GPU-based Ocean Rendering

Yung-Feng Chiu *Chun-Fa Chang*

National Tsing Hua University
{yfchiu, chang}@ibr.cs.nthu.edu.tw

ABSTRACT

We present efficient algorithms for real-time rendering of ocean using the newest features of programmable graphics processors (GPU). It differs from previous works in three aspects: adaptive GPU-based ocean surface tessellation, sophisticated optical effects for shallow water, and spray dynamics for oscillating waves. Our tessellation scheme not only offers easier level-of-detail (LOD) control but also avoids the loading of vertex attributes from CPU to GPU at each frame. The object-space wave sampling approach allows us to produce sophisticated optical effects for shallow water and implement a state-preserving particle system for simulating spray motions interactively.

1. INTRODUCTION

Interactive rendering of oceanic scenes has become more and more important nowadays, especially for computer games. Although a number of works present ready-to-use simulations, those techniques cover only a few possible ways of water interacting with the environment. With the advance of programmable features, many algorithms previously limited to offline processing have become feasible for real-time usage. For examples, vertex shaders can be used to create the moving water surfaces and to simulate spray motions. Pixel shaders are commonly used to produce the sophisticated light effects such as sky reflection, sunlight glare, refraction, and the blue-to-greenish water color.

In this work, we present a real-time method for rendering ocean scenes with the following features:

(1) A GPU-based surface tessellation scheme, which allows viewers to interactively fly over an unbounded animated ocean, as our scheme does not require the ocean surface mesh to be predetermined. It not only offers an intuitive way of level-of-detail (LOD) control for any viewpoint but also avoids the loading of vertex attributes from CPU to GPU at each frame.
(2) Interactive spray motion simulation for oscillating waves.
(3) Object-space wave sampling approach allows us to clip those objects across the water surface accurately for sophisticated optical behaviors.

By utilizing the flexibility in programmable shader hardware, we are able to develop practical methods to render realistic-looking water surfaces at the speed of about 10 frames per second on a PC (with a 2.8GHz Pentium 4 processor, 512 MB RAM, PCI Express, and an NVIDIA GeForce 6600 GPU).

2. RELATED WORK

The classic work in ocean wave generation is well described in [3]. Tessendorf [16] presented several offline approaches including spectral wave modeling and sophisticated light effects for realistic simulation, animation, and rendering of ocean water environment.

Premože and Ashikhmin [13] introduced a method for wave generation on water surfaces using a physically-based approach and described a non-real-time light transport approach for computing complex lighting effects of ocean.

Nishita and Nakamae [11] presented a method for rendering under-water optical effects such as caustics and shafts of light. They calculated the caustics on a scan-line basis, which took several minutes to create each image.

A good overview of deep water animation and hardware-accelerated rendering is given in [7]. They presented a texture-based method for rendering foam and used a CPU-based particle system to generate spray for oscillating waves. Jeschke et al. [8] presented a procedural model for breaking waves.

The optical behavior is also well simulated in real time in [4]. However their method is limited to calm ocean waves only, due to the use of planar mirrors for local reflection and refraction. Belyaev [1] proposed a real-time rendering method for the correct colors of water and refracted water bottom surfaces.

In [4], the wave geometry is represented view-dependently as a dynamic displacement map for near-view area and a bump map for a more distant region. The closed area is embedded an extra binary tree to determine the refinement level basing on the height of view point. The more distant region is decided by two conic curves, which cover all ocean waves with visible height variation for all viewing heights and directions.

Kryachko [9] used a static radial grid, centered at the camera position, to tessellate the water surface. However

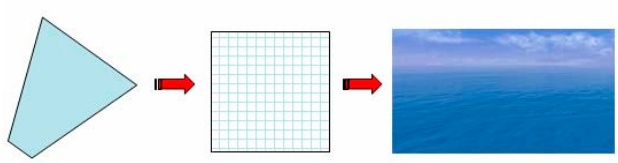


Fig. 1. The left is the visible ocean region in object space. The middle is the corresponding screen-space rectangle discretized to a regular grid which is represented as a vertex texture. The right is the corresponding region with applying waves on it.

the number of tessellated vertices that end up within the view frustum is about 25%.

Johanson [5] used a CPU-based tessellation scheme to tessellate the visible region of the water surface according to the current viewpoint. However, the computation for those positions and normals of the visible mesh burdens the CPU heavily. Besides, the loading of these vertices from CPU to GPU on-the-fly stalls the parallelism of the CPU and the GPU.

Demers [2] tessellated in eye space and mapped a regular grid to the ocean plane within the view frustum. This allows users to render only the visible geometry and tessellate more finely in the foreground than the background. However, further implementation detail is not currently available.

3. OCEAN WATER SIMULATION

3.1. Water surface representation

A pre-defined grid is a straightforward way to represent the water surface, but it suffers from a static resolution. So, we use GPU to tessellate the visible region of the water surface according to the current viewpoint. First, we determine a rectangle of the water surface that is big enough to cover the maximum height variations in waves according to the current viewpoint. Then, this minimum visible region of the water surface represented as a vertex texture is discretized in screen space to a regular grid with a set of coordinates which span across the [0,1]×[0,1] range in homogeneous coordinates in the same fashion as interpolating texture coordinates. Finally, we project these screen-space grid points to object space, and the resulting grid points provide the positions where the height field of waves is evaluated. Using this scheme, the viewer's motion induces a continuous movement of the mesh over the ocean surface, and an adequate resolution is maintained everywhere in the computed image (See Fig. 1).

3.2. Ocean waves modeling

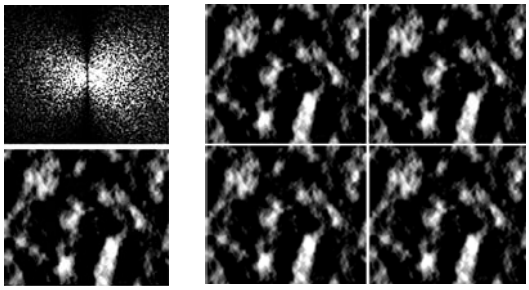


Fig. 2. The left are the random Fourier amplitudes of ocean waves and the corresponding height map. The right is a height map spanning over an unbounded water surface.

The spectral-like waves as described in [16] are suitable for the creation of a height field that spans over a large area. It is tiled in both space and time, so it can be stored to textures. These textures are height maps where each value represents the elevation for the corresponding point of the horizontal plane (See Fig. 2). More detail of GPU-based approach is in [6].

3.3. Object-space wave sampling

As described in Section 3.1, the resulting grid points of the visible water surface provide the positions where the height field of wave is evaluated. So, we subsequently sample a set of height maps at different scale to get a different spatial resolution of waves as follows:

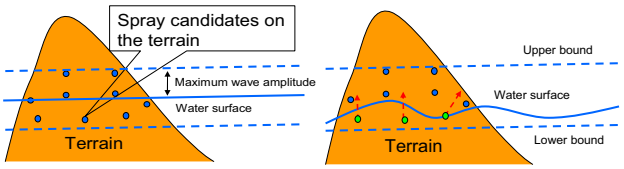
$$H_w = \sum_{i=0}^{n} tex2D(wave_i, V_{pos}.xz / scale_i) \qquad (1)$$

where *tex2D()* is an intrinsic sampling function of GPU, V_{pos} is the grid point of water surface, and $scale_i$ controls the filtering that is done during sampling.

Although the tessellation changes in each frame, we always use the object-space position from the resulting grid points to sampling the height field of the waves. In this manner, we can avoid the flickering artifact across successive frames.

3.4. Spray simulation

A state-preserving particle system [10, 14] is applied for spray simulation. First, we store the active tag, velocity, emit direction, and position in floating point textures and the index in vertex buffer for those vertices of objects (for example terrain or whales) between the lower bound and the upper bound of ocean wave (See Fig. 3). All particles are initially invisible by setting the w-component of position to zero. Then, we perform one time step to update those attributes (e.g., active tag, velocity, emit direction, and position) from previous values on GPU. This scheme can avoid uploading the data of attributes for all particles from CPU to GPU per frame, which is critical for rendering



(a) (b)

Fig. 3. (a) The blue particles are those candidates to simulate spray. (b) The green particles are set to active due to their positions are under water and emitted along the predefined directions.

the spray motion at interactive frame rate. Finally, those active particles are rendered as additive-alpha point sprites.

3.5. Optical effects

The optical properties can be decomposed into a reflection component and a refraction component, modulated by a Fresnel function as follows:

$$C_{result} = F(\theta)*C_{reflect} + (1-F(\theta))*C_{refract} \quad (2)$$

where C_{result} is the resulting color of the water surface, $C_{reflect}$ is the color coming from above-water environment along the reflection vector, $C_{refract}$ is the color coming from the underwater scene along the refraction vector, $F(\theta)$ is the Fresnel term, and θ is the angle of the viewer to the water surface.

3.5.1. Refelection

The reflection color caused by the environment can be further divided into three parts: sky reflection, sun light and local reflection.

$$C_{reflect} = C_{skylight} + C_{sunlight} + C_{localreflect} \quad (3)$$

Sky reflection is based on environment mapping which stores the cloud thickness in the alpha channel ($C_{cloud}.\alpha$) for sky rendering and Preetham's spectral radiance model [12] to approximate full spectrum daylight for various atmospheric conditions.

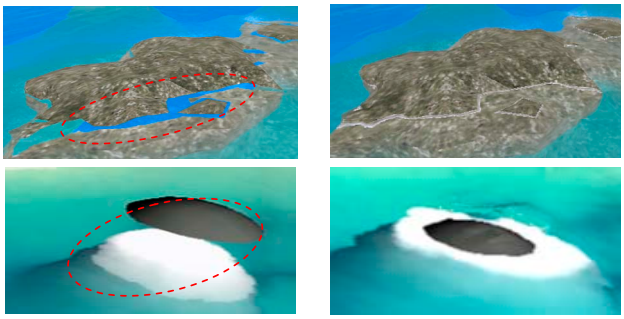
$$C_{skylight} = C_{skycolor}*(1-C_{cloud}.\alpha)+ C_{cloud}.rgb*C_{cloud}.\alpha \quad (4)$$

where $C_{skycolor}$ is Preetham's spectral radiance.

Water is an excellent specular reflector at grazing angles. We use per-pixel Phong lighting model with a large specular exponent to calculate the specular highlight term for sun light and multiply the depth value to the shininess term to adjust the shape of sun light on the water surface.

$$C_{sunlight}=C_{suncolor}*(V_{reflect}\bullet V_{sun})^{shininess*VhPos.z} \quad (5)$$

While the environment mapping is ideal for reflecting environment in distance, it is not suitable for local reflection as described in [1, 7]. The reflection color caused by the local environment can be generated as regarding the water surface to be a mirror, and all objects above the water surface are rendered into a reflection map. This method works well for clam water surface only. For oscillating water, it needs to take into account the per-pixel height of ocean waves instead of roughly using the plane y = 0 to clip those objects across the water surface accurately (See Fig. 4).



(a) (b)

Fig. 4. (a) Scenes that shows artifacts caused by using static height y=0 to clip across objects. (b) Clip those pixels that are under the water accurately by using the exact height of ocean waves.

In order to simulate high-frequency waves, the local reflection map is sampled by projective texture computations with perturbed texture coordinates from the bump map. We also divide the distortion with the post-perspective z-coordinate to make the distortion distance-dependant and to make its post-perspective space magnitude lessen with the distance.

$$C_{localrefectl} = tex2D(T_{reflmap}, V_{scr}.xy+\xi*V_{nor}.xz/V_{scr}.z) \quad (6)$$

where $T_{reflmap}$ is the local refection map. V_{nor} is the texel fetched from the bump map. V_{scr} is the screen coordinates of the water surface. ξ is the user-defined scale factor for the perturbed texture coordinate .

3.5.2. Refraction

The refraction color is decided by the scattering object color ($C_{objcolor}$) inside the water and the color of water ($C_{watcolor}$) which is influenced by scattering and absorption effects of water molecules and suspensions.

$$C_{refract}=(e^{-atten*dist})*C_{objcolor}+(1- e^{-atten*dist})*C_{watcolor} \quad (7)$$

where *dist* is the distance from the water surface to the object and *atten* is the attenuation coefficient of water. $C_{objcolor}$ is generated by the same consideration as described in Section 3.5.1 with two differences – all objects under the water surface are rendered into a refraction map and perceptually, the refraction distorts the image of all objects under water with the scaling factor 1/1.33 along the y-axis. $C_{watcolor}$ is based on the experiment performed in [15] with the depth y and can be written as:

$$C_{watcolor}.rgb = (0.30^{kd*y}, 0.73^{kd*y}, 0.63^{kd*y}) \quad (8)$$

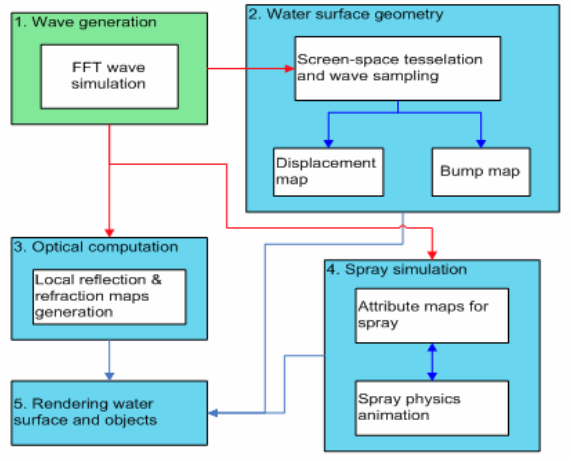


Fig. 5. System overview. The green region is executed by CPU and the others are done by GPU.

4. IMPLEMENTATION AND RESLUTS

Our water simulation system is divided into five stages (See Fig. 5): wave generation, surface tessellation, optical simulation, spray simulation, and water surface rendering. Except the first stage (wave generation), they are all implemented as shader programs in graphics hardware.

Fig. 6 demonstrates six screenshots captured from our real-time rendering system, with the performance of about 10 frames per second on a PC (with a 2.8GHz Pentium 4 processor, 512 MB RAM, PCI Express, and an NVIDIA GeForce 6600 GPU).

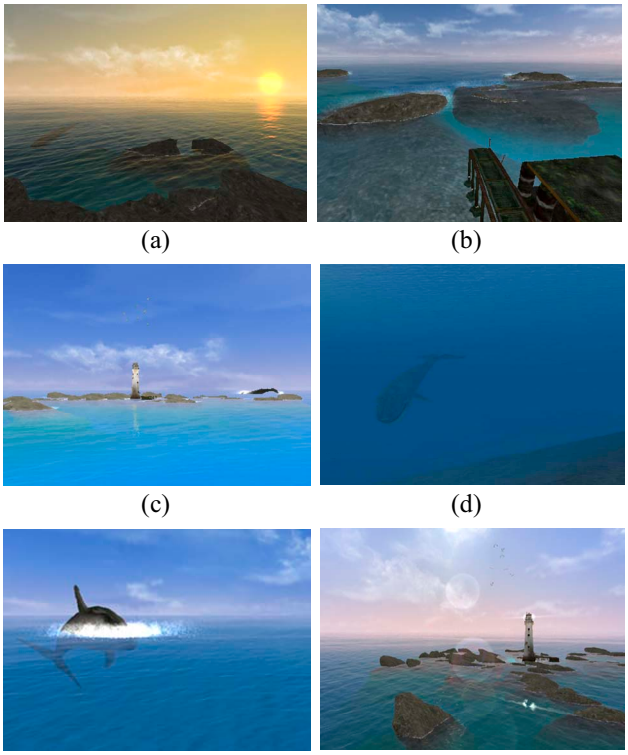


Fig. 6. The results captured from our real-time rendering system. (a) The sunset and sky light illumination. (b) The spray dynamics around rocks. (c) Local reflection and depth-dependent water color. (d) The underwater scattering effect. (e) Spray dynamics around the whale which is falling into water. (f) Refraction and Fresnel effects.

5. REFERENCES

[1] Belyaev, V., "Real-time rendering of shallow water," *GraphiCon'2004* , Conference Proceedings, 2004.

[2] Demers, J., "The Making of 'Clear Sailing," Secrets of the NVIDIA Demo Team, CEDEC 2004.

[3] Fournier, A., and Reeves, W.T., "A simple model of ocean waves," In *Computer Graphics (Proceedings of SIGGRAPH 86)*, vol.20, pp. 75-84, 1986.

[4] Hu, Y., Velho, L., Tong, X., Guo, B., and Shum, H., "Realistic, real-time rendering of ocean waves," *Computer Animation and Virtual Worlds*, Special Issue on Game Technologies, 2004.

[5] Johanson, C., "Real-time water rendering," *Master of science thesis,* Lund University, March, 2004.

[6] Jason L. Mitchell, "Real-Time Synthesis and Rendering of Ocean Water," ATI Technical Report, April 2005.

[7] Jensen, L. S. and Goliáš, R., "Deep-water animation and rendering," *Gamasutra article on realtime water*, Sept., http://www.gamasutra.com/gdce/jensen/jensen_01.htm, 2001.

[8] Jeschke, S., Birkholz, H., and Schmann, H., "A procedural model for interactive animation of breaking ocean waves," *WSCG'2003*, Plzen, February 3-7, 2003.

[9] Kryachko, Y., "Using vertex texture displacement for realistic water rendering," *GPU Gems 2*, Chapter 18, 2005.

[10] Latta, L., "Building a million particle systems," *Massive Development GmbH*, http://www.2ld.de/gdc2004/, 2004.

[11] Nishita, T. and Nakamae, E., "Method of displaying optical effects within water using accumulation-buffer," *Proc. of ACM SIGGRAPH 1994*, August, pp. 373–380, 1994.

[12] Preetham, A. J., Shirley P., and Smits, B., "A practical analytic model for daylight," *Proc. of ACM SIGGRAPH 1999*, August, pp.91–100, 1999.

[13] Premože, S. and Ashikhmin, M., "Rendering natural waters," *Computer Graphics Forum 20*, 4, pp. 189–200, 2001.

[14] Reeves, W. T., "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects," *ACM SIGGRAPH Proceedings,* 1983.

[15] Seafriends marine conservation and education centre, "Under water photography: water and light," http://www.seafriends.org.nz/phgraph/water.htm

[16] Tessendorf, J., "Simulating ocean water," *ACM SIGGRAPH 2001 course notes*, http://home1.get.net/tssndrf/, 2001