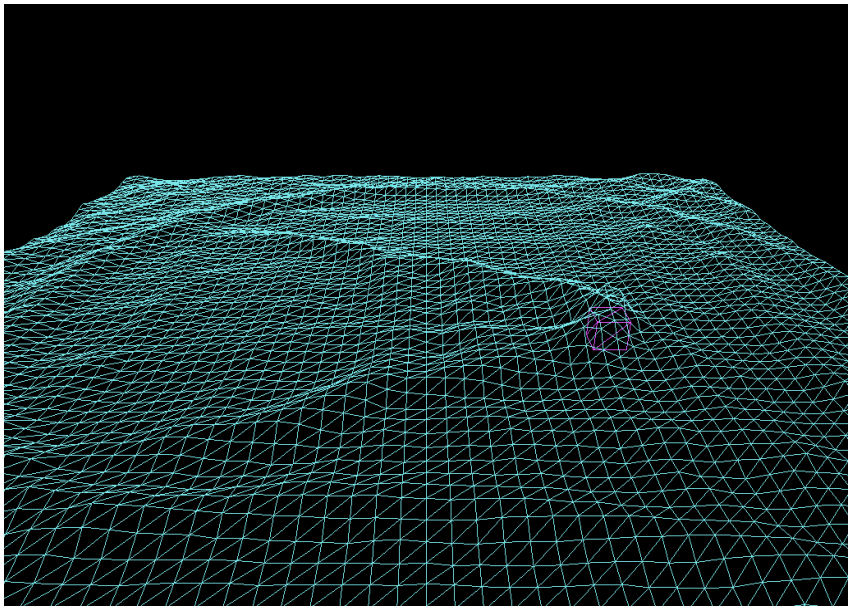
Simplified Wave Particles

Eduardo R. Poyart

Draft



Abstract

This work presents a simplified implementation of Wave Particles, described by [Yuksel et al. 2007], which produces good visual results and is very lightweight. For many applications, some of the aspects that are simulated in [Yuksel et al. 2007] can be left aside, while maintaining a satisfactory visual result. I propose a simplification of the technique by not modeling horizontal movement of the water surface and not providing interaction with objects. Further derivations of the original idea are proposed: a ring-shaped texture that models a wavefront instead of individual particles; and ripples in the surface of an ocean can be approximated by linear wavefronts.

1. A review of Wave Particles

The technique of Wave Particles is a way to perform wave simulation by means of a particle system. This is different from classical approaches such as [Tessendorf 2001], which is based on a frequency domain transform. In Wave Particles, an object moving on the surface of the water continually generates particles which move outwards radially from the object's position. These particles are then rendered on a pixel buffer, and this pixel buffer is used as a height map whose heights are applied to the water surface mesh.

Each particle essentially represents the effect of an interaction between the object and the liquid. Each particle has a value, which can be positive or negative When the object is pushing the liquid away (for example, the front part of a boat moving forward), the corresponding particles have a positive value. When the object is leaving an empty space to be filled by the liquid, as in the back side of a boat moving forward, negative particles are generated.

For each particle, a bump shaped texture is rendered on the

pixel buffer, which will eventually become a height map. Several particles close together will have their bumps overlapping, and the net effect will be of a continuous wavefront. If a positive and a negative particle overlaps, their effects can be partially or totally cancelled.

The generated particles move radially from the point of interaction at a constant speed. This poses a problem in which, if the particles get too far apart, their images in the pixel buffer will not overlap, rendering discrete bumps on the water surface. To avoid that, the particles are replicated according to a certain condition related to the distance between them.

2. Simplified Wave Particles

The original technique relies both on the vertical and horizontal components of the movement of the liquid surface being simulated. In this work, only the vertical component is simulated, in order to simplify the implementation and speed up the computations.

The basic outline of the algorithm is as follows: At every frame and for every object that moves on the surface of the liquid, a wave front is generated. This wave front consists initially of 4 particles, one for each cardinal direction in the positive and negative sides. A data structure called Wavefront stores a list of its particles, as well as a 2D vector of movement of the object. Magnitude values are assigned to the particles when they are created, and they are derived from the movement vector. The magnitude is 1 if the particle's movement direction is parallel to the object's movement direction, decreasing to 0 at a 90 degree angle with the object's movement direction, and further decreasing to -1 at a 180 degree angle. Each particle is represented by a WaveParticle data structure, which contains its position and direction of movement, as well as its magnitude.

3. Split

The particles start moving radially outwards from their point of origin. Whenever the distance between two particles becomes sufficiently large, more particles have to be generated. Let's call this operation “split”. If split is not performed, the bumps generated by each one on the height map will stop overlapping and the impression of continuity will be destroyed. Instead of directly computing the distance between particles, we can observe that this distance increases of a fixed amount whenever the distance from a particle to its point of creation is doubled. Furthermore, if the particle speed is constant, the distance from a particle to its point of creation is directly proportional to the elapsed time since the creation of the wavefront.

Therefore, a timer can be used in order to determine when to split particles. The elapsed time since the creation of the wavefront, *te*, is a real number and it's also stored at the Wavefront structure. By passing the elapsed time since the last frame to the Wavefront method that updates particles for the current frame, we can easily accumulate times with just an addition. Whenever this time reaches a predetermined split time, *ts*, let's say the distance between two neighboring particles is *d*. At this time, the split operation is performed. New particles are positioned between the original ones and the value of *ts* is doubled. This way, when the next split operation happens, the particles will again be at a distance *d* between each other.

When a split is performed, the new particles are generated at the appropriate position: not only between the original particles, but on top of the circle centered at the point of creation and on top of which all the other particles reside. Its magnitude is calculated according to the same rules as the initial particles, using the object's original movement vector.

4. Render to the height map

Once the particle's positions are updated and the particles are split (if needed), the particles are then rendered to a texture that will be used to compute a height map. In my experimental code I used OpenGL as a graphics library. OpenGL in its basic form can't perform subtraction operations in the render target, so two render targets were used: one for positive and one for negative values. A color of zero on either one of the render targets represents a magnitude of zero on the height map – a height that is halfway between the maximum and minimum heights. A color with maximum value on the positive render target represents the maximum possible height, and a color with maximum value on the negative render target represents the minimum possible height. Particles with positive magnitude are rendered to the positive render target, and particles with negative magnitude are rendered on the negative render target with their magnitude inverted.

A texture must be used to render each particle on the render targets. This texture represents the bump caused by the particle, and it will affect several pixels on the render target. Experimental results pointed to a 16x16 pixels texture with alpha, in which all pixels are white and are modulated by the varying alpha value.

The alpha value effectively determines the shape of the bump. The bump texture is pre-computed procedurally. Also from experiments, it was determined that a good shape for the bump is described by the equation:

$$\alpha_{i,j} = e^{-k(|D_{i,j}|)^2}$$

where α is the alpha value of a texel at (i, j), D is the distance between the pixel at (i, j) and the center of the texture, and k is an empirically-determined constant. This will result on a shape which has the highest value at the center, decaying exponentially while the distance to the center increases. k has to be obtained so that the edges of the texture have as small a value as possible, so that there is no visual discontinuity when the particle is rendered.

If too many particles are rendered on top of each other, for example by simulating a boat moving forward at the same approximate speed as the particles, the first experiments resulted on values that were too high, either saturating the maximum height value, which produced a flat top on the wave front, or if the scale of values was reduced, a very high peak was observed right in front of the moving boat.

This was resolved by modulating the height map magnitude by a function *f(m)*, where *m* is the original magnitude. This is done after all particles are rendered and before the render target is converted to a height map. The function used was:

f(m) = atan(c m) / c

where *atan* is the arc tangent and the variable *c* (for "compression" factor) is a limiting factor that determined how much will *atan* influence the result. For values of *c* close to 0, *f(m)* will be approximately equal to *m*. For higher values of *c*, *f(m)* will begin to have a compression at the highest values of *m*. The appropriate value of *c* was determined experimentally.

5. Conclusion

With careful selection of the exponential function that shapes the bump texture and the limiting function applied to the height map post-particle-render, a good visual result was obtained without the need to model horizontal movement of the liquid surface. A much simpler height map was used: just an array of scalars that come directly from the particle render targets. There was no need to implement an extended height map as described by [Yuksel et al. 2007].

Further work is planned to improve the algorithm:

- Ring-shaped texture

A ring-shaped texture can be used to render a whole wavefront at once into the render target. In some constrained cases, this can greatly reduce rendering demand on the pixel buffer. If there is no need to model reflections of waves, a ring-shaped texture can be directly applied. If there are reflections but they only happen in an enclosed convex surrounding, such as a rectangular pool or another convex-shaped pool, a ring-shaped texture can also be used. When a reflection happens, another ring-shaped texture can be drawn, with a center outside the pool, in a point which is the reflection of the original center with relation to the reflective surface. This way, one wave front, which begins as one ring-shaped texture, can end up being rendered as multiple ring-shaped textures after reflections happen.

- Linear-shaped texture

Likewise, in order to model waves of infinite origin, for example, in the middle of the ocean, there is no need to use several particles arranged in a linear fashion. One texture with a linear bump shape can be used. A set of these, in parallel and moving in a certain direction, can be combined with other similar sets moving in different directions, and the result is an approximation of the background movement of water on the ocean, and a local Wave Particle simulation of a boat, for example, can happen on top of it.

6. Bibliography

Yuksel et al. 2007. Wave Particles. In *Proc. Of SIGGRAPH '07.*

Tessendorf, J. 2001. Simulating Ocean Water. In Simulating Nature: Realistin and Interactive Techniques. ACM SIGGRAPH '01 Course #47 Notes.