Real-time Water Simulation with Wave Particles on the GPU

Daniel Mikeš*
Supervised by: Jiří Bittner

Department of Computer Graphics and Interaction
Czech Technical University in Prague, Czech Republic

Abstract

When rendering large bodies of water in real-time an efficient method is required to model water waves. This article describes a method for real-time interactive generation of such waves. We use the wave particle method to describe wave propagation in a fluid medium. The method allows to simulate interactions of water with general shaped rigid bodies in real-time. We present a GPU implementation of the method and show results in scenarios such as open ocean waters or pools with water boundaries.

Keywords: Wave particle, GPU algorithm, interactive, realtime simulation, waves, water rendering

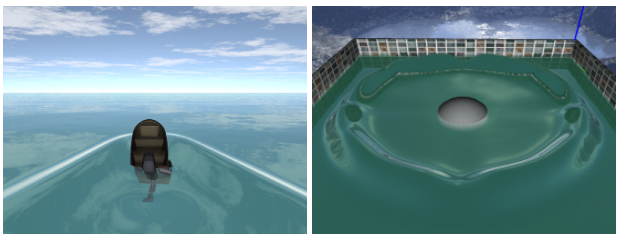

Figure 1: Real-time water simulation running on the GPU. Open ocean scenario with boat interaction (left, 290 FPS), Pool scenario with falling object (right, 270 FPS).

1 Introduction

In real-time applications such as video games it is crucial that all the computations are fast enough to be computed in a plausible frame rate, so both the rendering and the simulation stages should be fast. In general lots of computationally complex tasks can be measured or pre-computed and then used later on. On the other hand user interaction cannot be simply predicted so the computation must run on-line. For these reasons we usually need to settle for approximate solutions meaning the rendering and the simulation step is not necessarily physically correct but offers reasonable results.

Animation of large bodies of water such as lakes or oceans is important part of computer graphics and it is still an open challenge since 3D volumetric simulations are too computationally complex for mentioned scenarios.

In this article, we focus on a real-time water simulation of large bodies of water with local surface waves. We address height field representation to describe the water surface. In our simulation we use *wave particles* as a spatial information about the water surface deformation. The simulation utilizes the graphic hardware to efficiently distribute the wave particles onto the water plane according to the motion of the rigid body in the water medium.

This articles is based on the work of Yuksel et al. [12]. They presented the original wave particles method which is briefly described in section 3. In contrast to their work, we address using wave particles specifically on the GPU.

2 Related Work

Raveendran et al. [10] proposed a method for preserving uniform particle density in SPH. Onderik et al. [8] presented a SPH method improvement with small scale details such as splashes and foam. A method which uses Eulerian approach to simulate 3D water volume with a grid cell reduction was described by Irving et al. [7].

Chen et al. [2] used height field representation with spatial domain waves using shaders and bump mapping to create small ripples on the water surface.

Chou et al. [3] described a simple method for ocean simulation with one-way interaction between the water surface and rigid bodies.

Galin et al. [5] presented a real-time interactive water simulation method. They address special type of waves created by the engine of the boat, which also creates foam on the water surface. Therefore this method is usable only for specific type of rigid bodies. In contrast to the wave particle method they use a 2D grid to represent the waves instead of particles, therefore the performance is dependent only on the grid size and independent of the number of wave fronts. They are also able to handle the diffraction effect but the overall performance is lower, compared to the wave particle method.

*dm.mikes@gmail.com

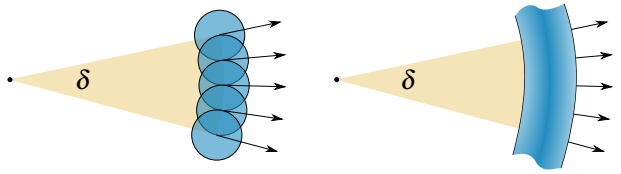

Figure 2: Wave front generation from particles. Source particles (left) and generated wave front (right) from the top view. Dispersion angle δ of the parent particle is depicted yellow.

3 Wave Particles

The wave particle method uses a particle system for representing surface deviation. Position $\mathbf{x}$ of each particle is used for localizing the deviation function $d_v(\mathbf{x},t)$ and they are totally independent of each other. Unlike Lagrangian methods wave particles move in a plane which is coplanar with the water surface. The wave particles do not represent elements of the water mass, but only a deformation on water surface.

Set of local deviation function is synthesized to the global deviation function

$$D_v(\mathbf{x},t) = y_0 + \sum_{i \in P} d_{v_i}(\mathbf{x},t)\,, \tag{1}$$

where $\mathbf{x}$ is the position on the water surface, t is the time, $d_{v_i}(\mathbf{x},t)$ is the local vertical deviation function of the i-th particle, y_0 is water base level, and P is a set of all particles.

The local deviation function can be expressed as

$$d_{v_i}(\mathbf{x},t) = \frac{A_i}{2}\left(\cos\left(\frac{\pi|\mathbf{x}-\mathbf{x}_i(t)|}{r_i}\right)+1\right)\Pi\left(\frac{|\mathbf{x}-\mathbf{x}_i(t)|}{2r_i}\right), \tag{2}$$

where A_i is the amplitude of i-th particle, r_i is the wave particle radius, $\mathbf{x}$ represents a point of the water surface, $\mathbf{x}_i(t)$ is a the position of the i-th wave particle in time t and Π is a box function, which limits cosine function over a finite region in 3D domain.

$$\Pi(x) = \begin{cases} 1, & -\frac{1}{2} \le x \le \frac{1}{2} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

3.1 Longitudinal Waves

The motion of the water surface is not limited only to the vertical deviation. In reality water particles propagate in circles which creates sharper peaks on the surface waves as shown in figure 3.

$$d_{h_i}(\mathbf{x},t) = d_{v_i}(\mathbf{x},t)\left(-\mathbf{v}_i\, sin\left(\frac{\pi\mathbf{u}}{r_i}\right)\Pi\left(\frac{u}{2r_i}\right)\right), \tag{4}$$

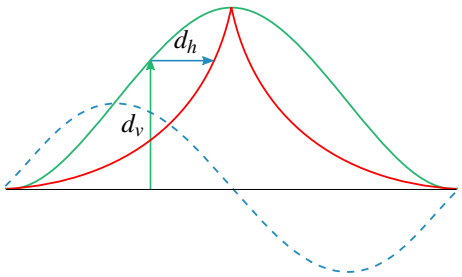

Figure 3: The horizontal (d_h) and vertical (d_v) deviation function. Original wave (green), the final wave (red), and the vertical deviation (dashed blue).

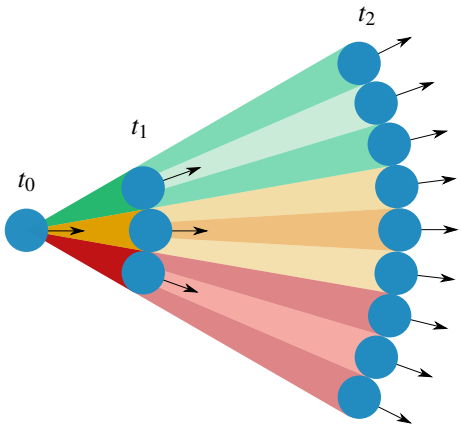

Figure 4: Dispersion angle partitioning after particle subdivision operation at different time steps (t_0, t_1, and t_2). Dispersion angle of each particle (blue circle with an identifier) is marked by different colour to enhance clarity.

where $\mathbf{u} = |\mathbf{x}-\mathbf{x}_i(t)|$, the propagation direction $\mathbf{v}_i$ and Π is the box function. This model can introduce undesirable self intersections if $r_i A > 1$. The problem is addressed by a new parameter which affects the strength of longitudinal wave.

3.2 Particle Subdivision

An expanding wave front arises when a wave is created in one point and distributes further in all directions in a 3D domain as shown in figure 2. The local deviation function does not allow changing the size of the wave particle in time. This causes that wave particles are getting further from each other while the wave front propagates. We need to cover the whole size of propagating wave front by placing one particle next to each other, so that even with a constant particle size the wave front is continuous. This is achieved by particle subdivision routine shown in figure 4.

If the distance between two neighbouring particles is larger then a defined threshold, a new particle is created on each side of the *parent* wave particle. The amplitude of the parent particle is distributed to the *child* particles in order to conserve energy.

Since the particle velocity is constant we can compute in advance at which time the distance between neighbouring particles will be beyond this threshold:

$$w_t = w_0 + \delta \mathbf{v}|t - t_0|, \tag{5}$$

where δ is the dispersion angle, w_0 is the distance between neighbouring particles in the current time t_0, and $\mathbf{v}$ is the particle velocity.

The threshold is set proportionally to the particle radius r_i and it ensures that adjacent particles will never be further from each other than the threshold parameter.

3.3 Wave Particle Properties

Besides the actual position $\mathbf{x}$, each wave particle stores the propagation angle α, dispersion angle δ, origin $\mathbf{o}$ and the amplitude A.

The propagation angle represents the wave particle direction in the 2D plane, dispersion angle δ is introduced to describe a spatial range in which new particles appear after subdivision process. The wave particle origin is the position of the particle at time $t = 0$ and it is fixed as the wave particle propagates. Amplitude represent the energy of the wave particle. Particles with low amplitude have also low contribution to the deviation function. In some scenarios it is also useful to model waves with negative amplitudes.

3.4 Creating new Wave Particles

After particle is subdivided two new particles are added to the system. Important property of the convincing physical simulation is the energy conservation criterion. Therefore the amplitude is evenly distributed to the newly created particles. The dispersion angle also changes because each particle now describes one third of the original range. Children particles are placed in the same distance r_δ from the origin. Descendants also inherit the origin of the parent particle.

3.5 Water Boundary

Scenarios such as pools require a model for reflecting incoming wave fronts off of the water boundary. The boundary represents the container which holds the simulated water.

The distance from the particle to the origin r_δ is important for the particle subdivision since it tells us when subdivision occurs. To handle particle reflection, the origin has to be mirrored over the boundary in order to persist the subdivision criterion. Since the distance between adjacent particles w does not change during the reflection, the dispersion angle may change owing to the curvature of the boundary as shown in figure 5.

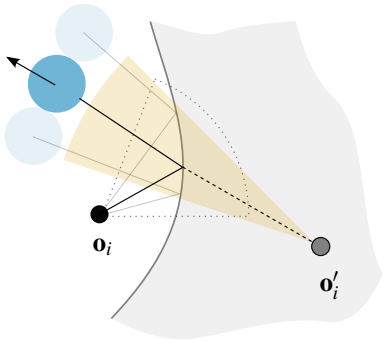

Figure 5: Wave particle (blue circle) reflection off of a curved boundary with origin $\mathbf{o}_i$. The grey area represents the boundary with the new mirrored origin $\mathbf{o}'_i$. The dotted sector represents the dispersion angle before reflection and the yellow sector is the dispersion angle after reflection.

$$w' = w \tag{6}$$

$$\delta' = \delta \frac{r_\delta}{r'_\delta}, \tag{7}$$

where δ is the dispersion angle immediately before reflection and δ' is the dispersion angle immediately after reflection; the same notation is used for other symbols. The curvature influences δ' indirectly via the origin $\mathbf{o}'$.

4 Wave Particles on the GPU

In our particle system it is essential to preserve the data on GPU memory without unnecessary data transfers from main memory to GPU buffer and vice versa. We use *attribute data* with *point* geometry to represent particles in OpenGL.

OpenGL Transform Feedback Buffer (TFB) allows us to capture the output of vertex or geometry shader inside the GPU memory. Location of TFB in the OpenGL pipeline is shown in figure 6. In each draw step the GPU fetches vertices[1] and pushes them in the vertex shader, where attribute properties can be modified or simply passed further in the pipeline. After that, the points can be stored in the TFB meaning that the data are persistent in one draw step.

The actual buffer where the primitives are stored can have different types. We use Vertex Buffer Object (VBO) as the destination of Transform Feedback operation because we reuse captured vertices in the next frame.

Therefore, we use two Transform Feedback Buffers and we chain them together in a way that output of the first buffer is the input of the second buffer as shown in figure 7. The TFBs are connected in the other way respectively. Two buffers are used due to the fact that OpenGL does

[1] Point is a 1D primitive and can be represented by one vertex. Therefore, terms point, vertex and particle are interchangeable in this context.

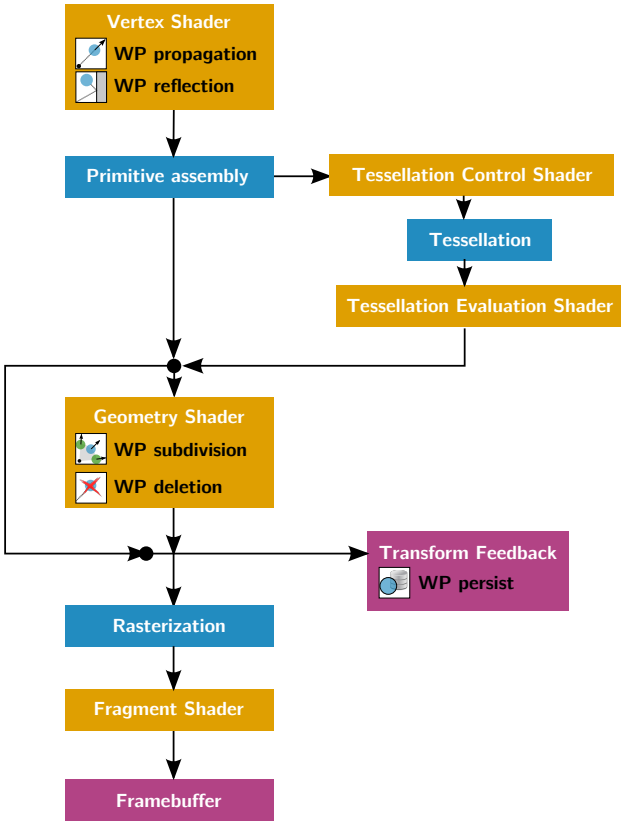

Figure 6: Simplified OpenGL pipeline. Yellow coloured boxes represents the programmable stages, blue are the fixed stages, and violet represents output buffers.

not allow reading from and writing into the same Vertex Buffer Object.

Wave particle propagation is done on the vertex shader. Only limitation in the vertex shader stage is that there is only one vertex in the input and one vertex on the output for one shader invocation. That means we cannot use vertex shaders for particle subdivision. For this purpose we can use tesselation shader or geometry shader, which is able to emit new vertices. Newly created vertices are then also stored in TFB. Figure 6 also shows which of the main tasks are performed in current stage of the OpenGL pipeline.

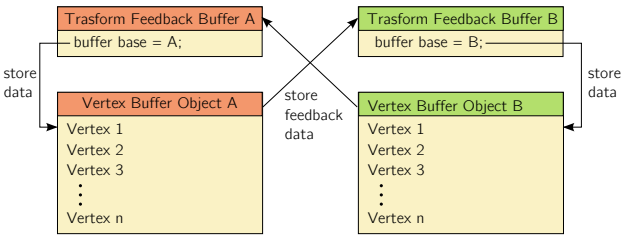

Figure 7: Illustration of Transform feedback buffer swapping. The output of first TFB is used as an input of the second TFB.

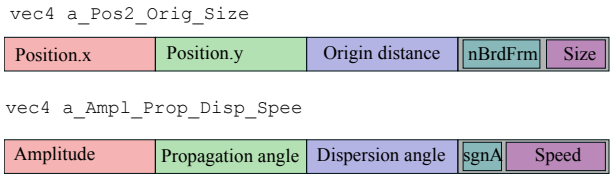

Figure 8: Wave particle structure encoded for the use on the GPU. Each field represents one floating point number.

4.1 Particle Data Structure

In order to avoid large memory consumption, we need to efficiently represent vertex attribute data.

Vertex attributes are internally packed and aligned into a multiple of `vec4`[2] in OpenGL[1]. This means that it is beneficial to use `vec4` data type and fit all necessary information to it as few member variables as possible.

Figure 8 shows the GPU packing of wave particle structure.[3] The wave particle structure is packed into 32 bytes.

Propagation routine The entry point for the simulation process is in the vertex shader, where particles are properly moved according to the propagation angle. If the particle should be subdivided or discarded, respective flag is set and the particle is passed further into the pipeline.

Subdivision and deletion routine Vertices are passed to the geometry shader, where the vertex can be either discarded or emitted. The particle is discarded if the amplitude is lower than a defined threshold and the influence of the wave particle is neglectable.

Particle generation routine Particle generation is the process of creating new particles based on the object to water interaction. The result of the interaction step is stored in the wave particle distribution texture. Wave particle distribution texture obtains information about spatial distribution of direct and indirect wave effect which is converted into particles in this step. Propagation direction is also part of the texture. The process of obtaining distribution texture is described further.

Each texel represents a potential wave particle. Therefore, we create a vertex buffer and fill it with vertices organized into 2D grid with the same resolution as the texture. Consequently we set the wave particles properties to the vertices from the texels and we convert the wave effect to the amplitude. Particles with non-zero amplitude are emitted and eventually captured by the TFB.

Wave particle reflection For performance purposes we represent boundaries as a texture. Normal vector of the boundary is encoded into each texel in order to compute

[2]GLSL representation of 4-dimensional 32-bit floating point number.

[3]nBrdFrames refers to number of consecutive frames behind border and is used for error correction in wave particle reflection routine.

the reflection. Discrete step collision detection can produce errors when the object is moving too fast and the object passes through the boundary in one frame. We adjust the texture mapping with respect to the particle speed to handle these situations.

4.2 Particle Filtering

Particles are rendered as circles with the radius equal to particle size. The final deviation of the water surface can be obtained by rendering all the particles with additive blending from a top orthographic view similarly to the texture splatting. Instead we use smaller points to represent the information about the particle presence and render them into texture. The wave particle render texture is filtered and the contribution of each wave particle in a local distance is accumulated. This is similar to the texture gathering process.

In this step wave particle deviation function is applied. The contribution of each wave particle in the filtering step is weighted by the deviation function in equation 2. Note that the function can be converted into separable filter. This means we can perform 1D filtering process consecutively for each axis and compose the final result.

The filter function can be denoted as

$$d_h^X(p) = \frac{1}{2}\left(cos\left(\frac{\pi p}{r}\right)+1\right), \tag{8}$$

$$d_h^Y(p) = \frac{1}{2}\left(cos\left(\frac{\pi p}{r}\right)+1\right), \tag{9}$$

where $d_h^X(x)$ is a X-axis horizontal deviation filter function, r represents the radius (kernel size), and $p = [-r,r]$ is the distance of a pixel to the kernel centre. The same notation is valid for Y.

4.3 Water to Object Interaction

We address four types of forces, all of which have similar implementation details: buoyancy, drag, lift and collision force.

Common feature of these forces is that they are computed for each face rendered from a top view orthographic camera into a texture. Blending must be turned on so that the information from some faces is not overridden. In order to efficiently sum the texture on the GPU and transfer only the result, we implement parallel reduction.

Buoyancy force In order to compute the buoyancy force we need to know the volume of the object's submerged part. The volume is obtained as the depth difference of the front and back faces of the rigid body. More specifically, it is obtained by an orthographic projection with blending and summed together.

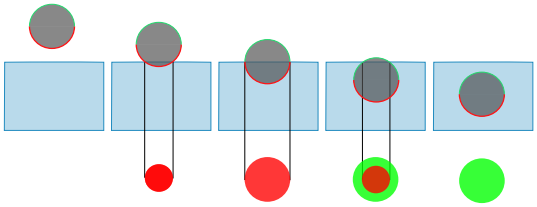

Figure 9: Upper image row represents a water tank seen from side view. We show different positions of a sphere relatively to the water level. Note that the sphere is green on the top (positive z-coordinate of the normal) and red on the bottom (negative normal). The bottom row shows the same object seen from a top view without the part which is above the water level.

Drag and lift force Drag and lift force [12] are computed for each face centroid and rendered into a texture similarly to the buoyancy force.

Water-object collision This force is calculated from the wave particle render texture. In addition of height value we also render wave particle propagation direction and speed in the remaining colour channels. Again we sum the texture in order to obtain the final force. While summing the forces we use buoyancy texture as an object silhouette stencil to precisely select which particles are affecting the object.

4.4 Object to Water Interaction

We have covered the process of the particle generated based on the wave particle distribution texture in section 4.1. This section denotes the distribution texture generation.

In order to create wave particle distribution texture we have to find out the silhouette of the submerged part of the floating object when looking from a top orthographic view and then distribute the wave effect (water volume) to the contour of that silhouette.

Figure 9 shows an example of such case. The silhouette is the union of the green and red part while the contour is the outer border of the red part.

The direction of the created wave is dependent on the direction of the object motion as shown in figure 10.

Important step of the object to water interaction stage is distributing the indirect wave effect to the object contour. This routine also smooths out the contour normals in order to uniformly cover the circular area around the object by the wave particle propagation angles.

Similarly to a parallel reduction approach we sum up neighbouring pixels into one. In each step we merge four adjacent pixels into one pixel. After few iterations when the texels are summed together the process is reversed and we reconstruct the original silhouette while using the textures from the intermediate steps. While descending to

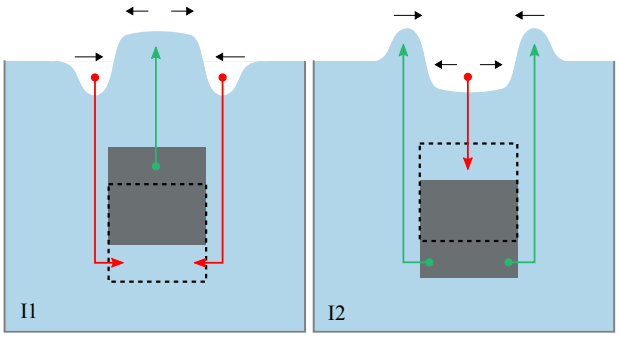

Figure 10: Different cases of wave propagation with respect to the position and the motion of the floating object. Striped line represents the object position in the previous time step. Cases I1 and I2 (inside) show the influence of both direct and indirect wave effect inside the volume.

higher texture resolutions we distribute the total indirect wave effect pixels recognized as contour pixels.

5 Results

To enhance the perception of the water surface, we render the surface with an approximative method [4] for light reflection and refraction. For reflection, the method uses a render of the flipped scene and modifies texture lookups according to the surface normal. We also use adaptive tessellation and tillable Perlin noise [6] as a height function to add high frequency details to water surface. Longitudinal waves are added in form of deformation of the x and z axis of the water plane similarly to Gerstner waves [11].

We have tested the performance of each simulation step in several testing cases.

Test case A In the first test case, particles are added to the buffer until it is full. This test cast measures the performance of the wave particle propagation procedure. Note that in the wave particle method the number of particles is not directly proportional to the quality of visual result. Especially when most of the particles in the system have low amplitudes. We have compared the performance of the wave particle routine to the performance mentioned in the original article. They mention three test cases with different maximal number of wave particles. Since the wave particle generation routine is part of our wave particle propagation routine, we have measured both of those steps at once. Therefore, in our comparison we have also summed corresponding columns from the original article. Our test ran on the following configuration: Intel Core i5-4590 3.30GHz, GIGABYTE GTX970 4GB, 8.0 GB RAM, Visual C++ compiler 18.00. We also show the computing power of the processor used in the original article (3.19 Gflops [9]) compared to our processor (12.5 Gflops [9]) to demonstrate the hardware difference. [4]

[4]Both values are measured on the same benchmark test.

Implementation	10k	600k	8M
CPU approach[12] (2007)	1.430	3.87	200.04
our GPU approach	0.196	1.94	22.50

Table 1: Comparison of the wave particle method implementation with varying number of particles (ms).

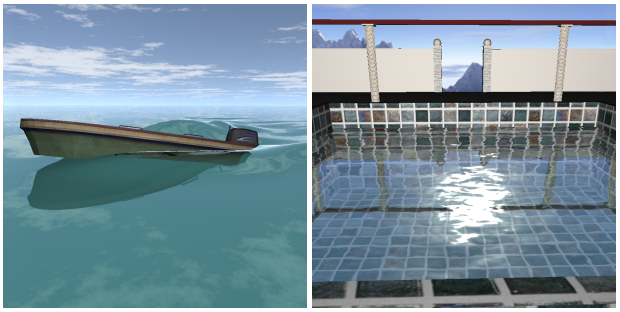

Figure 11: Test case A: ocean environment (left). Pool test case (right).

Test case B The second test case is a single boat floating in the ocean scene (figure 11). We created two modifications of this test case. In the first run we test the influence of the rigid body complexity on the performance of each phase. For example computation of drag and lift force is done per each face of the rigid body. Figure 12 shows the testing results. Simulations steps which are not dependent on the variable parameter are omitted.

Similarly we measured the influence of the wave particle render texture resolution (figure 12) on the performance. Note that this parameter affects not only the texture resolution, but also indirectly affects the number of fragment shader invocation etc.

Test case C The purpose of test case C is to show the usage of the wave particle method in a real simulation scenario. We have placed the boats in the scene in order to maximize the number of interaction between nearby floating object. Once the main boat moves, it creates wave front which pushes away the other floating objects. We measure the performance for different number of boats in the scene.

Test case D The fourth test case captures a scenario with high number of boats. Unlike the test case C, all the boats are moving.

Table 5 shows the average frame rates for test case C and D.

Ship count	1	2	4	8	16	32
Test case D	199	144	93	60	35	18
Test case C	145	136	97	60	33	24

Table 2: Average frame rate for the test case C and D.

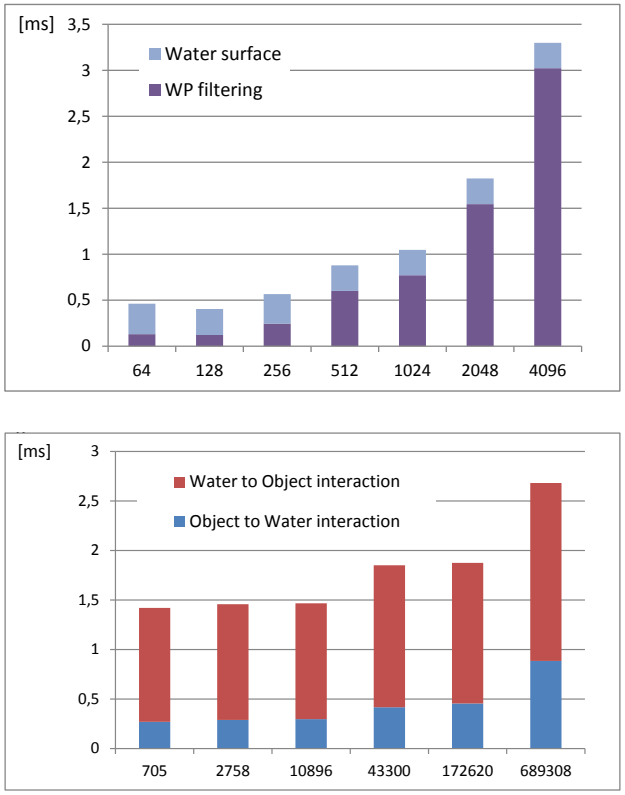

Figure 12: Mesh complexity (number of faces) influence on the simulation performance in ms (top). Particle render texture resolution complexity (bottom).

5.1 Limitations

We have developed such structure that each floating object contains its own render cameras, render textures, and wave particle buffers. This means that each boat in our simulation is independent and has every piece of localized information needed for the simulations. On the other hand, there is a structure using a single render texture with a single wave particle buffer shared for each floating object similarly to the implementation of Yuksel et al.

Our approach offers higher flexibility in terms of the floating object setup e.g. positions are not limited by the render texture resolution. Another advantage is that we

Figure 13: Test case C (left), and test case D (right).

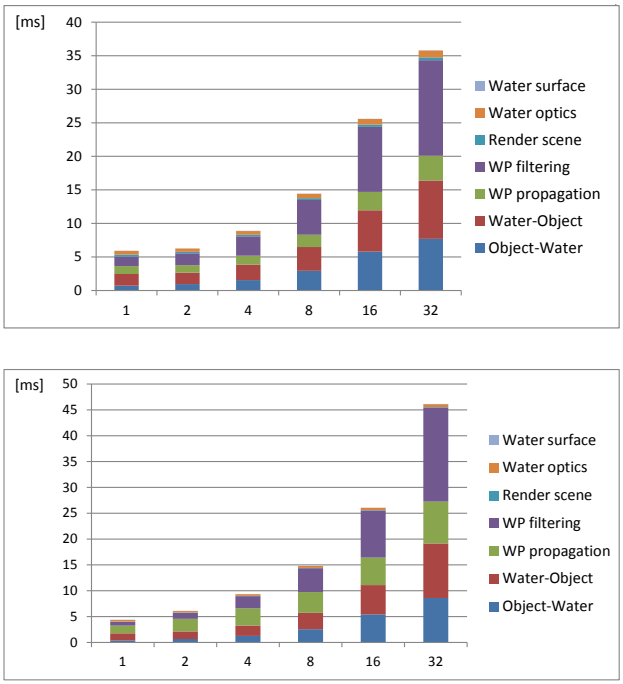

Figure 14: Result of the test case C (left), and case D (right).

can address only particles created by a concrete floating object, which is beneficial e.g. in LOD approach when we render only particles which come from a visible floating object.

On the other hand this approach has a limitation in the number of floating object in the simulation. Table 5 show significant performance drop for a large number of floating objects. Moreover, because we render the water surface in one step, all the vertical deviation functions are added at once to the global deviation. And since the number of GPU texture units is limited, we cannot apply all deviation function in one rendering step.

6 Conclusions

We have described theoretical background behind wave particle method for the purpose of real-time water simulation. We have showed how wave particles form continuous waves and that it can be used in an interactive environment. Consequently, we have implemented the wave particle method on the GPU. Part of our implementation is also the interaction between the water surface and a general shaped floating rigid body which can be controlled by user. We have measured and evaluated the performance of our GPU approach and showed that it offers plausible results at interactive frame rates.

References

[1] specification of OpenGL version 4.40.

[2] Haogang Chen, Qicheng Li, Guoping Wang, Feng Zhou, Xiaohui Tang, and Kun Yang. An efficient method for real-time ocean simulation. In *Technologies for E-Learning and Digital Entertainment*, volume 4469 of *Lecture Notes in Computer Science*, pages 3–11. Springer Berlin Heidelberg, 2007.

[3] CT Chou and LC Fu. Ships on real-time rendering dynamic ocean applied in 6-DOF platform motion simulator. In *CACS International Conference*, volume 3, 2007.

[4] Lund University Claes Johanson. Real-time water rendering : Introducing the projected grid concept, 2004.

[5] E. Galin, J. Schneider (editors), H. Cords, and O. Staadt. Real-time open water environments with interacting objects, 2009.

[6] John C. Hart. Perlin noise pixel shaders. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '01, pages 87–94, New York, NY, USA, 2001. ACM.

[7] Geoffrey Irving, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Trans. Graph.*, 25(3):805–811, July 2006.

[8] Juraj Onderik, Michal Chládek, and Roman Ďurikovič. SPH with small scale details and improved surface reconstruction. In *Proceedings of the 27th Spring Conference on Computer Graphics*, SCCG '11, pages 29–36, New York, NY, USA, 2013. ACM.

[9] Primatelabs. Cpu benchmarks.

[10] Karthik Raveendran, Chris Wojtan, and Greg Turk. Hybrid smoothed particle hydrodynamics. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 33–42, New York, NY, USA, 2011. ACM.

[11] Jerry Tessendorf. Simulating ocean water. in simulating nature: Realistic and interactive techniques. *ACM SIGGRAPH*, 2001.

[12] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), 2007.