Discretization and Simulation of the Multi-layer Shallow Water Equations

Leon Montealegre*

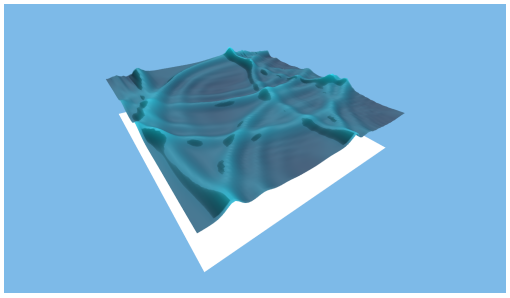

Figure 1: *A frame from the simulation of a single layer, flat bottom simulation with 2 initial drops*

Abstract

In this paper, I will describe my implementation of simulating fluid dynamics using the shallow water equations and simple central-difference discretization techniques. The goal is to have a quick and stable algorithm for the use of real-time applications.

Keywords: fluid simulation, shallow water equations

1 Introduction

The shallow water equations are a set of partial differential equations that describe the flow of a surface of a thin layer of fluid with constant density and in hydrostatic balance, bounded below by a set, rigid surface, and above by a free surface with negligible inertia. Such a form can be generalized to multiple layers of immiscible fluids lying atop of each other forming the "Multi-Layer" shallow water equations. The equations are derived from the Navier-Stokes equations that make many simplifications to the full 3D fluid dynamics simulation that allows the equations to be much easier to deal with and simulate. The main idealization is that the water is assumed to be much greater in horizontal scale then vertical scale (hence "shallow"), which leads to the assumption that all vertical motion is the same and so all z dependence of the Navier-Stokes equations are gone.

This presumption restricts the ability for waves to crash over themselves, meaning that you will never see the phenomena of wave-crashing as seen near beaches, but instead is meant to represent a patch of the water in the middle of the ocean with only mild disturbances in the height.

2 Motivation

My main motivation for choosing these equations are the amount of analytical work I have done with them previously that has allowed me to have a pretty great understanding of these equations and what they represent. I have loved fluid simulation for years and have never been at the level of understanding of any of the phenomena or methods until this semester. With the three very connected classes that I have taken this semester (Introduction to Numerical Methods for Differential Equations, Introduction to Geophysical Fluid Dynamics, and Computational Physics), I have finally achieved a level of understanding that I thought I was decades away from having.

*e-mail:leonm99@gmail.com

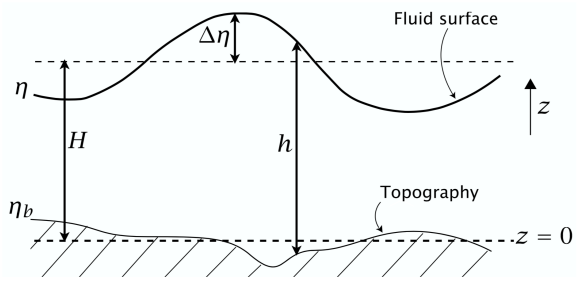

Figure 2: *A shallow water system. $h(x, y)$ is the thickness of a column, H is the mean thickness, $\eta(x, y)$ is the height of the surface and $\eta_b(x, y)$ is the height of the bottom floor.*

3 Background

The shallow water equations for a single-layer are given below:

$$\frac{D\mathbf{u}}{Dt} + \mathbf{f} \times \mathbf{u} = -g\nabla\eta \tag{1}$$

$$\frac{Dh}{Dt} + h\nabla \cdot \mathbf{u} = 0 \tag{2}$$

Where (1) is the momentum conservation equation with $\mathbf{u}$ being the horizontal velocity vector $\mathbf{u} = (u, v)$, $\mathbf{f}$ being the Coriolis parameter which represents the Coriolis rotation of the system, g being gravitational acceleration, and η representing the level of the surface of the fluid (as seen in Fig. 2).

(2) represents the mass conservation equation with $h(x, y)$ is the thickness of a fluid column where $h(x, y) = \eta(x, y) - \eta_b(x, y)$.

For both, $\frac{D}{Dt}$ is the so-called "material derivative" and is equal to $\frac{\partial}{\partial t} + (\mathbf{u} \cdot \nabla)$.

These equations can be extended for multiple layers and the forms

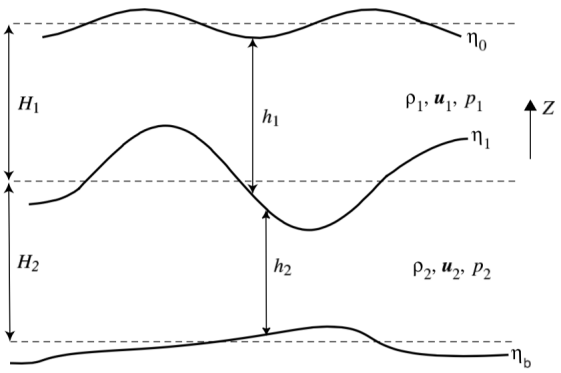

Figure 3: *A multi-layer shallow water system. A fluid of density ρ_1 lies over a denser fluid of density ρ_2.*

of those are given below:

$$\frac{D\mathbf{u}_n}{Dt} + \mathbf{f} \times \mathbf{u}_n = -\frac{1}{\rho_n}\nabla p_n \tag{3}$$

$$\frac{Dh_n}{Dt} + h_n \nabla \cdot \mathbf{u}_n = 0 \tag{4}$$

where

$$p_n = \sum_{i=0}^{n-1} (\rho_{i+1} - \rho_i)\eta_i$$

$$\eta_n = \eta_b + \sum_{i=n+1}^{N} h_i$$

n represents the n_{th} layer of the system. Each system needs to be solved in parallel with the rest.

4 Discretization

Discretization is the process of turning a continuous function into its discrete counterparts which is necessary for numerical computation with our digital systems. The process that will follow allows us to trivially compute the answer to the shallow water systems, with the cost of accuracy.

4.1 Grid

The first step for discretization is to define the grid that our system will live on. Each dimension of the equation needs to be discretized, so for our case, the horizontal space (x, y) and time, t. Mathematically, this means:

$$\begin{array}{lll} x_i = i\Delta x, & y_j = j\Delta y, & t_n = n\Delta t \\ i = 0, ..., N; & j = 0, ..., M; & t = 0, ..., \infty \end{array}$$

Where $\Delta x = \frac{W}{N}, \Delta y = \frac{H}{M}$ and W, H are the world-space scales of the system and N, M are the number of grid points in each dimension. Δt can be chosen as a small constant.

Finally, we let

$$\begin{array}{ll} u^n_{i,j} \simeq u(x_i, y_i, t_n) & v^n_{i,j} \simeq v(x_i, y_i, t_n) \\ h^n_{i,j} \simeq h(x_i, y_i, t_n) & \eta^n_{i,j} \simeq \eta(x_i, y_i, t_n) \end{array}$$

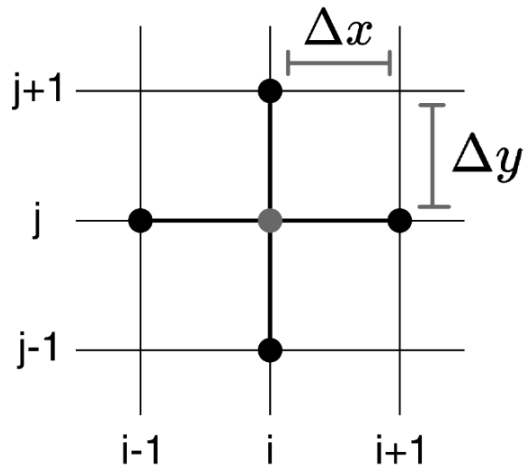

Figure 4: *Representation of a discretized 2D space with spacing of Δx and Δy*

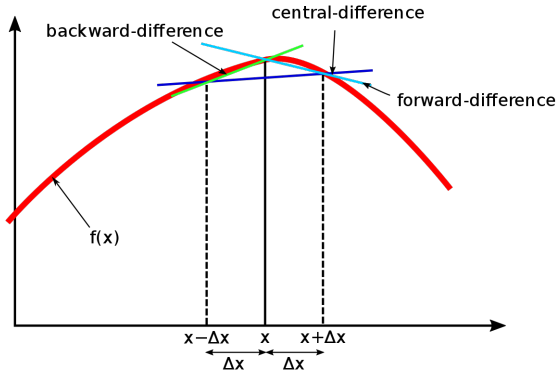

Figure 5: *Visualizations of the 3 first-order finite difference formulas*

4.2 Finite Difference Methods

The second step is to now discretize all of the operators acting in our equations, specifically the derivative operators. This is done through the process of finite difference methods. The 3 most common finite difference methods are shown in Figure 5.

With the example of the central difference formula, it is known that the derivative operator ($\frac{\partial}{\partial x}u(x_i, y_j, t_n)$) can be expressed as:

$$\frac{u(x_i + \Delta x, y_j, t_n) - u(x_i - \Delta x, y_j, t_n)}{2\Delta x} + O(\Delta x^2)$$

Which, using our compressed notation, and by throwing away the error term, leads to

$$\frac{\partial}{\partial x}u(x_i, y_j, t_n) \simeq \frac{u^n_{i+1,j} - u^n_{i-1,j}}{2\Delta x}$$

We are now going to compress this further by defining the four following finite difference operators:

$$\delta_{+x}u^n_{i,j} = u^n_{i+1,j} - u^n_{i,j} \quad \text{forward difference} \tag{5}$$
$$\delta_{-x}u^n_{i,j} = u^n_{i,j} - u^n_{i-1,j} \quad \text{backward difference} \tag{6}$$
$$\delta_{0x}u^n_{i,j} = u^n_{i+1,j} - u^n_{i-1,j} \quad \text{central difference} \tag{7}$$
$$\delta^2_x u^n_{i,j} = \delta_{+x}\delta_{-x}u^n_{i,j} \quad \text{2nd order central difference} \tag{8}$$

While these are all being defined for completeness-sake, the only one of use to us is the central difference finite difference operator as the shallow water equations have no 2nd order derivatives.

This means that the derivative can be written as:

$$\frac{\partial}{\partial x}u(x_i, y_j, t_n) \simeq \frac{1}{2\Delta x}\delta_{0x}u^n_{i,j}$$

4.3 Application

Finally, we can take our discretization approach and apply it to each of the shallow water equations. For simplicity-sake, we will take $f = 0$.

4.3.1 Momentum Equation $u(x, y, t)$ and $v(x, y, t)$

$$\frac{Du}{Dt} = -g\frac{\partial \eta}{\partial x} \Rightarrow \frac{\partial u}{\partial t} + (\mathbf{u}\cdot\nabla)u + g\frac{\partial \eta}{\partial x} = 0$$
$$\Rightarrow \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + g\frac{\partial \eta}{\partial x} = 0$$

Now using our discretization and finite difference operators, we can write this as:

$$\frac{\delta_{0t}u^n_{i,j}}{2\Delta t} + u^n_{i,j}\frac{\delta_{0x}u^n_{i,j}}{2\Delta x} + v^n_{i,j}\frac{\delta_{0y}u^n_{i,j}}{2\Delta y} + g\frac{\delta_{0x}\eta^n_{i,j}}{2\Delta x} = 0$$

Then, by multiplying both sides by $2\Delta t$ and expanding out the $\delta_{0t}u^n_{i,j}$ term, we get:

$$(u^{n+1}_{i,j} - u^{n-1}_{i,j}) + \Delta t\left(u^n_{i,j}\frac{\delta_{0x}u^n_{i,j}}{\Delta x} + v^n_{i,j}\frac{\delta_{0y}u^n_{i,j}}{\Delta y} + g\frac{\delta_{0x}\eta^n_{i,j}}{\Delta x}\right) = 0$$

Rearranging and isolating for $u^{n+1}_{i,j}$, we get:

$$u^{n+1}_{i,j} = u^{n-1}_{i,j} - \Delta t\left(u^n_{i,j}\frac{\delta_{0x}u^n_{i,j}}{\Delta x} + v^n_{i,j}\frac{\delta_{0y}u^n_{i,j}}{\Delta y} + g\frac{\delta_{0x}\eta^n_{i,j}}{\Delta x}\right)$$

This simple formula is very powerful; it's an explicit formula for the horizontal velocity u at a time $n+1$, using previous times n and $n-1$. This means that provided with the state of the system at $n = 1, 2$, we can solve for $n = 3$, which allows to solve for $n = 4$, and so on. This formula can be iterated on forever and leads to a reasonable approximation to our solution.

A very similar process can be performed for $v(x, y, t)$.

4.3.2 Mass Equation $h(x, y, t)$

$$\frac{Dh}{Dt} + h\nabla\cdot\mathbf{u} = 0 \Rightarrow \frac{\partial h}{\partial t} + (\mathbf{u}\cdot\nabla)h + h\nabla\cdot\mathbf{u} = 0$$
$$\Rightarrow \frac{\partial u}{\partial t} + u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial y} + h\frac{\partial u}{\partial x} + h\frac{\partial v}{\partial y} = 0$$

Now using our discretization and finite difference operators, we can write this as:

$$\frac{\delta_{0t}h^n_{i,j}}{2\Delta t} + u^n_{i,j}\frac{\delta_{0x}h^n_{i,j}}{2\Delta x} + v^n_{i,j}\frac{\delta_{0y}h^n_{i,j}}{2\Delta y} + h^n_{i,j}\frac{\delta_{0x}u^n_{i,j}}{2\Delta x} + h^n_{i,j}\frac{\delta_{0x}v^n_{i,j}}{2\Delta y} = 0$$

Then, by multiplying both sides by $2\Delta t$ and expanding out the $\delta_{0t}h^n_{i,j}$ term, we get:

$$(h^{n+1}_{i,j} - h^{n-1}_{i,j}) +$$
$$\Delta t\left(u^n_{i,j}\frac{\delta_{0x}h^n_{i,j}}{\Delta x} + v^n_{i,j}\frac{\delta_{0y}h^n_{i,j}}{\Delta y} + h^n_{i,j}\frac{\delta_{0x}u^n_{i,j}}{\Delta x} + h^n_{i,j}\frac{\delta_{0x}v^n_{i,j}}{\Delta y}\right) = 0$$

Rearranging and isolating for $h^{n+1}_{i,j}$, we get:

$$h^{n+1}_{i,j} = h^{n-1}_{i,j} -$$
$$\Delta t\left(u^n_{i,j}\frac{\delta_{0x}h^n_{i,j}}{\Delta x} + v^n_{i,j}\frac{\delta_{0y}h^n_{i,j}}{\Delta y} + h^n_{i,j}\frac{\delta_{0x}u^n_{i,j}}{\Delta x} + h^n_{i,j}\frac{\delta_{0x}v^n_{i,j}}{\Delta y}\right)$$

Similarly, this formula is now in explicit form and allows us to solve for h at a given time.

Using this and the momentum formulas, along with the fact that $h(x, y) = \eta(x, y) - \eta_b(x, y)$, we can now solve this system of PDEs.

I also performed this process for the multi-layer equations for the final result.

5 Drawbacks

Since the shallow water equations are based off of energy and mass conservation principles, it is given that all energy within the system needs to be conserved. However, by discretizing the space, we've introduced an inherent error into the solution. As a result, energy conservation is violated. And for the case of our approximations, the energy will tend to grow as t grows. Choosing a smaller value for Δt will slow this growth down, but will not stop it.

Using an implicit scheme or other, more complicated method would help counter this problem, yet would be much harder to implement and would slow the simulation down a lot as well.

An easier way to counteract this, is to add a damping term to the shallow water equations that help remove energy from the system and bring everything down to an equilibrium. This force is tacked onto the momentum equations and is as follows:

$$\frac{D\mathbf{u}}{Dt} + \mathbf{f}\times\mathbf{u} = -g\nabla\eta - b\mathbf{u} \tag{9}$$

where b is a constant that represents the amount of drag in the system (a higher value is more drag). Intuitively, this term represents a force that acts in the direction opposite of the current velocity and slows the velocity which effectively reduces the kinetic energy of the system.

6 Algorithm

In this section I will outline the algorithm I use for the fluid simulation along with explanations of important details.

Algorithm 1: Simulation Loop

foreach *layer l* **do**
  set u = layers[l].u
  set v = layers[i].v
  set η = layers[i].η
  set $h = \eta - \eta_b$
  set p = calc_pressure(l)
  foreach *u, v, η, h, p* = $u^n_{i,j}, v^n_{i,j}, \eta^n_{i,j}, h^n_{i,j}, p^n_{i,j}$ **do**
    $u = u - \Delta t \left(u \frac{\delta_{0x} u}{\Delta x} + v \frac{\delta_{0x} u}{\Delta y} + \frac{1}{\rho} \frac{\delta_{0x} p}{\Delta x} + bu \right)$
    $v = v - \Delta t \left(u \frac{\delta_{0x} v}{\Delta x} + v \frac{\delta_{0x} v}{\Delta y} + \frac{1}{\rho} \frac{\delta_{0x} p}{\Delta y} + bv \right)$
    $h = h - \Delta t \left(u \frac{\delta_{0x} h}{\Delta x} + v \frac{\delta_{0y} h}{\Delta y} + h \left(\frac{\delta_{0x} u}{\Delta x} + \frac{\delta_{0x} v}{\Delta y} \right) \right)$
  end
  impose boundary conditions
end

7 Rendering

For my 2D scenarios, rendering is trivially done through GNUPlot.

For the 3D scenarios, rendering is done with OpenGL and GLFW. The rendering for 3D was highly non-trivial and required many different aspects. The first challenge was procedurally generating an $N \times M$ plane.

From this, I created a displacement shader that takes in all the values for η as an attribute and displaces each vertex on the plane by that amount. Along with this, after calculating all the values for η, the normals for each vertex also had to be calculated which we calculated by finding the plane through the 4 neighboring points for each vertex. This allows nice shading to be acquired.

I performed the shading using a diffuse term, specular term for the "sun-reflection", fake reflection from the sky, and height-based coloring to mock how to the ocean appears darker as it's lower.

8 Future Work + Discussion

Because the method of discretization that I used was fairly simple, the program can be pretty trivially expanded for many similar types of simulations with the shallow water equations. Specifically, spherical coordinates would be a fairly reasonable extension as the displacement shader could be easily moved to a sphere. Accounting for Coriolis effects would also be quite trivial. The addition of wind forces and other external forces would be a neat addition. And finally, having a bottom floor that varies over time would also be a great addition.

The simulation worked out pretty nicely, with the exception of blowing up after a long period of time (which was fixed by adding a damping term).

Another interesting edge case is when the initial conditions is a very tall and thin bump. This splits into to smaller bumps that want to crash over themselves, yet are unable to since it's unsupported and causes major errors as shown in Figure 6

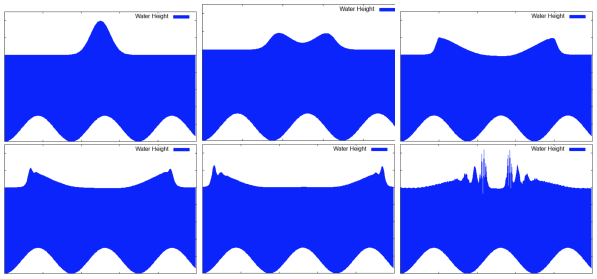

Figure 6: *2D simulation showing issues with large bumps that want to crash over themselves and cause major instability.*

9 Code

The code can be found on my Github at `https://github.com/LeonMontealegre/ShallowWaterEquations`.

References

VALLIS, G. K. 2005. *Atmospheric and Oceanic Fluid Dynamics*.

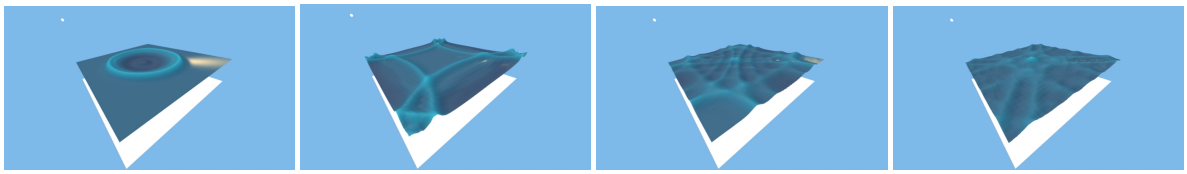

Figure 7: *A visualization of the single-layer shallow water equations in 3D with a drop in the center and a flat bottom floor*

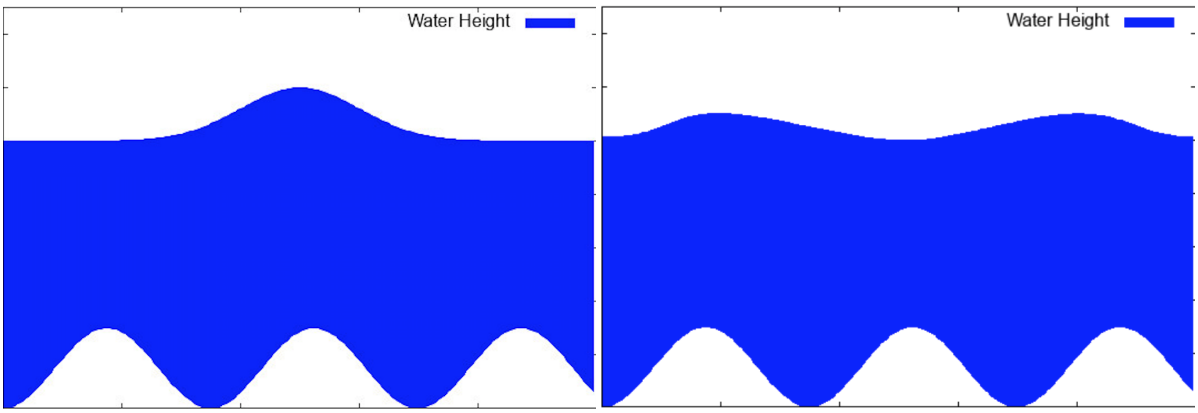

Figure 8: *A visualization of the single-layer shallow water equations in 2D with a drop in the center and a bumpy floor*

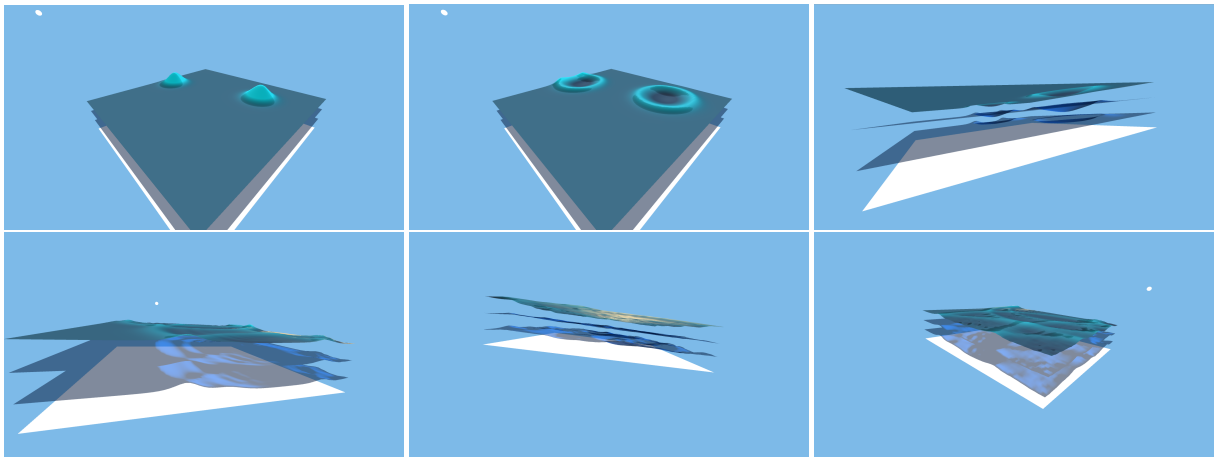

Figure 9: *A visualization of the multi-layer shallow water equations (w/ 3 layers) in 3D with 2 drop on the surface and a flat bottom floor*

(Videos + GIFs of the result can be found HERE: `https://imgur.com/a/ADXYhsc`)