Many applications of computer graphics involve fluid simulations of some sort. The most realistic simulations involve massive particle systems and take large amounts of time to run offline. For applications that are interactive, this is unreasonable and other solutions must be explored. Procedural fluids are acceptable for background situations where the user does not actually interact with it but it’s illusion of realism is rapidly destroyed when any object comes in contact with it. As such it is important to find a solution that allows interaction in real-time with fluids, primarily water.

While many solutions exist most all rely on reducing the complexity of the simulation. Wave particles reduce from a particle for each molecule of water to a particle for a wave that deforms the water surface as it passes. We chose to focus on height fields which reduce a 3-dimensional particle system to a 2 dimensional system of columns. This provides a massive reduction in computational intensity while maintaining an acceptable level of realism in most simulations.

In this project we implement a height field simulation and give the user the ability to drop a cube into the fluid in real-time. In this paper we will detail the internals of this implementation and problems we encountered. We will begin, however, by discussing the advantages and disadvantages of various types of fluid simulations.

Simulating water in real-time is very difficult as water is a large particle system where each particle interacts with its neighbors via a complex set of equations. Very realistic water simulations can be created with particle systems, however, it is not generally reasonable to expect them to run in real-time. This is due to the large amount of computational power required to update a complex 3-dimensional particle system. One solution to the problem is wave particles. Wave particles have a relatively low computational cost as the particles do not have to interact with each other and only have to apply a simple deformation to a plain. The problem with wave particles is that it is difficult to model many different types of object. The technique requires a separate set of information for each object resulting in large memory costs with many objects. Height fields were the solution we chose as they reduced the computational complexity of particle systems to two dimensions while still providing a great deal of realism and the ability to interact with objects relatively easily with minimal comparative computational costs. Shallow water equations provide a much more realistic simulation than height fields while requiring a minimal amount of extra memory but we were unable to reach a stage where we could implement them.

Any water simulation has two primary parts, the water movement and the water lighting. Our implementation focuses on water movement. The height field information is stored in two, two dimensional arrays. One for column height and one for column velocity. These columns are updated each frame. Each column calculates the force on it due to the four surrounding columns with the equation, f = c2\*(u[i+1,j]+u[i-1,j]+u[i,j+1]+u[i,j-1]– 4u[i,j])/h2. C represents the speed of a wave and h represents the width of a column. This equation is derived from a reduction of realistic wave equations to two dimensions. Our simulation takes place on a large scale with each column representing a 1 meter by 1 meter column of water. After the force is calculated it is applied to the velocity of each column. The force is multiplied by the amount of time that has passed since the last update, 1/60th of a second in our case. This ensures that steps remain constant even if framerate is changed. After all velocities have been updated the height of each column is modified by adding velocity multiplied by the time step. This process of forward Euler integration can lead to feedback loops if not controlled so we implement damping and clamping. Velocity is damped by a factor of 0.99 each step and column height is bounded between -100 and 100 although with the damping factor these extremes are rarely reached.

An important part of the simulation not yet discussed is boundary conditions…..

Object interaction is implemented with a cube. Our cube is therefore drawn with edges parallel to the edges of our columns. During each step of the simulation we find the location of the corners of the cube with respect to the indices of our columns. This allows us to compare the cubes height with that of the water. By subtracting the height of the base of the cube from the height of the water and clamping between 0 and cube width we calculate the amount of water displaced by the cube. The displacement of each column is stored in another 2-dimensional array and by comparing the new displacement with the old we are able to calculate the change in displacement of water by the cube. Once the total displacement and the change in displacement are known these values are used to calculate the force on the cube and the change in water level around the cube respectively. Any change in displaced water is added or subtracted directly to the columns surrounding the cube. Our height field inside the cube is not modified allowing water to cover the cube should it fall below the water level. Because the height field is not modified on the inside of the cube a feedback loop can occur. The displaced water is added to the surrounding columns which flow inward increasing the amount of displaced water by the cube, adding more water to the surrounding columns. This was solved in two ways. The first option sets the height of each internal column to halfway up the cube. This allows the following update due to column velocity to pull a column above the cube and prevents the feedback loop but gives the water a much more viscous appearance with it clinging to the cube, creating depressions and mounds in the surface when the cube is dropped and raised respectively. The second option is to use the average water level instead of the local column height. This leads to realistic simulations when the water is relatively flat but leads to disconnected water ripples and cube movement if the surface is far above or below average at that location. Once the water is updated the cube is moved using Forward Euler integration and standard physics equations. Force due to water is calculated with density\*gravity\*displacement and both it and gravity are applied to velocity and then velocity to cube location.

It is also important to ensure water is not added or removed from the simulation. To do this we initially calculate an average volume for a column. Upon each step we recalculate this average and add or subtract the difference from every column. This prevents small errors in calculation from drastically altering the volume of our fluid over time.

The final step is drawing the height field. To do this we created an additional array that contains a point representing the top left corner of each column. The x and z components were the indices of a location and the y the height. This array was transferred to the GPU via and VBO and VAO to as points. A geometry shader then took each point and expanded it into a column, adding sides with specified widths and a height determined by the height of each point. The columns were then shaded with Blinn-Phong shading. A simple set of Blinn-Phong shaders were applied to the cube as well, completing our lighting.

Our implementation works well for modeling the surface movement of water. Ripples are clearly visible and can be seen propagating off of walls. Additionally, we were able to simulate object interaction; our cube is able to both effect and be affected by water. The simulation also runs in real-time showing no significant slowdowns when run on our machine. While we were unable to implement a mesh over the surface of the water or any complex lighting techniques our focus was primarily on the simulation of water movement which our implementation successfully models.

Our implementation does have a few bugs, most notably both of our options for preventing feedback due to water displacement. Setting the water level to the center of the cube creates large depressions or hills as the cube moved below or above the water’s surface. While the water was still able to cover and separate from the cube the simulation was more akin to oil then water due to the surface tension. The other option produced realistic results when the water level was near level. However if for some reason the cube was falling into water that was significantly higher or lower than average the interaction would occur far later or earlier than it should. Other than this and minor problems due to our environmental constants our simulation does perform as it should.