SHALLOW WATER PROJECT REPORT

INF 585 - Computer Animation

March 2024

Nour RIZK

TABLE OF CONTENT

INTRODUCTION

The goal of this project is to implement a breaking wave visualisation based on a shallow water representation coupled with a particle based approach for the breaking waves modeling. We based our work on the 'Real-time Breaking Waves for Shallow Water Simulations' paper and implemented the shallow water representation for the surface of the fluid, as well as a wave front detection and wave line generation. Since the project is computation heavy to run in real-time on our machines, we thus didn't run the particle-based sheet representation for the actual breaking waves. However, we have developed the necessary code to detect the potential regions of overturning waves based on the height field, compute their advection over time and generate points and particles necessary to visualize the breaking waves.

1 PAPER OVERVIEW

The paper "Real-time Breaking Waves for Shallow Water Simulations" by Nils Thürey et al. introduces a method for enhancing shallow water simulations with the effect of overturning waves, which is particularly relevant for real-time applications where visual accuracy and computational efficiency are critical. Traditional 3D fluid simulations can capture wave breaking but are computationally expensive, especially for large bodies of water, making them unsuitable for real-time applications. The authors' method addresses this by combining height field-based shallow water simulations with particle-based sheets to represent breaking waves, thus providing a balance between computational efficiency and visual realism.

1.1 Shallow Water Simulation

Shallow water simulations are mathematical and computational techniques used to model the behavior of fluids—specifically water—in environments where the horizontal dimensions are much larger than the vertical dimension. These simulations are crucial for predicting and analyzing fluid dynamics in shallow bodies of water such as rivers, lakes, coastal areas, and the atmosphere.

The core of shallow water simulations lies in the shallow water equations (SWEs) which are a simplified form of the Navier-Stokes equations, the fundamental equations of fluid dynamics. The SWEs assume that the horizontal length scales are much larger than the vertical depth of the fluid, allowing the equations to

be simplified significantly. Despite this simplification, the shallow water equations can still accurately capture critical fluid dynamics phenomena such as wave propagation, tides, and the flow of water in rivers. The shallow water equations are as follows :

$$H_t = -\mathbf{u} \cdot \nabla H - H(\nabla \cdot \mathbf{u}) \tag{1}$$

$$u_t = -\mathbf{u} \cdot \nabla u - g h_x \tag{2}$$

$$v_t = -\mathbf{u} \cdot \nabla v - g h_y \tag{3}$$

with H_t, u_t, v_t represent the time derivatives of the height and the velocity components in the horizontal directions, $\mathbf{u}$ is the velocity vector, ∇H, h_x, h_y represent the gradient of the height field, and g is the acceleration due to gravity.

The paper introduces a novel method aimed at enhancing shallow water simulations with the effect of overturning waves, which is significant for applications like interactive environments. It addresses the challenge of simulating breaking waves within a shallow water framework. While full 3D fluid simulations can capture wave breaking, they're computationally intensive for real-time applications. Hence, it presents frame work to implement a breaking wave detection and rendering algorithm.

1.2 Breaking Waves

The "breaking wave simulation" involves first detecting steep wave fronts within a height field and marking them with line segments. These segments then generate fluid sheets represented by connected particles, which simulate the dynamics of breaking waves. When these sheets impinge on the water surface, they create particles that represent drops and foam, adding to the realism of the simulation. We detail here the first part of the simulation since it is the part we implemented in detail.

Detection and Construction of Wave Fronts : The algorithm detects steep wave fronts approaching shallow regions, where the height of the water decreases causing the wave to steepen and eventually overturn. This is particularly evident near beaches where the effect is amplified by the backflow from previous waves. The detection is based on a set of points in the shallow water grid that meet a specified criterion (4) related to the gradient of the fluid height and the fluid velocity (u). These points are then connected to form a line along the wave front.

$$|\nabla H(x)| > t_H \quad \text{and} \quad \nabla H(x) \cdot u(x) < 0. \tag{4}$$

Line Segmentation and Advection : After identifying points that potentially indicate a breaking wave, the method enlarges this set by adding nearby points to close gaps, and segments it to identify disconnected regions. For each segmented region, a line of connected points is constructed by following the height field's tangent vector. The process accounts for the direction of the wave front and aims to track the steepening wave accurately.

Advection and Projection : The constructed lines are adaptively tracked through the simulation, allowing them to merge with neighboring lines. This adaptability accurately simulates the dynamics of breaking waves. To track the front of a shallow water wave accurately, the method combines advection with the calculated wave velocity and projects this along the gradient direction onto the steepest gradient line on the wave front. The wave speed (c) for shallow water simulations is derived from the water depth (H) and gravity (g), using the equation :

$$c = \sqrt{gH}. \tag{5}$$

Movement Direction and Correction : The movement direction for each point on the wave line (L) is determined by the gradient of the height field from the previous time step. This ensures that each point is correctly positioned to simulate the wave's progression. If the wave crest has passed, the method projects the point to the maximum fluid height and then to the steepest gradient, adjusting the wave line as needed to maintain accuracy.

Adaptive Resampling : As the wave moves, its front can change length significantly. To manage this, the wave line is adaptively resampled : new points are added if the distance between neighbors exceeds a threshold, and points are merged if they are too close. This adaptive approach ensures the wave line accurately represents the wave's shape and movement, maintaining a scale similar to the simulation's grid size throughout its lifetime.

Particle Generation and Connectivity : Particles are generated along the wave line at specified time intervals, forming a wave patch. These particles maintain the same connectivity as the wave line, and new layers of particles are added over time to create a closed surface representation of the wave.

Velocity and Position realism : The velocity of each particle is determined by the velocity of the point on the wave line from which it was spawned. To accurately depict the wave crest's overturning, particles are initially positioned along the wave line and then adjusted to the wave crest's location.

Transition to Shallow Water Surface : Once particles detach from the main fluid body represented in the shallow water simulation, their movement is governed by their initial velocity and gravity. Collision detection with the shallow water surface ensures that the wave patch interacts realistically with the underlying water.

2 IMPLEMENTATION

2.1 Shallow water simulation

To implement the paper, we first created a `height_simulation` file gathering the functions required to compute and dynamically update the height field (`update_height` and the velocity `update_velocity` of the shallow water representation based on the equations (1), (2) and (3). We intialize the value of the different `gird_2D` elements (height, color, velocity...) in the `initialize_fields` in the `scene` script.

The rendering of the shallow water surface and height field is done using the `initialize_height_visual` and `update_height_visual` functions implemented in the `helper` code file. To render this shallow water model we used a 60 by 60 grid mesh. The position of the vertices is given by the `height_vector` which is the `vec3` 2D grid generalization of the `height_field` we base our computation upon. We divide every grid into two triangle meshes and specify the color of the vertices and fragments using the `height_to_colour` function.

The animation of the water surface can either be generated through the initialization of the height field with a spatially varying function, or through the user's interaction with the surface. First, the initialization of the height field we implement is based on the sine wave function in the `initialize_height_with_wave` function in the `scene` code file. Second, for the user interaction, the water surface is initialized with a constant value making it a flat plane with homogeneous color. The dynamic movement of water is due to the velocity injected in it through the mouse interaction with the surface. This is achieved through the `mouse_velocity_to_grid` function implemented in the `helper` code file.

2.2 Breaking Wave

To simulate the breaking waves we first need to detect potential overturning wave regions in the shallow water. The functions for computing the properties of the waves are implemented in the `height_simulation` code file. First, to detect the waves we implement the function `detect_wave_fronts` based on the logic of the equation (4). We implement the threshold as specified in the paper $t_H = \frac{p_H g \Delta t}{\Delta x}$, with $p_H = 0.25$, Δx the grid size, g the gravitational acceleration.

Once we have detected the grid's vertices that carry a wave front, we enlarge the detected set of points by adding nearby points to close gaps using the function `enlarge_point_set`. As multiple wave regions can be present at a single time step, this broadened set of points is segmented with a flood filling algorithm to identify disconnected regions. We do so by implementing the `segment_enlarged_set` based on the `flood_fill` function to identify disconnected regions.

Once we have segmented the enlarged point set we can construct the lines using the `construct_line_segment` function based on the height field's tangent vector. We track the evolution of the points and their advection using the function `update_wave_fronts`. We re project the points of the line based on the speed of the wave that we computed based on the equation (5). First, we project the points upwards towards the maximum wave height and then we project them along the steepest gradient. This is done using the functions `project_point_adjusted`, `project_point`, and `project_to_maximum_height`. Once the reprojection is done, we adaptively resample the points : new points are added if the distance between neighbors exceeds a threshold, and points are merged if they are too close using the function `adaptive_resample`.

Finally, we visualize the wave fronts based on the `visualize_wave_fronts_mod` function. We have ran through issues in visualizing the waves. It seems that the algorithm is too computational heavy, we can barely see an animation with a few fps.

3 RESULTS

We have implemented a shallow wave simulation as seen on *Fig.*1 With a flat plane animated by the user by gently moving the surface of the water with his mouse. Since the grid's boundary conditions are reflective, we can observe interference between different wave fronts generating a dynamic and interesting water surface. However, since the energy impulsed into the plane is conserved, we can see waves interfere destructively in a certain point, which would physically lead to the water going still. However, once this point absorbs the waves, it generates back a circle shaped wave, as if the water ripples. This issue could be solved by addressing the energy conservation and the physical equations behind the shallow water equations.

We implemented the detection of wave fronts, their enlargement, segmentation and the construction of wave lines. However, we had some trouble with the visualization of the lines since they are to computation heavy. Thus, we tried allowing a unique segment and a unique line to observe one wave per simulation. Furthermore, we didn't implement the particle based visualisation of the breaking waves and the collision with the shallow water framework. This is because the part of the paper we implemented took a lot of debugging from us.

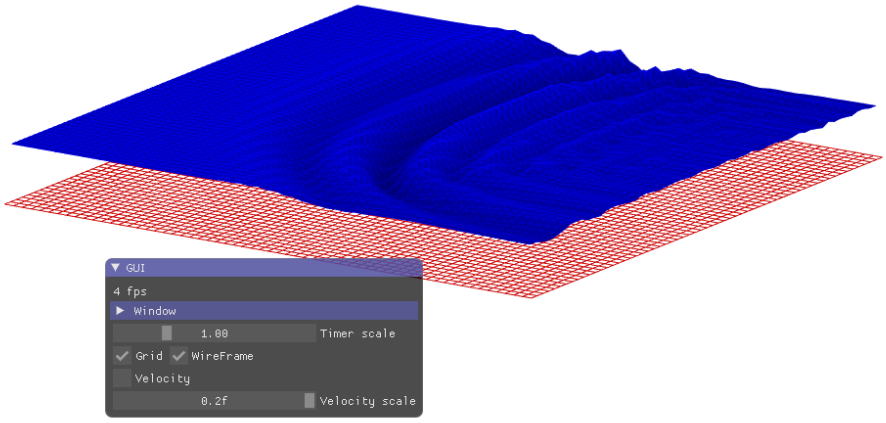

Figure 1 – Waves simulated by user mouse interaction

CONCLUSION

This paper effectively simulates breaking waves by using a combination of steepness detection, adaptive line tracking, and particle-based representation of fluid dynamics. The method strikes a balance between computational efficiency and visual realism, making it particularly suitable for real-time applications where both factors are critical. Finally, an interesting aspect of the authors' simulation is the two-way coupling of rigid bodies with the fluid, allowing for interactions between objects and the water surface. This addition opens up possibilities for more dynamic and interactive simulations, such as surf riding characters (which they implemented) and objects interacting with waves. Moreover, potential extensions of the paper might encompass improving the handling of chaotic wave conditions and enhancing the simulation of small-scale splashes, drops and ripples as mentioned in the result section.