A Model For Real Time Ocean Breaking Waves Animation

Diogo Strube de Lima, Henry Braun and Soraia Raupp Musse

Graduate Programme in Computer Science - PUCRS
Av. Ipiranga, 6681 - Building 32 - Porto Alegre/RS - Brazil

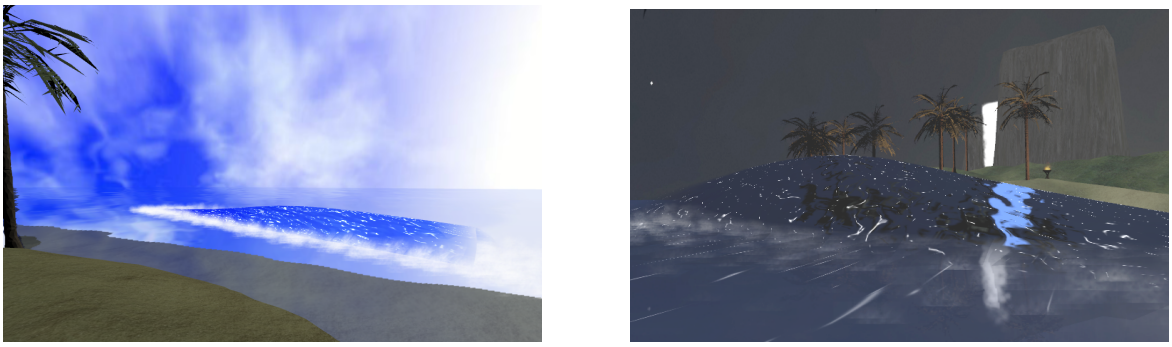


Figure 1: *Pictures from our breaking wave animation.*

Abstract

This paper presents a procedural model for ocean breaking waves in real time. Since the growth of digital games, more ocean, coast and beach scenarios have appeared, but these games rarely use breaking waves. The model proposed in this paper aims to animate waves with minimal graphic processing unit (GPU) and computer processing unit (CPU) costs. Using a few set of parameters, our procedural model generates an unique ocean-like scenario. This model could be employed in several types of games and applications. Results indicate that our real time breaking waves are visually accepted and have minimal impact on application performance.

Keywords:: Procedural modeling, Shader, Breaking Waves, Real Time.

Author's Contact:

diogo.lima@acad.pucrs.br,
henry.braun@acad.pucrs.br,
soraia.musse@pucrs.br

1 Introduction

Ocean scenarios are largely used in computer animated movies, games and simulations. Several of these applications require, or could be visually improved with the usage of, breaking waves. Unfortunately, the modeling of such ocean scenes has been a challenge in computer graphics for a long time [Jeschke S. 2003]. The main goal of our procedural model is to render several breaking waves in real time that can be easily attached to any scenario representing a minimal cost of GPU and CPU resources. Our model shows that using the GPU, simple wave breaking effects can be rendered using less than one millisecond. Furthermore, adding this effects to previously build scenarios are an easy task using maps and procedural techniques.

In our procedural model, a set of parameters are used to create a structure called wave map, which represents the waves locations, behaviors and colors in the scene. Simple mathematical functions are used to animate the wave in function of time. Shaders[1] have been used for the waves animation and rendering, allowing the CPU to be responsible for other tasks, such as a particle system for the wave splash and terrain rendering. The main contributions are a procedural model for generating breaking waves through wave maps and a high performance computing animation and rendering, providing approximately 1200 waves in real time.

The paper is organized as follows: related work are described in the next section, followed by the description of our model in Section 3. Some results are shown in Section 4 while in Section 5 some future work is mentioned as well as final considerations.

2 Related Work

Reeves and Fournier [Fournier 1986] present a simple model for the surface of the ocean. Their model is suitable for modeling and rendering waves when disturbing forces from the wind and gravity exists. In their work it is possible to determine position, direction and speed of the breaking waves. The foam existing in waves is modeled by particle systems and the waves are varied according to orbits parameters, this orbits are described in Rankine work [Rankine 1863].

Jeschke et al. [Jeschke S. 2003] describe a procedural model for an interactive animation of breaking ocean waves. The fact of being an interactive model implies that their simulation has at least a frame rate of 5 frames per second. Using a procedural approach, they are capable of a continuous surface description in time and space that enables to calculate the ocean surface without any previous time steps information. In order to create the wave life cycle, it is used a parametric function that is blended over time and it is divided in three major constants: round, breaking and collapsing.

Metaxas et al. [Metaxas D. 2004] developed an application to control breaking waves, and focus on the ideal of freedom to choose how the free surface of a liquid will look at a specific moment in time. In order to achieve this goal, object animators responsible for controlling the wave behavior are used. Their application also uses a library of waves behaviors and generates the subsequent evolution by flowing forward in time based on the 3D Navier-Stokes equation.

Another important work in ocean rendering was developed by Velho et al. [Velho 2006]. Their main goal is to render realistic ocean waves in real time. In order to achieve this goal, several techniques are employed in a system. Using a view-dependently geometry for waves, a dynamic bump map and an illumination model that includes reflection, refraction and Fresnel effects [Heidrich 1999] to create an immersive looking water. The system developed by Velho et al. is divided into two stages: preprocessing and rendering. The first stage is responsible for generating two dynamic maps and constructing the view-dependent wave geometry representation. The rendering stage takes four passes in the GPU for the reflection, refraction, shadow and Fresnel generation.

Thürey et al. [Thürey N. 2007] created a real time method that en-

[1] Set of software instructions primarily used to calculate rendering effects in the GPU with a high degree of flexibility.

hances shallow water simulations by using breaking waves. A particular characteristic of their method is the possibility of allowing interaction with rigid bodies. The created method starts by detecting the steep wave front in the height field and then spawn sheets of fluids in the detected area. Those fluids are created by a set of connected particles and results in water drops and foam effects.

In contrast to previous work, our method presents a wave map representation, allowing the breaking waves animations to be attached in real time applications, such as electronic games. Since our waves are represented by a low polygonal mesh and processed in the GPU, each wave is computed in less than one millisecond.

3 Our Model for Ocean Breaking Waves

Our model can be divided in two major steps: wave map generation and wave visualization, which includes animation and rendering. The first step in our model consists in generating the wave based on few parameters defined by the user. A procedural model processes these parameters and generates a structure we call wave map, which describes the scene including the waves behaviors, animations and colors. The second step is related to the wave animation and rendering, in this step the CPU is used for controlling the timers[2] and the particle system, as further detailed in Section 3.2. The GPU is responsible for all the waves animations, calculated in the vertex shader, and the rendering effects, processed in the pixel shader.

In this work the algorithms and the techniques used in our approach are organized in modules and are explained in Sections 3.1, 3.2 and 3.3. Figure 2 illustrates an overview of our model. Wave parameters and map generation are explained in Section 3.1, while wave animation and rendering are explained in Sections 3.2 and 3.3.

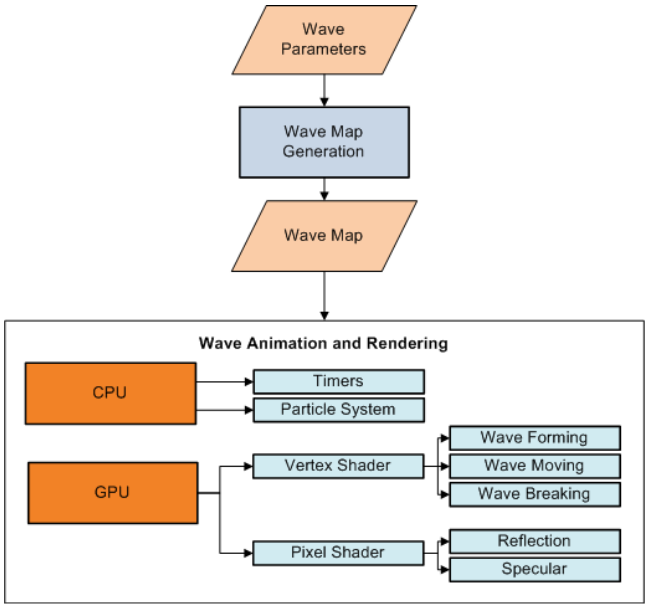


Figure 2: *Overview of our procedural model for breaking waves.*

3.1 Wave Parameters

The wave parameters are represented through intervals that defines the waves location, behavior and color. These intervals are *min* and *max* values defined by the user, that represents specific wave attributes. The parameters are used to generate a structure we called wave map and are listed bellow:

- number of waves;
- wave animation speed;
- blue color amount;
- wave width;
- wave length;
- wave height;
- wave location; and
- wave orientation.

In order to generate the waves, a procedural technique is responsible for considering the input parameters and applying a random process to specify values to the user. If necessary, the user can define as input equal values for *min* and *max* in the interval, forcing the waves to have a specific attribute.

The wave animation speed is a normalized value that defines how fast the wave geometry plane is deformed to create the breaking wave animation, while the blue color amount parameter is used only for rendering purposes, varying between 0 and 255. The wave length, width and height represents the volume that the wave can reach.

In order to create different scenes such as an island or a simple shore, the waves must have a location zone and an orientation in the wave map. Even that this approach provides limited positioning features for the wave, it presents an easy and intuitive way of configuring most of the natural waves distributions.

The location zone is given by two points forming a rectangle where the waves are randomly placed accordingly to the number of waves parameter (illustrated in Figure 4). Finally, the lasts parameters are related to the waves orientation, that can be a reference point or a reference vector. In first case all the waves are rotated in order to face the defined reference position. If a reference vector is used, all the waves will have the same orientation according to the specified vector. Figure 3 illustrates an example of a shore (using reference vector) **(a)** and an island (reference point) **(b)**.

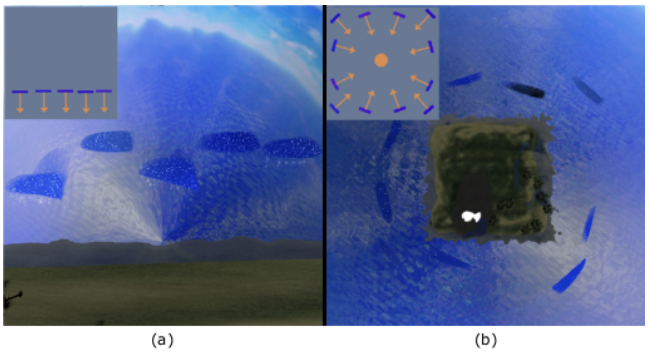


Figure 3: *Example of a shore* ***(a)*** *and an island* ***(b)*** *where waves have different locations and orientations.*

This procedural approach allows the creation of different scenes anytime that it may be required by just rearranging the parameters differently. After this step, since all the waves location are known, the procedural module is able to create particle emitters for every wave in the scene. The emitters from our particle system [Reeves 1983] use small textures as billboards that represent the wave splash created by the waves stress, detailed in Section 3.3.1.

One contribution of this work is to structure all wave parameters in a wave map. The wave map created as output of this step is an image file that describes all the waves information. This file can be used in other animations, skipping the procedural module. In the wave map file, the rectangles indicate the waves and their location coordinates in the scene. The *RGB* color channel holds the information of the speed, height and color respectively. The ocean *RGB* color channels only contains *Blue* information, in other words the pixels that represent the ocean have no speed and height information[3]. The shape of the wave rectangles represents the wave width and length. Figure 4 illustrates the described wave map.

[2] Specialized type of clock used for controlling a sequence of an event or process.

[3] In this paper we modified the *RGB* of ocean and increased the reference point size for visualization purposes, in Figure 4

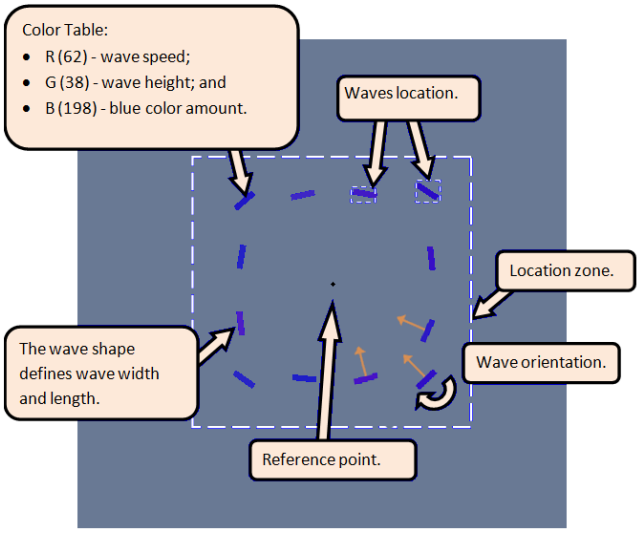


Figure 4: *Wave map output generated in the procedural model.*

In this work, for terrain generation it is used a grayscale image file that represents the *heightmap* of the terrain [Röttger et al. 1998], where as whiter is the pixel, higher the related vertex should be placed. The terrain heightmap dimensions are 256 x 256 pixels and our wave map has a bigger dimension of 512 x 512 pixels involving the terrain. Figure 5 **(a)** illustrates the terrain *heightmap*, followed by our wave map combined with the terrain *heightmap* **(b)** and the rendering result **(c)**.

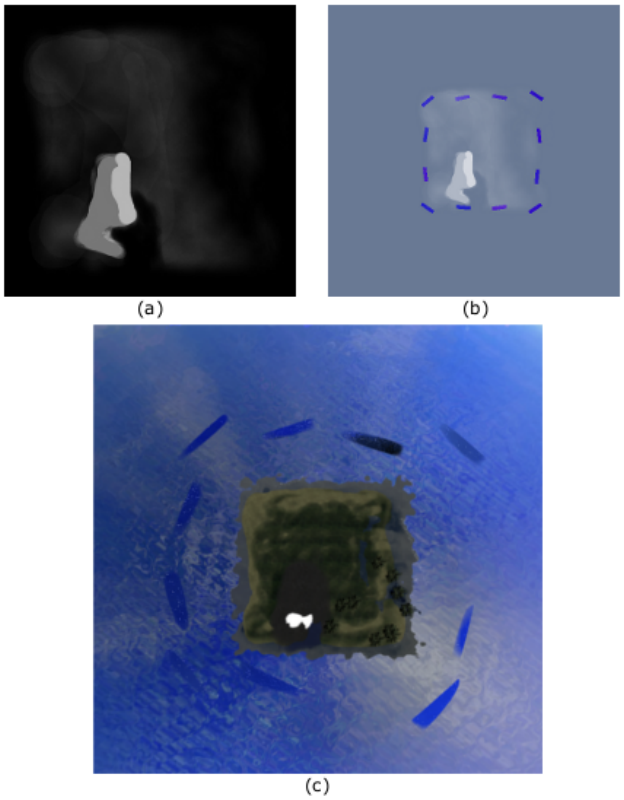


Figure 5: *Terrain heightmap* ***(a)*** *combined with our wave map* ***(b)*** *used for the scene rendering* ***(c)***.

3.2 Wave Mathematical Model

The wave behavior is processed in the GPU, with a vertex shader, using the wave map generated by the procedural model. The wave is a geometric plane that is deformed in function of time using mathematical functions and its behavior is divided in three stages: **Wave Forming**, **Wave Moving** and **Wave Breaking**. The first stage is responsible for folding the wave geometry, until it reaches the desired height. After reaching the desired height, the wave enters in the second stage and moves through a defined length accordingly a previous determined speed. After the **Wave Moving** stage, the wave geometry is rapidly deformed in the third stage, disappearing under the ocean. The wave life cycle and the wave stages are illustrated in Figure 6.

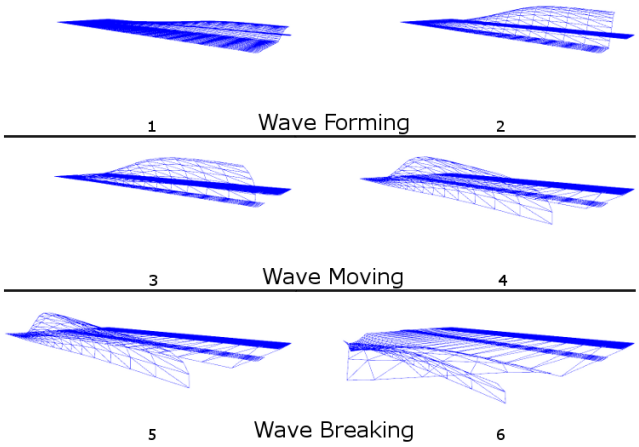


Figure 6: *Wave life cycle illustrating the three wave stages.*

The wave behavior uses vertex displacement, and the wave geometry is a rectangular plane, formed by set of 200 vertexes. Each vertex has three attributes: position, normal vector and texture coordinates. The vertex position is modified by our algorithm, making the plane geometry animate across the wave stages. The vertex normal vector is projected to the world and view space and then passed to the pixel shader, where it is used for lighting calculations.

The texture coordinates are used to identify the vertex position relative to the plane. Using such vertex position, it is possible to create effects with minimal calculations, such as the **Wave Breaking**. The usage of texture coordinates is illustrated in Figure 7, where the coordinates represents the vertex position **(a)** and the coordinates used for creating the **Wave Breaking** geometry deformation **(b)**.

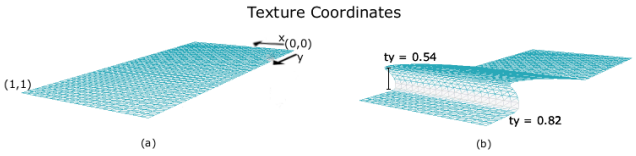


Figure 7: *Illustration of wave texture coordinates used for wave geometry deformation.*

The texture coordinates are used in the **Wave Forming** and **Wave Breaking** stages. In this work the notation tx and ty refers to the texture coordinates, used to represent the wave width and length, respectively. These coordinates are also employed for controlling the animation region in the plane, which is defined by the condition: $\{ty \in R : 0.075 < ty < 0.95\}$. This condition assures that the wave animation starts and ends under the ocean, avoiding animation artifacts.

The vertex shader has a wave life cycle timer, described as lt, which controls the activated wave stages. This timer is set to zero every time the animation begins and it is the same data used by the particle system. Each stage has its own timer, which is responsible for controlling stage animation and is set to zero according to the wave life cycle time.

Our mathematical model have four variables used by the wave stages equations: the sine based s, the cosine based c, and the vertexes relative positions dx and dy. The sine and cosine based variables are used for the vertical and horizontal vertex deformations, respectively. The dx and dy variables are used for calculating the vertexes positions relative to the plane.

The variable s, described in Equation 1, use the sine mathematical function and uses a starting time st, the wave life cycle timer lt, the wave speed sp and the texture coordinate ty:

$$s = sin(st + (lt \cdot sp \cdot (1 - ty))), \quad (1)$$

where ty ranges from 0 to 1.

The c variable, described in Equation 2, uses the cosine mathematical function and is calculated using **Wave Forming** stage timer ft and the wave speed sp:

$$c = cos(ft \cdot \frac{s}{K} - \frac{\pi}{2}), \quad (2)$$

where K is empirically defined as 5 and $\frac{\pi}{2}$ represents the relevant part of the cosine function, in our deformation model.

The vertexes relative positions variables dx and dy uses the texture coordinates tx and ty, respectively. The Equation 3 and 4 describe dx and dy respectively. Both equations use the constant P, which represents the relevant part of the wave geometry plane and is equal to 0.75:

$$dx = (1 - \|tx - P\|), \quad (3)$$

$$dy = (1 - \|ty - P\|). \quad (4)$$

The vertical deformation in the **Wave Forming** stage is defined by $fv(s)$. This function uses s and dy, combined with the wave height h and texture coordinate tx. This deformation is defined in Equation 5:

$$fv(s) = s \cdot h \cdot dt \cdot tx^2. \quad (5)$$

For obtaining the vertex horizontal displacement in the **Wave Forming** stage the function $fh(c)$ is proposed. This function uses three variables: cosine (c) and sine (s) based variables and vertex relative position dx. These variables are combined with the wave width w, wave speed sp and vertex texture coordinate ty. Function $fh(c)$ is defined in Equation 6:

$$fh(c) = c \cdot w \cdot (s \cdot dy)^2 \cdot (1 - ty) \cdot \pi. \quad (6)$$

The **Wave Moving** stage is active almost all the wave life cycle and therefore has been projected to use less processing time. The horizontal deformation uses the **Wave Moving** stage timer mt and is calculated in the moving function $mh(mt)$. The wave length l in the moving function is used to determine the maximum horizontal displacement as defined in Equation 7.

$$mh(mt) = mt \cdot l. \quad (7)$$

At last, the **Wave Breaking** stage has two functions: to animate the wave in the horizontal and vertical axis. Both functions use the **Wave Breaking** stage timer bt. The horizontal breaking function $bh(bt)$ has a constant Q that is equal to 0.02, while the vertical breaking function $bv(bt)$ has a constant G that is equal to 5. These constants are empirically defined and used to establish the different moving rate between the horizontal and vertical axis when the wave is breaking. Equations 8 and Equation 9 define the horizontal and vertical displacement functions for the **Wave Breaking**, respectively.

$$bh(bt) = bt \cdot s \cdot h \cdot dy \cdot Q, \quad (8)$$

$$bv(bt) = bt^2 \cdot s \cdot h \cdot dy \cdot G. \quad (9)$$

The vertex shader animates the wave geometry through all the described stages and equations using the timers controlled by the CPU. A pseudo-algorithm is presented in Listing 1, illustrating the vertex shader steps used for creating the wave behavior.

```
1  //Constants
2  float K = 5;
3  float P = 0.75;
4  float Q = 0.02;
5  float G = 5;
6
7  if( TCoord.y > 0.10 && TCoord.y < 0.90 )
8  {
9    float sine = sin( startTime + waveTimer * speed * (1 - TCoord.y));
10   float cosine = cos(formTimer * (speed / K) - PI/2 );
11   float dy = 1 - abs(TCoord.y - P);
12   if( WaveForming )
13   {
14      //function fv
15      float vDisp = sine * height * dy
16      pos.y += vDisp * TCoord.x * TCoord.x;
17
18      //function fh
19      float hDisp = cosine * width * (sine * dy) * (sine * dy);
20      pos.z -= hDisp * TCoord.x * (1 - TCoord.y) * 2;
21   }
22   if( WaveMoving )
23   {
24      //function mh
25      pos.z -= moveTimer * length;
26   }
27   if( WaveBreaking )
28   {
29      //functions bh and bv
30      pos.z -= breakTimer * sine * height * dy  * Q;
31      pos.y -= breakTimer * breakTimer * sine * height * dy * P;
32   }
33 }
```

Listing 1: *Pseudo-algorithm illustrating the vertex shader steps used for creating the wave behavior.*

3.3 Wave Visualization

Since our waves are merely geometric planes, it is important that the render technique provides a good look and feel of the ocean. In order to achieve this goal we use specular lightning and reflection techniques calculated in the GPU leaving only the particle system, used for the wave splash, in the CPU. Our approach uses cylindrical particle emitters combined with a timer, this timer represents the wave movement and is also passed to the GPU. The particle system is explained in Section 3.3.1 where we also explain the used rendering techniques.

3.3.1 Particle System

The wave splash is a set of particles represented through billboards [Holmberg and Wünsche 2004] that are created accordingly to a particular emitter. The emitter has a cylindrical shape. This particular shape looks better for representing the wave splash created from the wave stress. The emitter is displaced in the wave plane geometry accordingly to the wave movement timer. Figure 8 illustrates the emitter attached to the wave geometry.

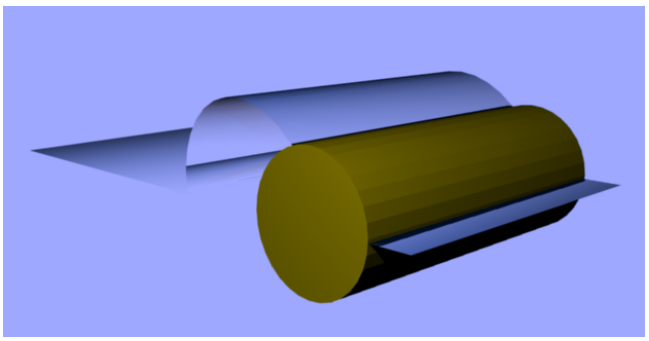


Figure 8: *Illustration of the cylindrical emitter attached to the wave geometry.*

In our animation, the timer is normalized and influences the number of particles created from the emitter. If the timer is equal to zero, none particles are created. On the other hand, as the timer increases so does the number of particles. Figure 9 illustrates the rendering result of the particle system.

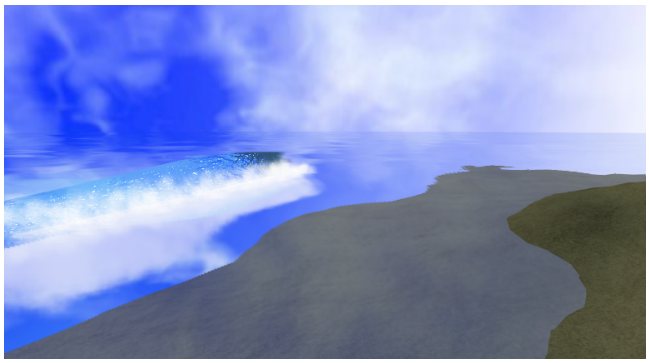


Figure 9: *Rendering result illustrating the particle system.*

3.3.2 Pixel Shader

Since the wave animation is made in the vertex shader, the pixel shader can be easily replaced by any other, for example a cartoon shading. The pixel shader in our work is responsible for hiding the wave geometry under the ocean plane and for lightning effects composed by specular, parallax mapping and reflection techniques. For hiding the part of wave geometry that is currently under the ocean, it is used the distance between the wave and the ocean. This is made by changing the pixel color, creating a smooth degrade effect between transparent and solid state. Figure 10 illustrates the wave without **(a)** and with **(b)** transparent degrade effect.

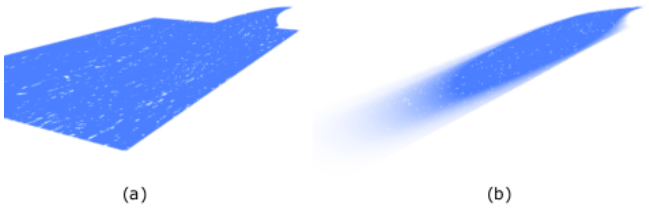


Figure 10: *Illustration without **(a)** and with **(b)** transparent degrade effect.*

The specular lightning is combined with parallax mapping technique which involves the usage of a normal map texture [Fournier 1992], illustrated in Figure 11, in order to achieve water visual results such as water shininess and wave foam.

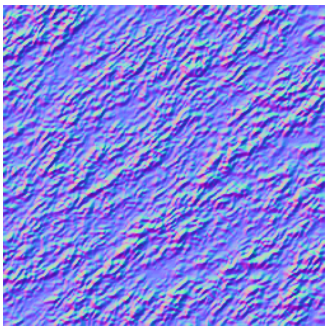


Figure 11: *Water normal map texture used for pixel shader effects.*

The reflection is made by using a previous rendered texture of the scene, and then passed to the pixel shader in order to create a noise effect by rearranging the texture coordinates. Figure 12 illustrates the pixel shader effects.

4 Results

The results are obtained using an Intel Xeon E405 equipped with NVidia Quadro FX 4800. For rendering and animation, our prototype uses Irrlicht Engine[4]. Figure 13 illustrates the wave map and the rendering result, while Figure 14 shows a different wave map and the rendering result in a different light condition.

[4] `http://irrlicht.sourceforge.net/`

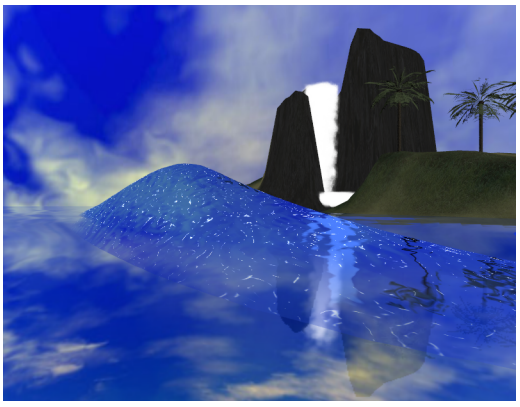


Figure 12: *Illustration of pixel shader effects.*

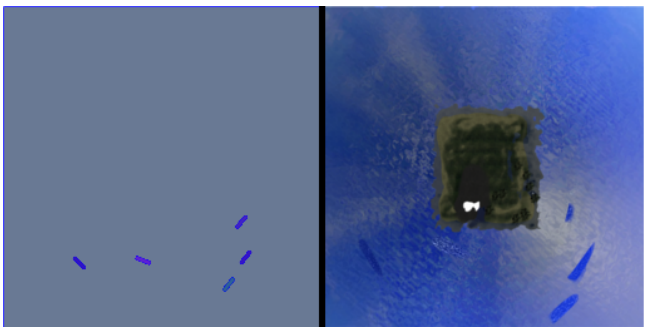


Figure 13: *Wave map and daylight render.*

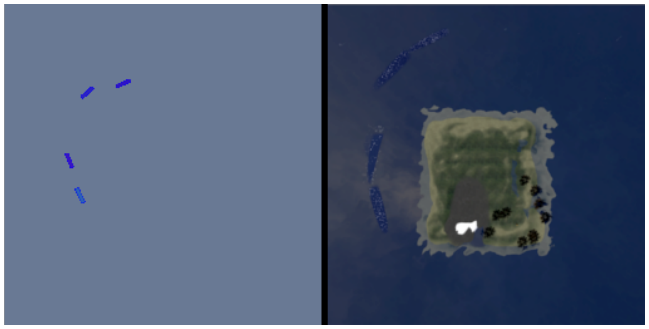


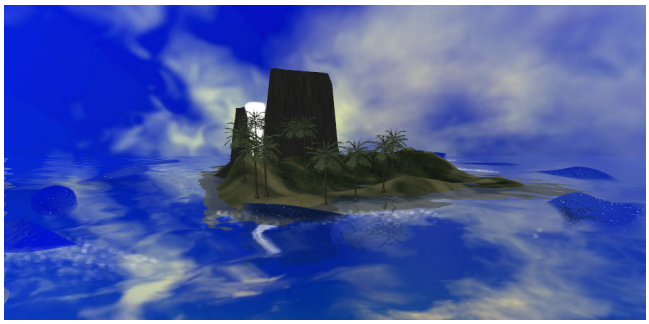
Figure 14: *Wave map and different light render.*

Figure 15 illustrates different colors for the waves using day(a) and night(b) illuminations. Figure 16 shows the waves evolving near a specific island.

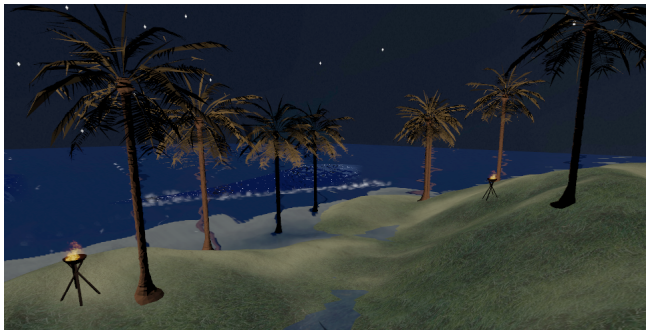
Since the vertex shader is responsible for the wave animation, when removing the particle system, it is possible to render (all waves inside the frustum) 1200 waves simultaneously at a frame rate of 30 frames per second (FPS). Figure 17 illustrates a trade off between number of waves and FPS, where we can see that the main advantage of our model is the low processing computational usage.

5 Final Considerations

This paper presented a model for animation and generation of procedural waves. The wave animation is performed in the GPU, ensuring that the CPU is capable of dealing with other tasks. In order to achieve better visual and performance results the particle system should be improved, creating a better looking and less costing wave splash animation. Another improvement could be employed in the wave geometry by replacing it for a volumetric one, this approach could avoid existing artifacts when the geometry is a plane. Another topic of research is to modify our model to interact with rigid



(a) Daylight scene.



(b) Night scene.

Figure 15: *Wave rendering under different lightning conditions.*

Figure 16: *Waves in an island environment under a night light condition.*

bodies, this approach could be used in games that has boats, surfing features or any other type of ocean contact.

References

FOURNIER, A., R.-W. T. 1986. A simple model of ocean waves. *SIGGRAPH Comput. Graph. 20*, 4, 75–84.

FOURNIER, A. 1992. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination*, 45–52.

HEIDRICH, W., S.-H.-P. 1999. Realistic, hardware-accelerated shading and lighting. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 171–178.

HOLMBERG, N., AND WÜNSCHE, B. C. 2004. Efficient modeling and rendering of turbulent water over natural terrain. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, ACM, New York, NY, USA, 15–22.

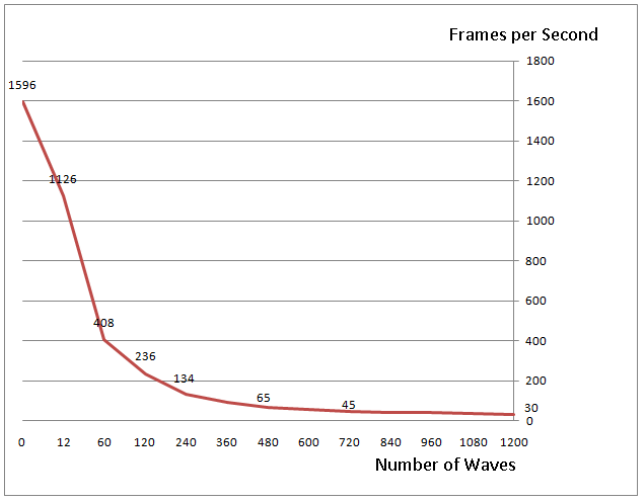


Figure 17: *Trade off between FPS and number of waves rendered.*

JESCHKE S., BIRKHOLZ H., S. H. 2003. A procedural model for interactive animation of breaking ocean waves. In *WSCG*.

METAXAS D., MIHALEF V., S. M. 2004. Animation and control of breaking waves. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 315–324.

RANKINE, W. J. W. 1863. On the exact form of waves near the surfaces of deep water. In *Philosophical Transactions of the Royal Society of London*, 127–138.

REEVES, W. T. 1983. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics 2*, 359–376.

RÖTTGER, S., H. W. S. H., (IMMD, G. D., AND ERLANGEN-NÜRNBERG, U. 1998. Real-time generation of continuous levels of detail for height fields. 315–322.

THÜREY N., MÜLLER-FISCHER M., S. S. G. M. 2007. Real-time breaking waves for shallow water simulations. In *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, 39–46.

VELHO, L., H.-Y. T. X. G. B. S. H. 2006. Realistic, real-time rendering of ocean waves: Research articles. *Comput. Animat. Virtual Worlds 17*, 1, 59–67.