SYRACUSE UNIVERSITY

Shallow Water Equations

PHY 307

Colin Richard Robinson

12/16/2011

Table of Contents

Introduction

In this project, the author simulates waves using MATLAB [1] and the shallow water equations (SWEs) in a variety of environments, from droplets in a bathtub to tsunamis in the Pacific. The SWEs are used to model waves, especially in water, where the wavelength is significantly larger than the depth of the medium. The SWEs are especially useful for modeling tsunamis. While one might not normally consider the 4km average depth of the Pacific Ocean to be "shallow water", the SWEs are still valid when applied to tsunamis which can have wavelengths in excess of 100km [2].

The SWEs are derived from the Navier-Stoke equations. The assumption made during the derivation is that the vertical velocity is small enough to be considered negligible. While an uneven seafloor does cause a vertical acceleration (beyond that of gravity), the vertical velocity is still negligible when compared to tsunamis which can travel in excess of 700 km/hr [2]. The vertical velocity of tsunami waves is most relevant when the wave approaches the shoreline. The steep incline of the continental rise at the coast is what gives tsunamis their great height. Many have described approaching tsunamis as not a wave but a wall of water. The goal of this project is to model the global propagation and reflections of tsunamis, not their effect on coastal regions. Because of this, the edges of land masses will be treated as perfect boundaries and the effect of continental selves and rises can be safely neglected.

Procedure

Empirical Formulations

The set of partial differential equations (PDEs) which make up the SWEs are provided below:

$$\frac{\partial h}{\partial t} + \frac{\partial (uh)}{\partial x} + \frac{\partial (vh)}{\partial y} = 0 \tag{1}$$

$$\frac{\partial (uh)}{\partial t} + \frac{\partial (u^2 h + \frac{1}{2} g h^2)}{\partial x} + \frac{\partial (uvh)}{\partial y} = 0 \tag{2}$$

$$\frac{\partial (vh)}{\partial t} + \frac{\partial (uvh)}{\partial x} + \frac{\partial (v^2 h + \frac{1}{2} g h^2)}{\partial y} = 0 \tag{3}$$

Where the independent variables *x*, *y*, and *t* make up the special dimensions and time. The dependent variables are the height *h* with respect to the surface (from henceforth referred to as displacement) and the two-dimensional velocities *u* and *v* [3]. The partial derivatives taken with respect to the same term ($\partial x, \partial y, \partial t$) are grouped into vectors and rewritten as a single hyperbolic partial differential equation.

The vectors:

$$H = \begin{pmatrix} h \\ uh \\ vh \end{pmatrix} \tag{4}$$

$$U(H) = \begin{pmatrix} uh \\ u^2h + \frac{1}{2}gh^2 \\ uvh \end{pmatrix} \tag{5}$$

$$V(H) = \begin{pmatrix} vh \\ uvh \\ v^2h + \frac{1}{2}gh^2 \end{pmatrix} \tag{6}$$

The single hyperbolic PDE:

$$\frac{\partial H}{\partial t} + \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} = 0 \tag{7}$$

H is a two dimensional height displacement matrix. It's also used to draw the mesh animation. *U* is a 2D matrix which stores the velocity of each point in the x-direction (a positive number represents a velocity in the positive x-direction; the reverse is true for negative numbers). *V* is 2D matrix of the velocities in the positive and negative y-directions.

There is a numerical method for solving hyperbolic partial differential equations known as the Lax-Wendroff method. Unlike the Euler methods which calculate each step of a function, the Lax-Wendroff method involves first calculating a half step and then using the result from the half step to calculate the full step [3]. It is formally expressed below.

If a function has the form:

$$\frac{\partial f(x,t)}{\partial t} = \frac{\partial g(f(x,t))}{\partial x} \tag{8}$$

The first step is:

$$\frac{f_{i+1/2}^{n+1/2} - \frac{f_i^n + f_{i+1}^n}{2}}{\left(\frac{1}{2}\right) * \Delta t} = \frac{g_{i+1}^n - g_i^n}{\Delta x} \tag{9}$$

And the second step is:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} = \frac{g_{i+1/2}^{n+1/2} - g_{i-1/2}^{n+1/2}}{\Delta x} \tag{10}$$

To quickly demonstrate how this applies to the shallow water equations, substitute *H* for *f* and *U(H)* for *g(f)*. Equation (9) can be rewritten as:

$$Hx_{i+1/2}^{n+1/2} = \frac{H_{i+1}^n + H_i^n}{2} + (2 * \Delta t)\frac{U_{i+1}^n - U_i^n}{\Delta x} \tag{11}$$

$$Hy_{j+1/2}^{n+1/2} = \frac{H_{j+1}^n + H_j^n}{2} + (2 * \Delta t)\frac{V_{j+1}^n - V_j^n}{\Delta y} \tag{12}$$

Because *H* is 2 dimensional, the step must be calculated twice, once for each dimension. The half steps are stored in separate matrices *Hx* and *Hy* to be reintroduced during the second step. The half steps for *U* and *V* are calculated in the same manner and stored in *Ux*,*Uy* and *Vx*,*Vy*. Equation (10) is rewritten as:

$$H_{i,j}^{n+1} = H_{i,j}^n + \Delta t\frac{Ux_{i+1/2}^{n+1/2} - Ux_{i-1/2}^{n+1/2}}{\Delta x} + \Delta t\frac{Vy_{i+1/2}^{n+1/2} - Vy_{i-1/2}^{n+1/2}}{\Delta y} \tag{13}$$

It is worth noting that the *n* superscript references the time of the event. This data is never stored in the simulation as it is immediately presented to the user in the form of an animation. It is also worth noting that the properties of MATLAB prevent referencing fractional array locations, therefore half steps are represented as single steps and single steps are represented as double steps. This has no effect on the simulation, but may be a source of confusion. The MATLAB representations of equations (11-13) are:

```
Hx(i,j) = (H(i+1,j+1)+H(i,j+1))/2 -
dt/(2*dx)*(U(i+1,j+1)-U(i,j+1));
```
(14)

```
Hy(i,j) = (H(i+1,j+1)+H(i+1,j))/2 -
dt/(2*dy)*(V(i+1,j+1)-V(i+1,j));
```
(15)

```
H(i,j) = H(i,j) - (dt/dx)*(Ux(i,j-1)-Ux(i-1,j-1)) -
(dt/dy)*(Vy(i-1,j)-Vy(i-1,j-1));
```
(16)

The Bathtub Model

The simplest simulation of the SWEs is the “bathtub” model. The environment is a square mesh with 4 perfectly reflective boundaries. The first boundary condition states that the velocities of *U* at the positive and negative edges of the x-dimension are equal in magnitude and opposite in direction to the velocity one row away (ie. The wave “bounces off” the edge). The second boundary condition is the same but it applies to *V* and in the y-direction. The forced velocity at the edges is a source of instability in the algorithm which causes the height of the boundary edges to rapidly increase. The third boundary

condition replaces the height of the boundary edge with the height of the next row over; this prevents the instability.

The displacement matrix is initialized to a level height of 1. A hill shaped displacement is generated and added to the matrix. The figure is then drawn via the **surf()** function and the simulation enters an infinite loop where the SWEs are used to update the *H*, *U*, and *V* matrices. The figure is animated by replacing the z-dimensional data with the updated displacement matrix via **set(grid, 'zdata', H);** and then calling **drawnow**.

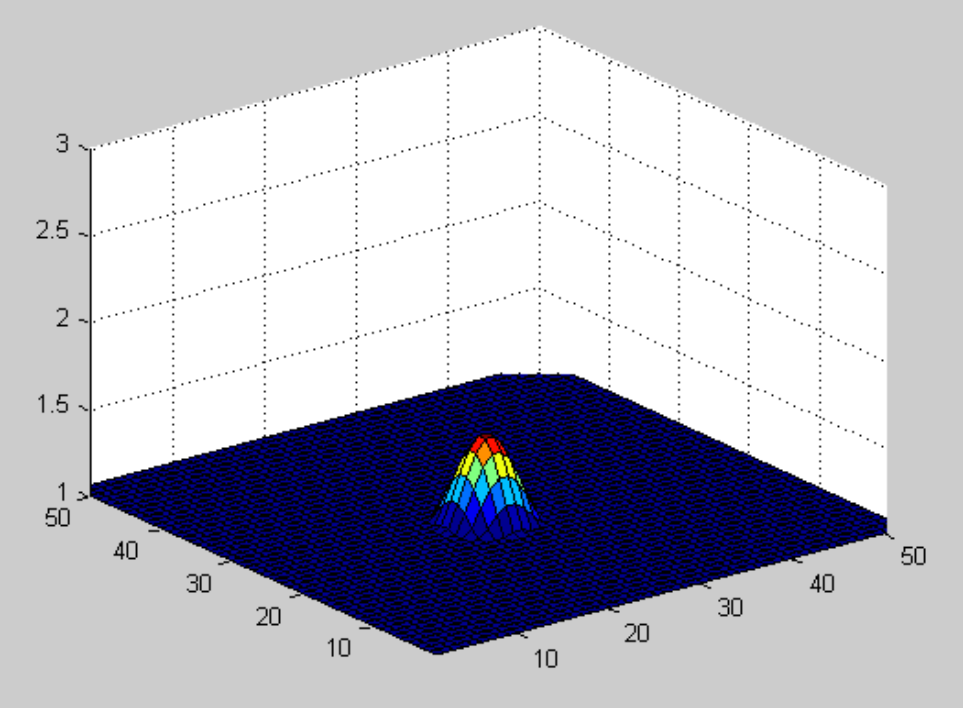

Figure 1: The initial displacement is added to the mesh.

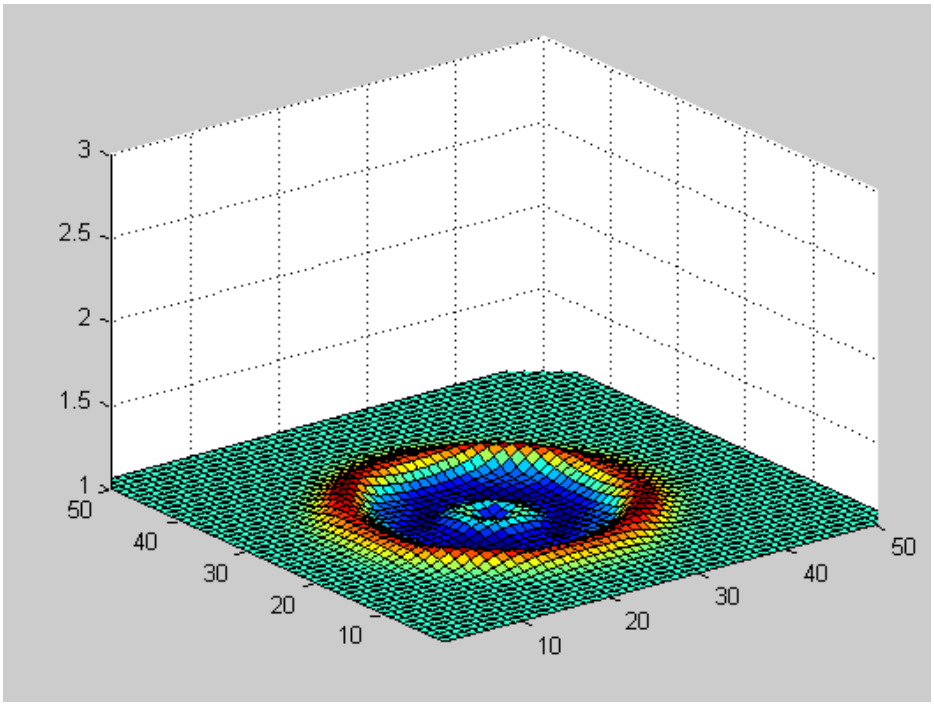

Figure 2: The displacement collapses, creating outwards momentum.

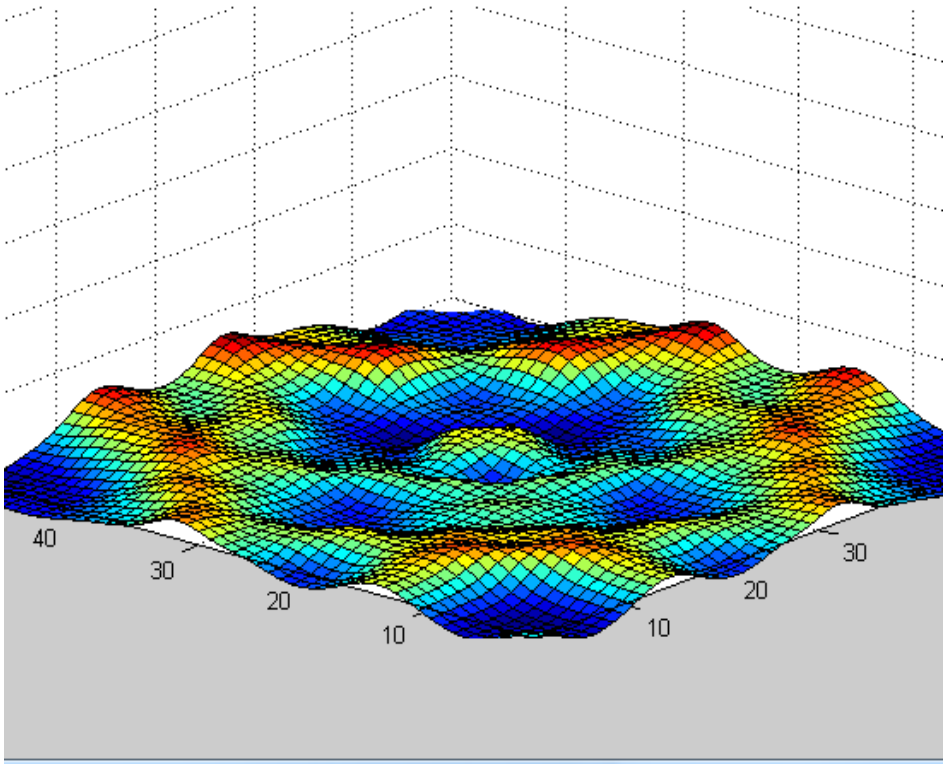

Figure 3: The surface after several seconds.

Portals

Spherical objects such as the Earth do not have edges or boundaries. The first step in transitioning from the square modal to a spherical model is to treat the square as a spherical object. The boundary conditions were altered so waves could pass through walls and seamlessly appear on the opposite side. This was done by averaging the properties of opposite sides. The new boundary conditions:

```
% height
x_edge = ( H(2,:) + H(n+1,:) )./2;
H(1,:) = x_edge;
H(n+2,:) = x_edge;

y_edge = ( H(:,2) + H(:,n+1) )./2;
H(:,1) = y_edge;
H(:,n+2) = y_edge;

% velocity at the x edges
x_v = ( U(2,:) + U(n+1,:) )./2;
U(1,:) = x_v;
U(n+2,:) = x_v;

% velocity at the y edges
y_v = ( V(:,2) + V(:,n+1) )./2;
V(:,1) = y_v;
V(:,n+2) = y_v;
```

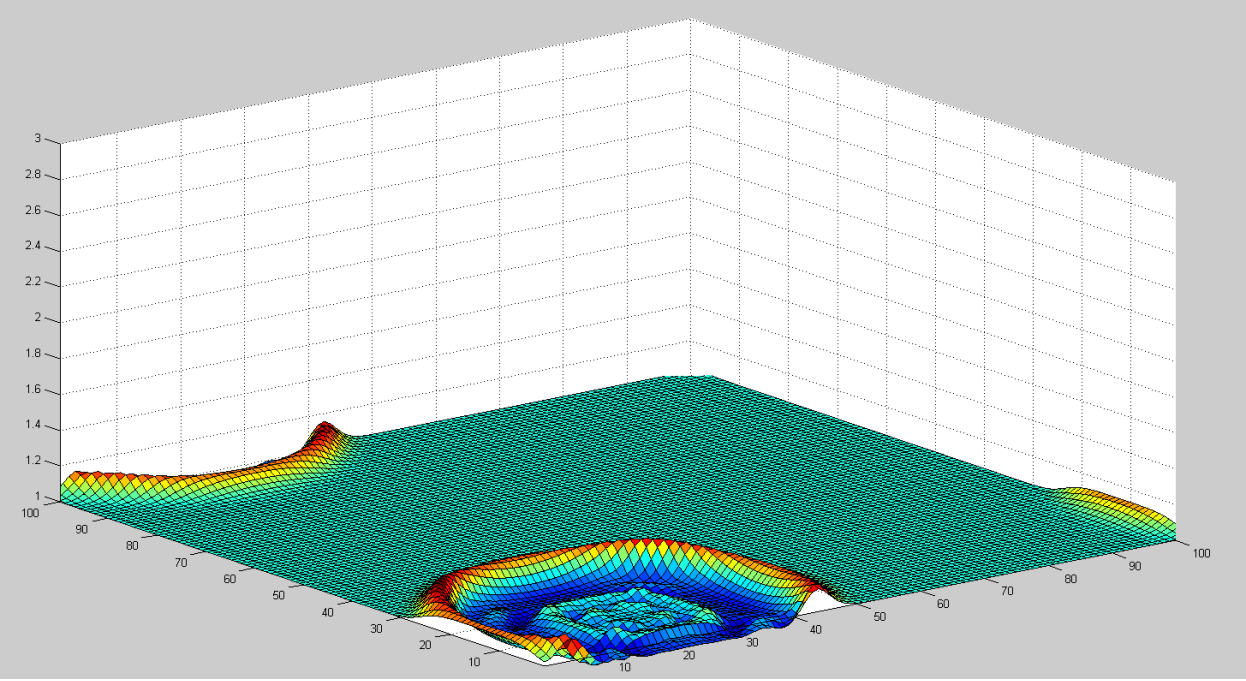

Figure 4: The wave is shown passing though the wall and appear on the other side

Spheres

Because of the nature of the displacement matrix, it is easy to apply onto other surfaces. In order to simulate a sphere, a sphere is first generated in rectangular coordinates using the built in MATLAB functions. The rectangular coordinates are then converted to spherical coordinates. The radius matrix is replaced with the displacement matrix and then converted back to rectangular coordinates for plotting. The entire code for the spherical simulation is in Appendix B, but the important lines are:

```
[x,y,z] = sphere(n+1);
[t,p,r] = cart2sph(x,y,z);
[x,y,z] = sph2cart(t,p,H);
set(grid,'xdata',x,'ydata',y,'zdata',z);
drawnow
```

Due to the lack of objects (land masses) on this sphere, there are now two initial displacements to make a more interesting plot. The figures below show the effect of two massive asteroids colliding into the Earth simultaneously.

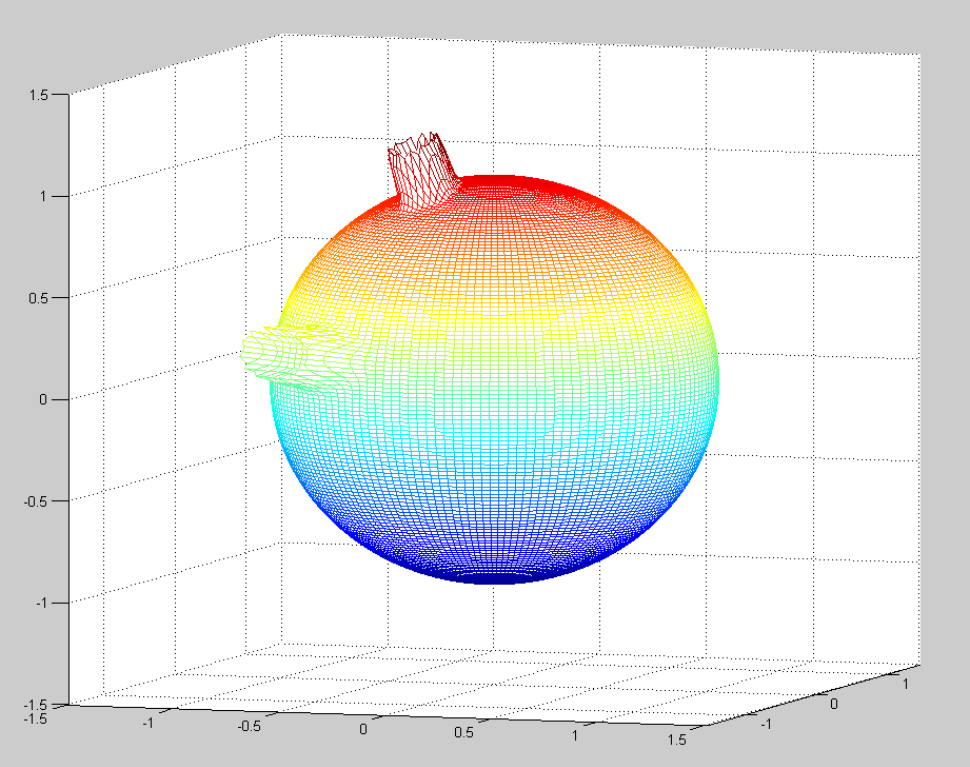

Figure 5: Initial asteroid impact

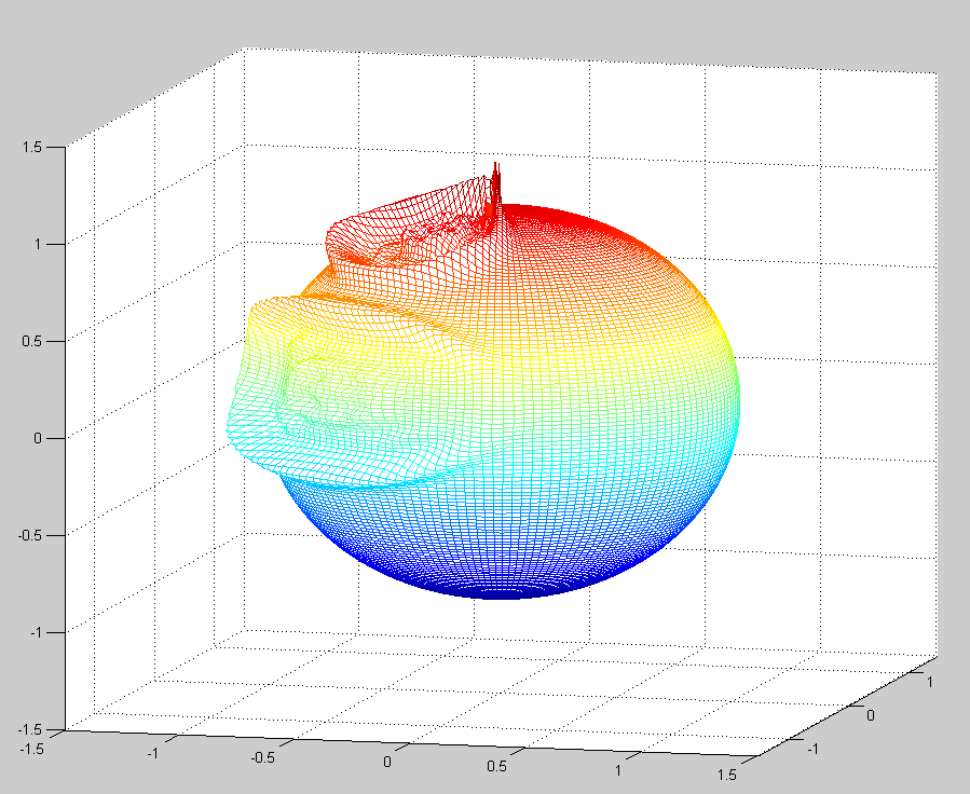

Figure 6: Wave patterns appear after impact

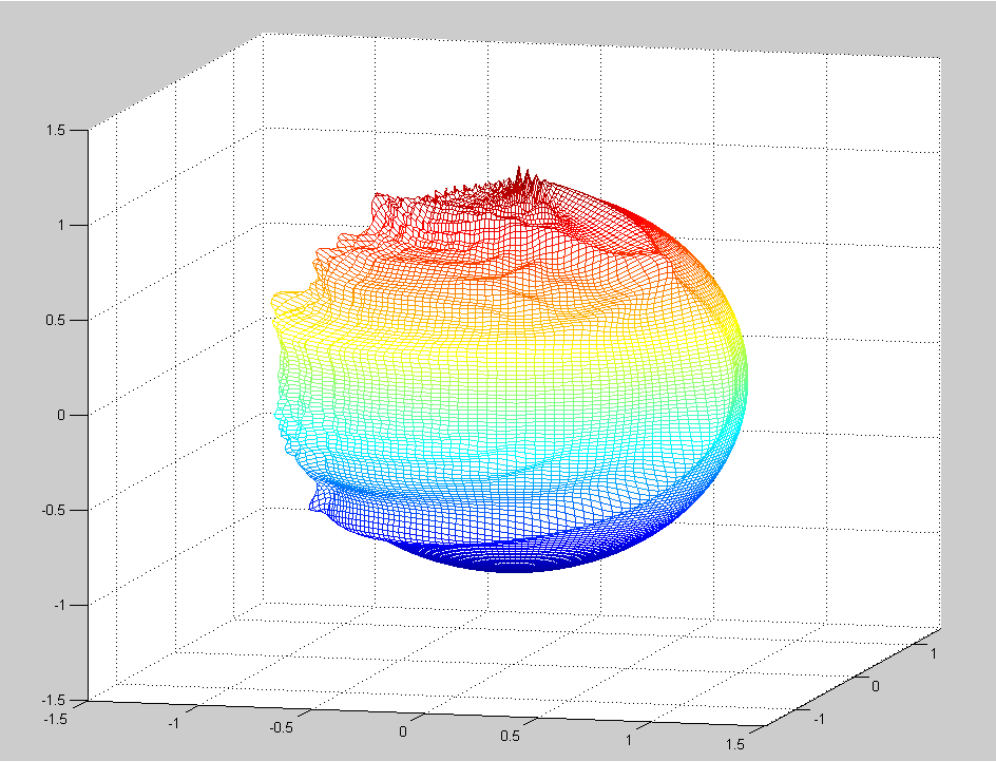

Figure 7: The waves collide and form interesting new reflections

Conclusion

The project was mostly a success. Regrettably, I was unable to add landmasses to the spherical model before the project deadline. The spherical model is also quite unstable because the entire sphere is treated as a moldable object whereas the Earth's crust would prevent this behavior in a real world scenario. The spherical model is also just a simple projection of the rectangular model onto a spherical surface. It would be interesting, albeit time consuming, to rederive the Shallow Water Equations in spherical coordinates from the original Navier-Stokes equations. Using true spherical coordinates would provide a much better model, but at that point, MATLAB is probably no longer the best tool for the job.

References

[1] "How Do Tsunamis Differ from Other Water Waves?" *Earth and Space Sciences at the University of Washington*. Web. <http://www.ess.washington.edu/tsunami/general/physics/characteristics.html>.

[2] *MATLAB*. Vers. 2011b. Natick, MA: MathWorks, 2011. Computer software.

[3] Moler, Cleve. "Chapter 18: Shallow Water Equations." *Experiments with MATLAB*. MathWorks. Web. <http://www.mathworks.com/moler/exm/chapters.html>.

[4] Garc´ıa-Navarr, P. "The Shallow Water Equations: An Example of Hyperbolic System." *Fluid Mechanics. CPS. University of Zaragoza* (2008). <http://iuma.unizar.es/maths_water/Actas/089.pdf>.

[5] Rifle, Guillaume. "A Shallow Water Numerical Model V0 . 1." *Fluid Mechanics. CPS. University of Zaragoza* (2006). Web. <http://www.scribd.com/doc/2022736/ShallowWater0-2>.

Appendix A – Bathtub Model

```
clf;
clear all;

% define the grid size
n = 50;

dt = 0.01;
dx = 1;
dy = 1;
g = 9.8;

H = ones(n+2,n+2);  % displacement matrix (this is what gets drawn)
U = zeros(n+2,n+2); % x velocity
```

```
V = zeros(n+2,n+2); % y velocity

% draw the mesh
grid = surf(H);
axis([1 n 1 n 1 3]);
hold all;

% create initial displacement
[x,y] = meshgrid( linspace(-3,3,10) );
R = sqrt(x.^2 + y.^2) + eps;
Z = (sin(R)./R);
Z = max(Z,0);

% add displacement to the height matrix
w = size(Z,1);
i = 10:w+9;
j = 20:w+19;
H(i,j) = H(i,j) + Z;


% empty matrix for half-step calculations
Hx = zeros(n+1,n+1);
Hy = zeros(n+1,n+1);
Ux = zeros(n+1,n+1);
Uy = zeros(n+1,n+1);
Vx = zeros(n+1,n+1);
Vy = zeros(n+1,n+1);

while 1==1

   % redraw the mesh
   set(grid, 'zdata', H);
   drawnow

   % blending the edges keeps the function stable
   H(:,1) = H(:,2);
   H(:,n+2) = H(:,n+1);
   H(1,:) = H(2,:);
   H(n+2,:) = H(n+1,:);

   % reverse direction at the x edges
   U(1,:) = -U(2,:);
   U(n+2,:) = -U(n+1,:);

   % reverse direction at the y edges
   V(:,1) = -V(:,2);
   V(:,n+2) = -V(:,n+1);


   % First half step
   i = 1:n+1;
   j = 1:n+1;

   % height
   Hx(i,j) = (H(i+1,j+1)+H(i,j+1))/2 - dt/(2*dx)*(U(i+1,j+1)-U(i,j+1));
   Hy(i,j) = (H(i+1,j+1)+H(i+1,j))/2 - dt/(2*dy)*(V(i+1,j+1)-V(i+1,j));

   % x momentum
```

```
    Ux(i,j) = (U(i+1,j+1)+U(i,j+1))/2 -  ...
            dt/(2*dx)*( U(i+1,j+1).^2./H(i+1,j+1) - U(i,j+1).^2./H(i,j+1) + ...
                    g/2*H(i+1,j+1).^2 - g/2*H(i,j+1).^2 ...
                   );

    Uy(i,j) = (U(i+1,j+1)+U(i+1,j))/2 - ...
            dt/(2*dy)*( (V(i+1,j+1).*U(i+1,j+1)./H(i+1,j+1)) - (V(i+1,j).*U(i+1,j)./H(i+1,j)) );

    % y momentum
    Vx(i,j) = (V(i+1,j+1)+V(i,j+1))/2 - ...
            dt/(2*dx)*((U(i+1,j+1).*V(i+1,j+1)./H(i+1,j+1)) - ...
            (U(i,j+1).*V(i,j+1)./H(i,j+1)));

    Vy(i,j) = (V(i+1,j+1)+V(i+1,j))/2 - ...
            dt/(2*dy)*((V(i+1,j+1).^2./H(i+1,j+1) + g/2*H(i+1,j+1).^2) - ...
            (V(i+1,j).^2./H(i+1,j) + g/2*H(i+1,j).^2));

    % Second half step
    i = 2:n+1;
    j = 2:n+1;

    % height
    H(i,j) = H(i,j) - (dt/dx)*(Ux(i,j-1)-Ux(i-1,j-1)) - ...
                    (dt/dy)*(Vy(i-1,j)-Vy(i-1,j-1));
    % x momentum
    U(i,j) = U(i,j) - (dt/dx)*((Ux(i,j-1).^2./Hx(i,j-1) + g/2*Hx(i,j-1).^2) - ...
                    (Ux(i-1,j-1).^2./Hx(i-1,j-1) + g/2*Hx(i-1,j-1).^2)) ...
                  - (dt/dy)*((Vy(i-1,j).*Uy(i-1,j)./Hy(i-1,j)) - ...
                    (Vy(i-1,j-1).*Uy(i-1,j-1)./Hy(i-1,j-1)));
    % y momentum
    V(i,j) = V(i,j) - (dt/dx)*((Ux(i,j-1).*Vx(i,j-1)./Hx(i,j-1)) - ...
                    (Ux(i-1,j-1).*Vx(i-1,j-1)./Hx(i-1,j-1))) ...
                  - (dt/dy)*((Vy(i-1,j).^2./Hy(i-1,j) + g/2*Hy(i-1,j).^2) - ...
                    (Vy(i-1,j-1).^2./Hy(i-1,j-1) + g/2*Hy(i-1,j-1).^2));

end
```

Appendix B – Spherical Model

```
clf;
clear all;

% define the grid size
n = 150;

dt = 0.01;
dx = 1;
dy = 1;
g = 9.8;

H = ones(n+2,n+2);  % displacement matrix (this is what gets drawn)
U = zeros(n+2,n+2); % x velocity
V = zeros(n+2,n+2); % y velocity

% draw the mesh
grid = mesh(H);
```

```
axis([-1.5 1.5 -1.5 1.5 -1.5 1.5]);
hold all;

% create initial displacement
[x,y] = meshgrid( linspace(-3,3,10) );
R = sqrt(x.^2 + y.^2) + eps;
Z = (sin(R)./R);
Z = max(Z,0);

% add displacement to the height matrix
w = size(Z,1);
i = 130:w+129;
j = 20:w+19;
H(i,j) = H(i,j) + Z;

% add another one
i = 80:w+79;
j = 20:w+19;
H(i,j) = H(i,j) + Z;

% empty matrix for half-step calculations
Hx = zeros(n+1,n+1);
Hy = zeros(n+1,n+1);
Ux = zeros(n+1,n+1);
Uy = zeros(n+1,n+1);
Vx = zeros(n+1,n+1);
Vy = zeros(n+1,n+1);

while 1==1

    [x,y,z] = sphere(n+1);
    [t,p,r] = cart2sph(x,y,z);
    [x,y,z] = sph2cart(t,p,H);
    set(grid,'xdata',x,'ydata',y,'zdata',z);
    drawnow

    % First half step
    i = 1:n+1;
    j = 1:n+1;

    % height
    Hx(i,j) = (H(i+1,j+1)+H(i,j+1))/2 - dt/(2*dx)*(U(i+1,j+1)-U(i,j+1));
    Hy(i,j) = (H(i+1,j+1)+H(i+1,j))/2 - dt/(2*dy)*(V(i+1,j+1)-V(i+1,j));

    % x momentum
    Ux(i,j) = (U(i+1,j+1)+U(i,j+1))/2 -  ...
              dt/(2*dx)*( U(i+1,j+1).^2./H(i+1,j+1) - U(i,j+1).^2./H(i,j+1) + ...
                          g/2*H(i+1,j+1).^2 - g/2*H(i,j+1).^2 ...
                        );

    Uy(i,j) = (U(i+1,j+1)+U(i+1,j))/2 - ...
              dt/(2*dy)*( (V(i+1,j+1).*U(i+1,j+1)./H(i+1,j+1)) - (V(i+1,j).*U(i+1,j)./H(i+1,j))
);

    % y momentum
    Vx(i,j) = (V(i+1,j+1)+V(i,j+1))/2 - ...
              dt/(2*dx)*((U(i+1,j+1).*V(i+1,j+1)./H(i+1,j+1)) - ...
              (U(i,j+1).*V(i,j+1)./H(i,j+1)));

    Vy(i,j) = (V(i+1,j+1)+V(i+1,j))/2 - ...
              dt/(2*dy)*((V(i+1,j+1).^2./H(i+1,j+1) + g/2*H(i+1,j+1).^2) - ...
              (V(i+1,j).^2./H(i+1,j) + g/2*H(i+1,j).^2));

    % Second half step
    i = 2:n+1;
    j = 2:n+1;

    % height
```

```
    H(i,j) = H(i,j) - (dt/dx)*(Ux(i,j-1)-Ux(i-1,j-1)) - ...
                        (dt/dy)*(Vy(i-1,j)-Vy(i-1,j-1));
    % x momentum
    U(i,j) = U(i,j) - (dt/dx)*((Ux(i,j-1).^2./Hx(i,j-1) + g/2*Hx(i,j-1).^2) - ...
                        (Ux(i-1,j-1).^2./Hx(i-1,j-1) + g/2*Hx(i-1,j-1).^2)) ...
                      - (dt/dy)*((Vy(i-1,j).*Uy(i-1,j)./Hy(i-1,j)) - ...
                        (Vy(i-1,j-1).*Uy(i-1,j-1)./Hy(i-1,j-1)));
    % y momentum
    V(i,j) = V(i,j) - (dt/dx)*((Ux(i,j-1).*Vx(i,j-1)./Hx(i,j-1)) - ...
                        (Ux(i-1,j-1).*Vx(i-1,j-1)./Hx(i-1,j-1))) ...
                      - (dt/dy)*((Vy(i-1,j).^2./Hy(i-1,j) + g/2*Hy(i-1,j).^2) - ...
                        (Vy(i-1,j-1).^2./Hy(i-1,j-1) + g/2*Hy(i-1,j-1).^2));

end
```