views obtained by rotating the viewing direction around the x-axis.

When the viewpoint moves along the y-axis, we just extend the far patch to cover the new visible area on the ocean surface. If the viewing direction rotates around the y-axis, we rotate the far patch to the corresponding visible area. Thus the pre-computed far patch can be used for any view above the ocean surface. To stitch the far patch and near patch seamlessly, we force the height (y value) of the vertices on the near patch's boundary to be zero. To avoid overlapping and blending between the two patches, during rendering we process triangles of the far patch in the following way: First, triangles that are totally contained in the near patch region are culled; Then for triangles that are partly in the near patch, we move their inside vertices to the near patch boundary.

The far patch allows efficient view frustum culling, so that only triangles between the near and far planes are rendered.

4 Realistic Rendering of Ocean Waves

In this section we describe a physical lighting model of ocean waves. As in [Tessendorf 2001], we also assume that the ocean surface is "nearly perfect specular reflector". That is, we treat the ocean surface as a set of locally planar facets. Because each facet is regarded as a perfect mirror, for any view vector V, we only consider the incident rays coming from the reflection direction L and refraction direction R. The view vector V and reflection ray L have the same angle with respect to the surface normal N, while the refraction ray R follows the Snell's rule. Thus the radiance along the view direction V can be computed by:

$$C_{\text{water}} = FC_{\text{reflect}} + (1 - F)C_{\text{refract}},$$

where the F is the Fresnel term.

4.1 Fresnel Term

The Fresnel term can be computed by

$$F_\lambda = \frac{1}{2}\frac{(g-c)^2}{(g+c)^2}\left(1+\frac{(c(g+c)-1)^2}{(c(g-c)+1)^2}\right), \tag{2}$$

where $c = \cos\theta_i = L \cdot H$, $g^2 = \eta(\lambda)^2 + c^2 - 1$ and $\eta_\lambda = \frac{\eta_t(\lambda)}{\eta_i(\lambda)}$.

Here, η_t and η_i are indices of refraction of the two media (air and water). H is the normalized half vector of lighting vector L and view vector V. For the ocean surface, H is the same as the local surface normal N. Because $c = L \cdot H = L \cdot N = V \cdot N$, the Fresnel term is a function of $V \cdot N$. The Fresnel term varies quickly on the ocean surface due to normal variation of the detailed waves. Thus, the color variation across the resulting image due to Fresnel is the most important feature of ocean surface appearance.

4.2 Reflection

The reflection color can be directly traced along the reflection direction $L = 2N - V$. However, if high-dynamic range images cannot be used to represent the irradiance, we must compute the reflection C_{envir} caused by the environment, and the reflection C_{specular} caused by the light source (sunlight) separately. So the C_{reflect} is computed as

$$C_{\text{reflect}} = C_{\text{envir}} + C_{\text{specular}}.$$

4.3 Refraction

Given the view vector V, the refraction ray direction R can be computed by Snell's rule. Thus, we have: $\eta_i \sin(\theta_i) = \eta_t \sin(\theta_t)$.

The azimuth angle of the refraction vector is the same as the azimuth angle of the view vector V.

The refraction color also contains two parts: the refraction color of the object under the water Cbottom and the color of the water volume C_{wvol}. Thus, the refraction color is computed as

$$C_{\text{refract}} = C_{\text{wvol}} + C_{\text{bottom}}$$

The refracted object color is computed as:

$$C_{\text{bottom}} = C_{\text{obj}} e^{-KS_c},$$

where C_{obj} is the object color and K is the diffuse extinction coefficient of water. S_c is the distance in the ocean from the object point to the camera.

The color of the water volume is determined by several factors, such as intensity of the incident sunlight $I_s un$, extinction coefficient of water c, scattering coefficient of water b, diffuse extinction coefficient of water K, and so on. The water color can be computed by the following simplified formula:

$$C_{\text{wvol}} = \int_{S_c} b I_{\text{sun}} e^{cs} e^{-Kl} dl,$$

in which s is the length of the underwater path of the sunlight to unit l in the water.

5 Real-Time Implementation

In this section we discuss in detail our implementation for realistic real-time rendering of ocean waves.

The rendering pass draws the waves with Fresnel reflection and refraction in addition to the specular reflection caused by the sunlight. In the case of calm ocean waves simulated in our system, occlusion between waves is modeled by the displacement map. We augment the flat triangles of this displacement map with a bump map to model the detail of the ocean surface. Specifically, for the ocean surface $y = h_{\text{water}}$, we tile bump maps on it. Thus, for each position (x,y,z), its normal $(0,1,0)$ after bump mapping becomes $(du, \sqrt{1-du^2-dv^2}, dv)$, in which $f(x,z) = (du,dv)$ is the bump map value for this position.

In our system, the ocean color is computed according to the following equation:

$$C_{\text{water}} = F_{\text{above}} C_{\text{reflect}} + (1-F_{\text{above}}) C_{\text{refract}} + A_{\text{shadow}} C_{\text{specular}}. \tag{3}$$

The Fresnel term $F_{\text{above}} = F(V \cdot N)$, where N is the bump-map modified surface normal and V is the normalized view vector.

5.1 Fresnel Bump Mapping

To implement the Fresnel effect, we pre-compute the Fresnel term and store it into a texture map. [Heidrich and Seidel 1999] used a 1D texture map for the Fresnel term, which is indexed by $V \cdot N$. Unfortunately, evaluating both N and $V \cdot N$ per-pixel for a dynamic bump map will introduce too much burden for the final rendering step, making a single-pass implementation of this step unlikely. To address this issue, we store the pre-computed Fresnel term into a 2D texture map. By assuming that the surface normal points towards the positive Z direction in the local coordinate frame, the 2D Fresnel texture stores the Fresnel term for all possible view directions. For each texel (s,t), we compute the Fresnel value for the normalized local view vector $V = (s-0.5, t-0.5, \sqrt{1-(s-0.5)^2-(t-0.5)^2})$. Figure 3 shows the coordinate frame used for Fresnel texture construction and the resulting Fresnel texture.

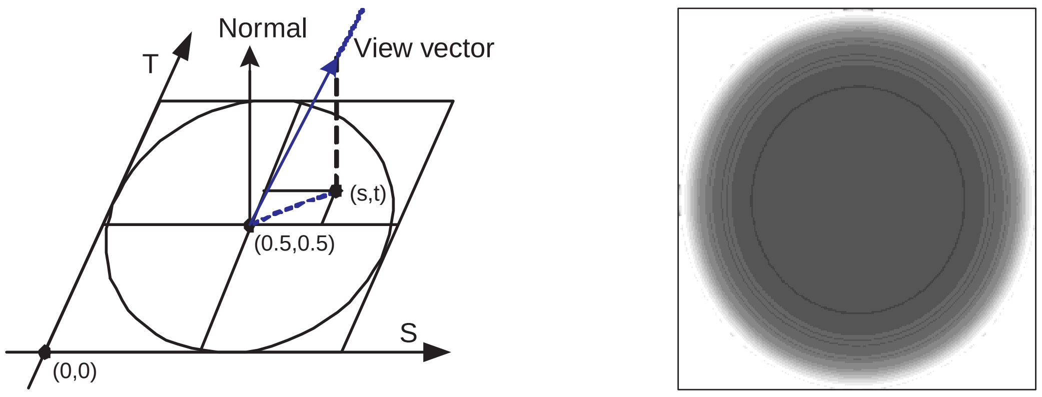


Figure 3: Fresnel texture construction.

Given a point $P = (x_p, y_p, z_p)$ on the ocean surface, we can compute the normalized view direction $V = (x_v, y_v, z_v)$ for this point. Because the normal of the flat ocean surface is towards the positive y axis, we use (x_v, z_v) as texture coordinates for the Fresnel texture map. (To get the Fresnel value for this point, we need to rotate the view vector into the local coordinate frame defined for the Fresnel texture.) When the normal at P is bumped by (du, dv), the Fresnel term changes accordingly. However, it is computationally expensive to find the Fresnel value for this bumped normal. As shown in Figure 4, instead of bumping the normal for Fresnel term computation, we bump the view vector by $(-du, -dv)$ directly. Thus, we can easily find the Fresnel term by looking up $(x_v - du, z_v - dv)$ in the Fresnel texture.

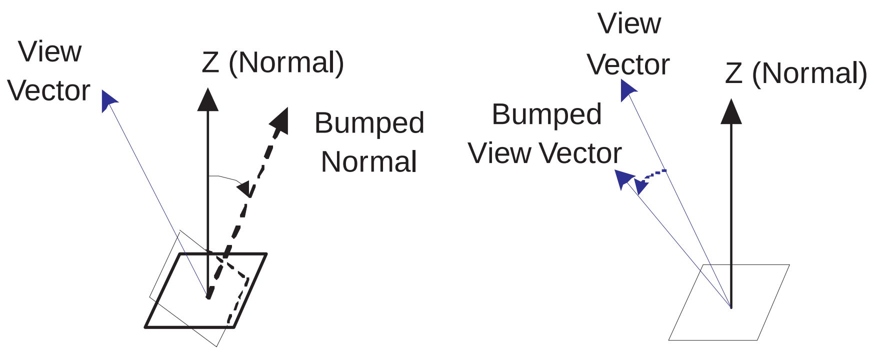


Figure 4: Fresnel texture look-up for a bump-mapped surface.

The main advantage of our technique is that it can accurately evaluate the Fresnel term with a single texture fetch command in the pixel shader of the GeForce 3 graphics card. [Schneider and Westermann 2001] approximately evaluated the Fresnel term using a formulae from [Schlick 1994]. Their approach would require about six arithmetic operations to evaluate the Fresnel term, which is a serious burden on the pixel shader since it has only a small number of commands.

5.2 Reflection and Refraction

The reflection color caused by the environment can be computed from the reflection map or environment map. We generate a reflection map for each frame. To do so, we regard the ocean surface as a flat mirror and then transform all objects above the water to their mirror position with respect to the ocean surface. After rendering all transformed objects into a texture, we can map them on the ocean surface by projective texture mapping.

For a bumped ocean surface, it is difficult to trace the bumped reflection direction to get the correct result. An alternative solution is to shift the original reflection map coordinates by the scaled bump map values (du, dv), as shown in Figure 5. The result is only correct for some rays. However, our experiments show that the result is acceptable.

As mentioned before, even for a flat ocean surface, we need to compute the refraction direction for each view vector, which is complicated. Previous methods [NVIDIA 2002] usually compute the refraction direction for each vertex and then interpolate the refraction vector at each pixel. In our implementation, the refraction effect is approximated by scaling the underwater scene along the y axis. We scale the underwater scene and then render it into the refraction map for each frame.

In the refraction map, the refraction color is approximated with the following equation:

$$C_{\text{refract}} = \alpha_{\text{fog}}(d) C_{\text{deep\_water}} + (1 - \alpha_{\text{fog}}(d)) C_{\text{obj}}, \tag{4}$$

where $\alpha_{\text{fog}}(d)$ is the exponent function of the water depth, which is used to simulate the water volume transmission effects. Similar to the reflection map, we apply the refraction map for the bumped ocean surface. We bump the texture coordinates to approximate the disturbed refraction rays.

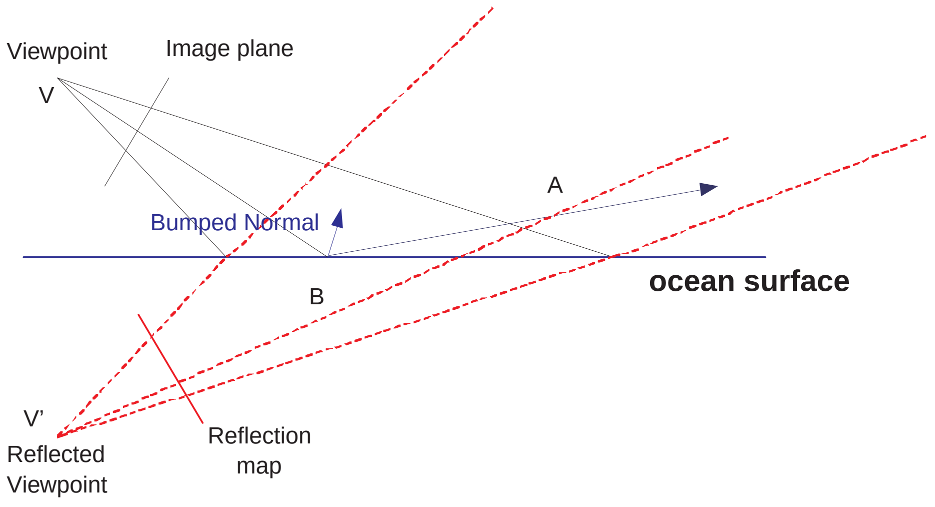


Figure 5: Reflection map generation (1D case).

5.3 Specular Computation

The specular color resulting from directional sunlight is approximated by the Phong model. The Fresnel effect for the specular reflection is ignored. To compute the specular color for each pixel, we store the normalized view vector in the RGB channel of the Fresnel texture. Similar to Fresnel computation, the normalized bumped view vector is found in the Fresnel texture and is then used to compute $(V \cdot R)^s$ for each pixel, where R is the reflected sunlight direction with respect to the default ocean surface normal $(0,1,0)$, and V is the view direction. The parameter s is specified before rendering. Note that specular color is attenuated by the shadow map.

5.4 Final Rendering Pass

Figure 6 shows the single-pass implementation of the final rendering on a GeForce 3 graphics card. We input the patch mesh vertices' 2D positions (i.e., coordinates on the ocean plane) into the vertex shader. To compute the lighting effects, we also input the viewpoint position and the sunlight direction. In the vertex shader, we first transform the mesh to the world space and then find its vertex heights by looking in the displacement map. Then the vertex's 2D position on the ocean plane is used as the texture coordinates for the bump map, reflection map, and refraction map.

In the pixel shader, for each pixel, the bump map is used to perturb the Fresnel, reflection, and refraction texture coordinates. Using the perturbed texture coordinates we can obtain the corresponding components by texture lookups. After computing the specular color resulting from the sunlight, we combine all these components together to get the final pixel color.

6 Results and Final Comments

Figure 7 shows three different views that demonstrate sky illumination, shadows, reflection, refraction and Fresnel effects. Figure 7(a) highlights the sky illumination, Figure 7(b) emphasizes reflections and shadows, Figure 7(c) reveals refraction and Fresnel effects. These images come from a 3D game and they were grabbed directly from the screen. The companion video demonstrates that, with our techniques, it is possible to render in real time ocean waves with sophisticated lighting effects including Fresnel reflection and refraction with today's GPU.

References

CABRAL, B., OLANO, M., AND NEMEC, P. 1999. Reflection space and image-based rendering. *Proceedings of SIGGRAPH*, 165–170.

DUCHAINEAU, M. A., WOLINSKY, M., SIGETI, D. E., MILLER, M. C., ALDRICH, C., AND MINEEV-WEINSTEIN, M. B. 1997. Roaming terrain: Real-time optimally adapting meshes. *Proceedings of IEEE Visualization*, 81–88.

FOURNIER, A., AND REEVES, W. T. 1986. A simple model of ocean waves. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, vol. 20, 75–84.

HEIDRICH, W., AND SEIDEL, H.-P. 1999. Realistic, hardware accelerated shading and lighting. *ACM SIGGRAPH*, 171–178.

HINSINGER, D., NEYRET, F., AND CANI, M.-P. 2002. Interactive animation of ocean waves. *ACM Symposium on Computer Animation*, 161–166.

JENSEN, L., 2001. Deep-water animation and rendering. http://www.gamasutra.com/gdce/2001/jensen/jensen_pfv.htm

NVIDIA, 2002. Reflection, refraction demo. Demo in NVidia Cg Browser 4.0, NVIDIA Cooperation.

PREMOZE, S., AND ASHIKHMIN, M. 2001. Rendering natural waters. *Computer Graphics Forum 20*, 4, 189–200.

SCHLICK, C. 1994. An inexpensive BRDF model for physically-based rendering. *Computer Graphics Forum 13*, 3, 233–246.

SCHNEIDER, J., AND WESTERMANN, R., 2001. Towards real-time visual simulation of water surfaces. Workshop on Visual Modeling and Visualization.

TESSENDORF, J., 2001. Simulating ocean waters. In SIGGRAPH course notes (course 47), ACM SIGGRAPH, http://home1.get.net/tssndrf/.

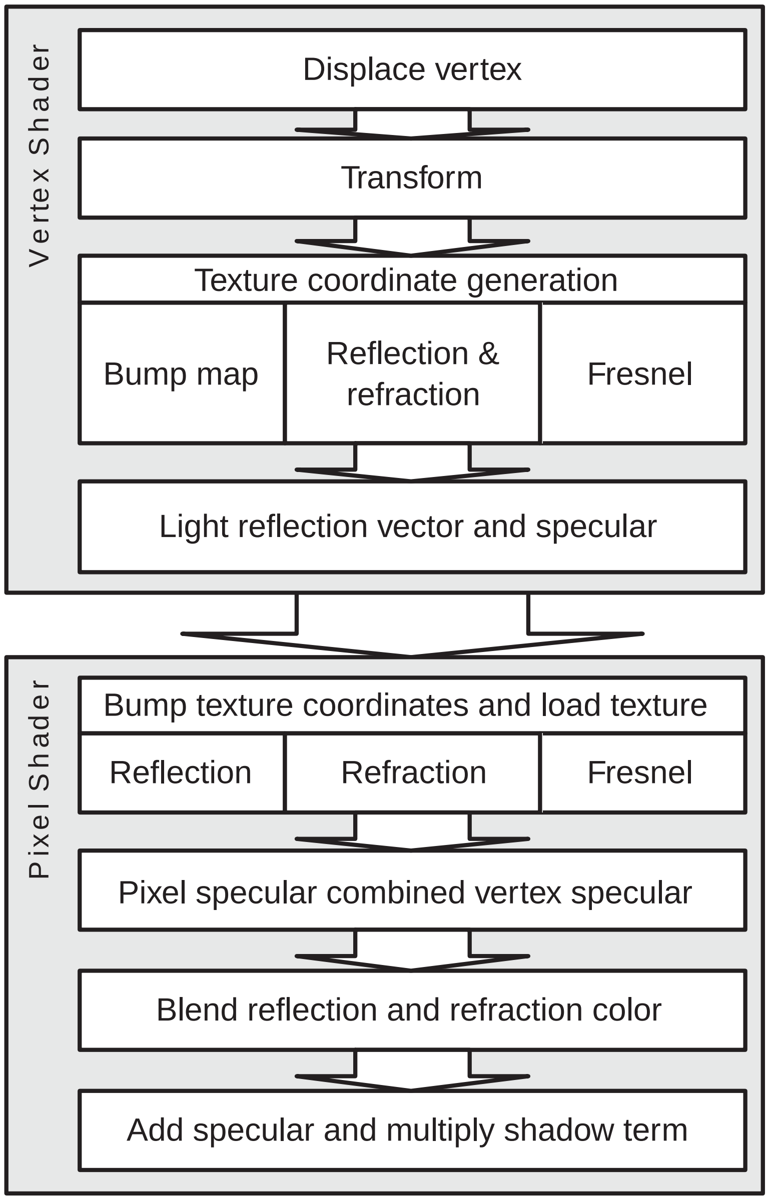
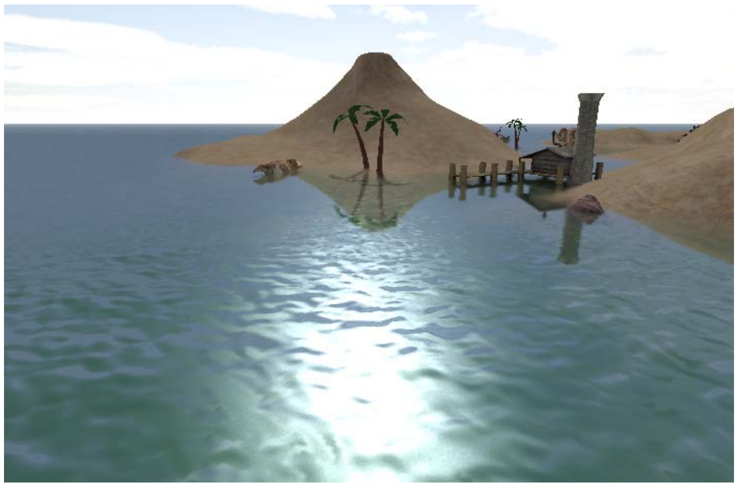
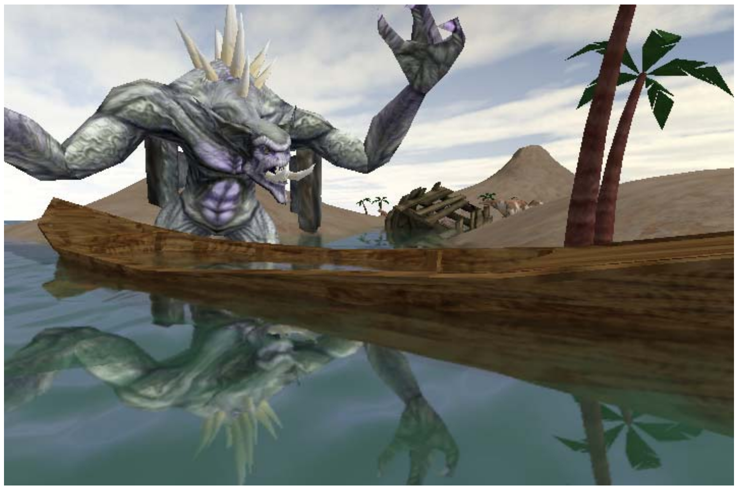

Vertex Shader
Displace vertex
Transform
Texture coordinate generation
Bump map
Reflection & refraction
Fresnel
Light reflection vector and specular
Pixel Shader
Bump texture coordinates and load texture
Reflection
Refraction
Fresnel
Pixel specular combined vertex specular
Blend reflection and refraction color
Add specular and multiply shadow term

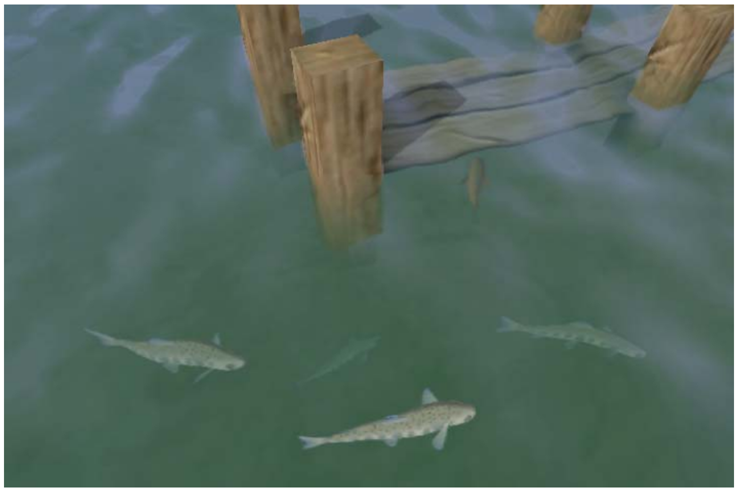
Figure 6: Final rendering in a single pass.



(a) Island



(b) Boat



(c) Fish

Figure 7: 3D game with realistic ocean rendering.