Rotterdam Must Die: Triangular Finite Volume Methods Applied to the Shallow Water Equations

Luke Bovard and Katharine Hyatt
University of Waterloo

June 29, 2012

Introduction

In this paper we apply the method of finite volumes using a triangular mesh with a Roe solver to solve the shallow water wave equations. In order to demonstrate the advantages of using a triangular mesh, we solve two problems that are not easily solved using rectangular finite volume methods. We first solve the classic problem of a broken circular dam and then apply the scheme to the Maeslantkering, a movable barrier along the Nieuwe Waterweg in Holland used to regulate water flow from storms into the shipping canal, to demonstrate the complicated geometry that triangular meshes are able to model.

Background

Failure of Finite Difference Schemes

The simplest scheme avaiable to solve PDEs numerically is to use a finite difference formula. The ideas behind finite difference formula is to use simple approximations to the derivatives and numerically solve the resulting system of equations. However, finite difference schemes are fairly limited in their scope. For example, consider the well known one dimensional PDE Burger's equation [Ach90]

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$$

the exact solution of this equation is well known and a prototypical example in the method of characteristics. However, suppose we try and apply a finite difference scheme to the above. Discretising both derivatives we have that

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + \frac{u_j^n}{\Delta x}(u_j^n - u_{j-1}^n) = 0$$

where u_j^n is the approximation at position j and time n. Suppose that we subject the above system to the initial condition

$$u_0(x) = \begin{cases} 1 & x \leq 0 \\ 0 & x > 0 \end{cases}$$

Let us re-write the above scheme explicitly

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}u_j^n(u_j^n - u_{j-1}^n)$$

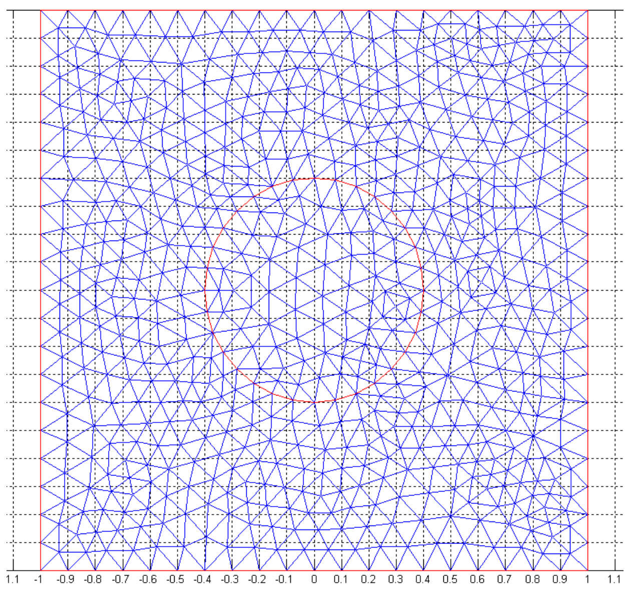

Figure 0.1: The triangular mesh used for the circular dam problem

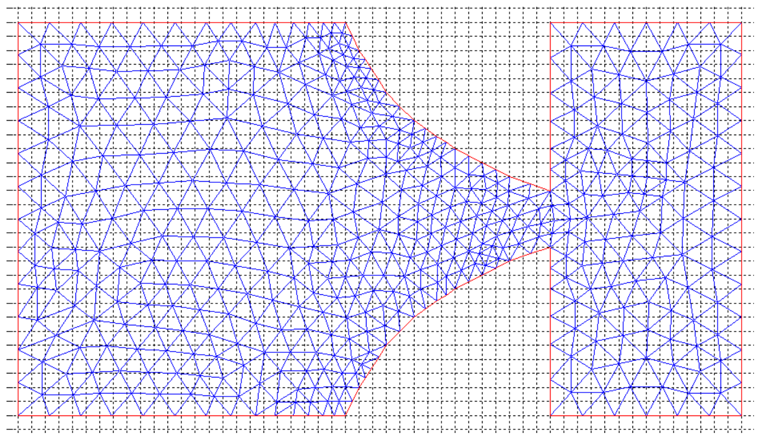

Figure 0.2: The triangular mesh used for the Maeslantkering

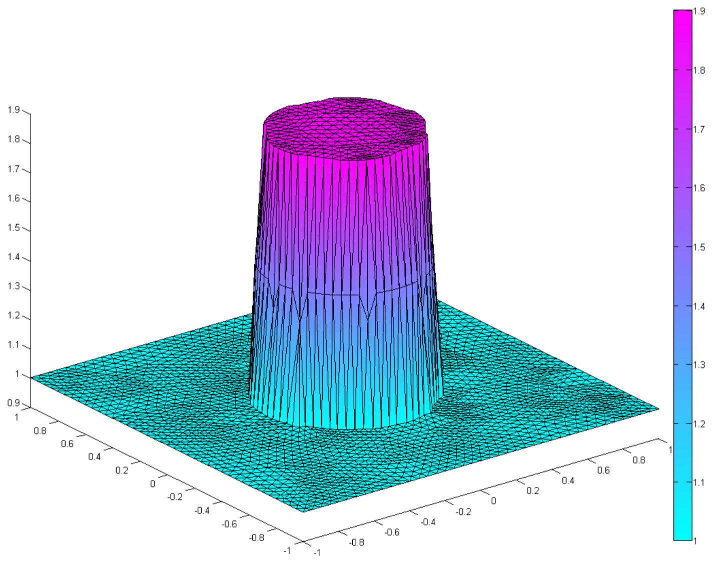

Figure 0.3: Initial condition of the circular dam problem

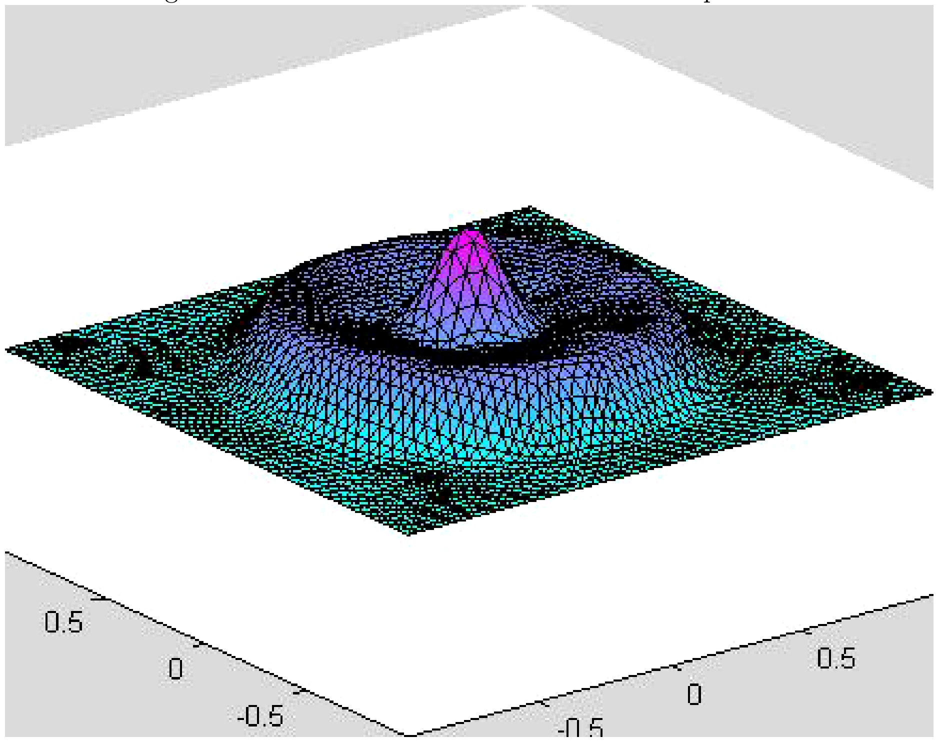

Figure 0.4: The circular dam at 100 timesteps

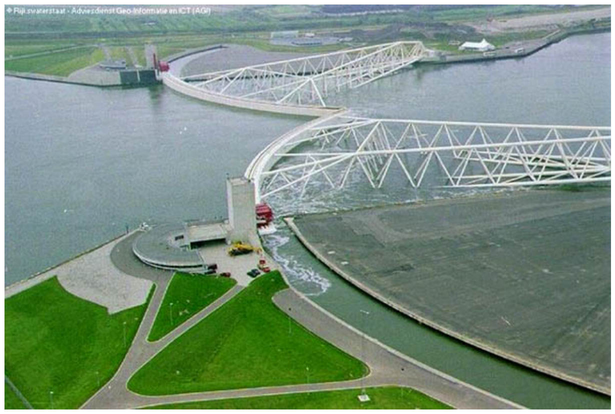
© Rijkswaterstaat - Adviesdienst Geo-informatie en ICT (AGI)

Figure 0.5: The Maeslantkering fully closed along the Nieuwe Waterweg. (Source Rijkswaterstaat)

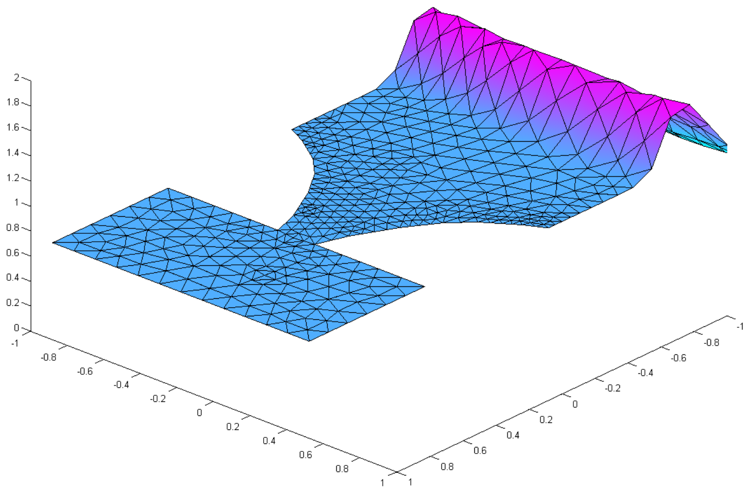

Figure 0.6: The initial wave sent in from the sea towards the Maeslantkering

Consider the first time-step. For $x_j > 0$ we would have that

$$\begin{aligned} u_j^1 &= u_j^0 - \frac{\Delta t}{\Delta x} u_j^0 (u_j^0 - u_{j-1}^0) \\ &= 0 - \frac{\Delta t}{\Delta x} 0 (0 - u_{j-1}^0) \\ &= 0 \end{aligned}$$

For $x_j \leq 0$ we have that

$$\begin{aligned} u_j^1 &= u_j^0 - \frac{\Delta t}{\Delta x} u_j^0 (u_j^0 - u_{j-1}^0) \\ &= 1 - \frac{\Delta t}{\Delta x} (1 - 1) \\ &= 1 \end{aligned}$$

Thus after one iteration, the scheme has not changed and the profile will always stay the same. Clearly, as can be verified by the method of characteristics, there is evolution, and for this specific example, shock waves will form. Thus the limitation of finite difference schemes requires a different scheme is apparent. In this paper we consider equations very similar to Burger's equation. We note that we can re-write the Burger's equation in the following form

$$\frac{\partial u}{\partial t} + \frac{1}{2} \frac{\partial u^2}{\partial x} = 0$$

which is referred to as a conservation law as it can be interpretted as the conservation of the solution.

Shallow Water Waves

The shallow water equations describe the motion of incompressible fluids in situations where the vertical depth of the system is much smaller than the relevant horizontal length scale. Although simplified, the shallow water equations are very powerful and can accurately model the motion of water in a puddle to the entire ocean. The starting point of the derivation is given by the Navier-Stokes equations

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \nu \nabla^2 \mathbf{v} + \mathbf{f}$$

Since the Navier-Stokes equations are a very complicated non-linear set of partial differential equations, we make a few simplifying assumptions. Firstly we assume inviscid flow ($\nu = 0$), a reasonable assumption far away from the boundary. However, since we are modelling a physical system, in actuality there will be boundary layer effects since water is not actually inviscid, but these effects are small and negligible in this approximation. Additionally, any turbulent effects are neglected. We also assume that the geometry is only two dimensional since vertical length scale H, is much less than the horizontal scale L, giving $H \ll L$ (which is valid in the case of a long canal since the depth is about 10-15 m while the length is on the order of hundreds of meters), and thus vertical velocity of the fluid can be neglected since the horizontal velocities will dominate the dynamics (see [Ray] for an elementary derivation). A more rigorous argument can be made by appealing to the orders of magnitude of the various terms in the Navier-Stokes equations [Ach90] [KC04] in which the same result is shown. This approximation forms the basis of the shallow water wave which can be in conservative form with $h(x, y, t)$, the height of the water and $u(x, y, t), v(x, y, t)$ the horizontal

$$\begin{aligned}
\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} &= 0 \\
\frac{\partial (hu)}{\partial t} + \frac{\partial}{\partial x}(hu^2 + gh^2/2) + \frac{\partial (huv)}{\partial y} &= 0 \\
\frac{\partial (hv)}{\partial t} + \frac{\partial (huv)}{\partial x} + \frac{\partial}{\partial y}(hv^2 + gh^2/2) &= 0
\end{aligned} \tag{0.1}$$

on some domain Ω with boundary $\partial\Omega$. It is possible to rewrite this set of equations in a vector form:

$$\frac{\partial \varphi}{\partial t} + \frac{\partial J^x}{\partial x} + \frac{\partial J^y}{\partial y} = 0 \tag{0.2}$$

$$\varphi = \begin{pmatrix} h \\ hu \\ hv \end{pmatrix} \quad J^x = \begin{pmatrix} hu \\ hu^2 + \frac{g}{2}h^2 \\ huv \end{pmatrix} \quad J^y = \begin{pmatrix} hv \\ huv \\ hv^2 + \frac{g}{2}h^2 \end{pmatrix} \tag{0.3}$$

If the “floor” of the system is flat and there is no sourcing term, this equation correctly describes the motion of the fluid if there are no external forces.

Although not discussed in this paper, if we want to examine a system where the bed of the water body is not flat (this situation is certainly more physical) a simple forcing term on the right hand side can be added, see [KC04] [CDMW98] [AC97]

Finite Volume Scheme

Since solving systems described by these equations analytically is infeasible, except in a few very few simple cases [Ach90], a numerical approach is needed. However, as demontrated above, a simple finite difference scheme is not well suited. As can be seen from (0.2) we can write the equation in a well known conservation law form

$$\frac{\partial \varphi}{\partial t} + \nabla \cdot \mathbf{F} = 0 \tag{0.4}$$

These types of equations arise in many places in physics and applied mathematics, especially in fluid mechanics. When written in this form, the conservation law is classified as a hyperbolic problem and is well suited for finite volume techniques [Lev04]. The idea of finite volume method is to break the domain up into cells and describe the changes in a cell over by considering fluxes through the cell boundary. For example, suppose we have cell 1 and cell 2 and we approximate the solution in cell 1 to be constant and the solution in cell 2 to be another, typically different, constant. Depending on the values of the constants the flux through the cell boundary will be different. Consider the simple fluid dynamical example where the flux is simply the velocity of the water. If the velocity of the water in cell 1 is greater then cell 2, the water will want to flow to the right. If it was greater in cell 2, the water would want to flow to the left. For more complicated situations, there can be more possible combinations.

Collectively, the problem of solving a PDE with a constant solution with a single discontinuity is known as a Riemann problem. The example given in the previous section is a very simple example of a Riemann problem. A more complete discussion can be found in [Lev04].

There are many choices for how to calculate this flux for a given Riemann problem. In particular we want to investigate the shallow water equations in situations that lend themselves well to description by triangular tilings. One potential way of determining the flux through a cell boundary is to simply take the average. However, if we do this, and write out the resulting scheme, we get an equation that is similar to the one derived above for finite difference which fails to capture the relevant important evolution behaviour.

Instead, we modify the average by adding a correction term that involves the normal component of the flux through a cell which is effectively a viscous correction. This solution to the Riemann problem, is a variation of the Godunov scheme called a Roe solver. For a full derivation for the 1D shallow water equations, see [Lev04].

To derive a finite volume scheme, we first transform the conservation law form of the shallow wave equations into a form

$$\frac{\partial}{\partial t}\int_{\Omega}\varphi d\Omega + \oint_{\partial\Omega} dS\mathbf{J}\cdot\hat{\mathbf{n}} = 0 \qquad \mathbf{J}\cdot\hat{\mathbf{n}} = J^x n_x + J^y n_y$$

which is obtained by integrating over the domain and applying the divergence theorem. To proceed, we now break the integrals into components over each of the cell domains

$$\sum_i \frac{\partial}{\partial t}\int_{\Omega_i}\varphi_i d\Omega_i + \oint_{\partial\Omega_i} dS_i\mathbf{J}_i\cdot\hat{\mathbf{n}}_i = 0$$

where we are now integrating each of the functions over a cell labelled by i. However, we now make the approximation that in each of the cells, the variables are constant. In other words we make the approximation that

$$\int_{\Omega_i}\varphi_i d\Omega_i \approx \varphi_i d\Omega_i$$

$$\oint_{\partial\Omega_i}\mathbf{J}_i\cdot\hat{\mathbf{n}}_i dS \approx \sum_{j=k(i)} J_{i,j}\Delta l_j \qquad J_{i,j} = J_i^x n_j^x + J_i^y n_j^y$$

where Δl_j labels the length of the boundary and $k(i)$ is a list of the edges of the cell. Thus we now have an equation for each cell i of the form

$$\frac{\partial\varphi_i}{\partial t} = \frac{1}{\Delta\Omega_i}\sum_{j=k(i)} J_{i,j}\Delta l_j$$

We now make a simple finite difference approximation to the time derivative to obtain the scheme

$$\varphi_i^{n+1} = \varphi_i^n - \frac{\Delta t}{\Delta\Omega_i}\sum_{j=k(i)} J_{i,j}\Delta l_j \tag{0.5}$$

This is the finite volume scheme we will be applying in this paper. So far, we have left the geometry of the cells unspecified but we will now assume a triangular mesh. It is now important to note how this differs from the simpler rectangular finite volume meshes. For rectangular finite volume meshes this expression greatly simplifies since the area of each cell is identical as are the lengths. With a triangular grid, this is no longer true. Additionally, for a rectangular grid the normal vectors are very simple and one component of the flux $J_{i,j}$ will cancel out, however in a triangular grid, the normal vector varies from triangle to triangle and even each side of the triangle. While not more conceptually difficult, this procedure means that much more bookkeeping must be done.

Triangular Meshes

In order to implement triangular meshes we used the toolbox `pdetool` in MATLAB. This toolbox allows for the automatic creation triangular meshes when given a certain geometry. For the two problems we are considering, we have two different geometries given by Figures 1 and 2.

In order to translate to be implemented in the finite volume scheme we use the command `initmesh` which allows the creation of three matrices that encode all the information about the geometry. Using these three matrices we are able to encode all the vital information about the mesh. Unfortunately MATLAB does not order the triangles in any particular order so we had to implement a method of ordering the triangles. This was achieved by using a simple search over all the triangles and matching triangle vertices. Special care had to be paid for the boundary edges as MATLAB does not keep track of whether the triangle is a boundary point or not. From this bookkeeping, the normal vectors, lengths, and areas of the triangles were calculated and stored. For more information about how this is done specifically, see the Appendix.

The Riemann Problem

In the finite volume class of methods, finding the value of the fluxes at the interface is of primary importance, as this allows us to advance the system in time. For systems of nontrivial complexity, determining this value exactly is very difficult or impossible. A variety of approximation techniques have been developed to allow efficient calculation of the solution to the Riemann problem. One of these is the Roe solver, developed by Philip Roe. The Roe solver linearises the Jacobian of the fluxes over the normal vector. This allows us to calculate the flux at the interface relatively easily while still preserving features such as shocks. We write, following [AC97]

$$J_{i,j} = \frac{1}{2}\left[J(\varphi^+_{i,j}) + J(\varphi^-_{i,j}) - |A|(\varphi^+_{i,j} - \varphi^-_{i,j})\right] \tag{0.6}$$

where i, j refer to the i-th cell's j-th interface, φ^+ refers to the values of φ in the current cell, and φ^- refers to the values of φ in the adjacent cell being considered.

$$\begin{aligned} A =& \frac{\partial(\mathbf{J}\cdot\mathbf{n})}{\partial\varphi} & (0.7)\\ =& \begin{bmatrix} 0 & \mathbf{n}\cdot\hat{\mathbf{x}} & \mathbf{n}\cdot\hat{\mathbf{y}} \\ (gh-u^2)\mathbf{n}\cdot\hat{\mathbf{x}} - uv\mathbf{n}\cdot\hat{\mathbf{y}} & 2u\mathbf{n}\cdot\hat{\mathbf{x}} + v\mathbf{n}\cdot\hat{\mathbf{y}} & u\mathbf{n}\cdot\hat{\mathbf{y}} \\ (gh-v^2)\mathbf{n}\cdot\hat{\mathbf{y}} - uv\mathbf{n}\cdot\hat{\mathbf{x}} & v\mathbf{n}\cdot\hat{\mathbf{x}} & u\mathbf{n}\cdot\hat{\mathbf{x}} + 2v\mathbf{n}\cdot\hat{\mathbf{y}} \end{bmatrix} & (0.8) \end{aligned}$$

As we can see, we compute the average flux with a viscous correction of the form $-|A|(\varphi^+_{i,j} - \varphi^-_{i,j})$. We can split diagonalize this matrix to find $|A|$:

$$\begin{aligned} |A| =& R|\Lambda|L & (0.9)\\ \Lambda =& \begin{bmatrix} u\mathbf{n}\cdot\hat{\mathbf{x}}) + v\mathbf{n}\cdot\hat{\mathbf{y}} & 0 & 0 \\ 0 & u\mathbf{n}\cdot\hat{\mathbf{x}} + v\mathbf{n}\cdot\hat{\mathbf{y}} - \sqrt{gh} & 0 \\ 0 & 0 & u\mathbf{n}\cdot\hat{\mathbf{x}} + v\mathbf{n}\cdot\hat{\mathbf{y}} + \sqrt{gh} \end{bmatrix} & (0.10)\\ R =& \begin{bmatrix} 0 & 1 & 1 \\ \mathbf{n}\cdot\hat{\mathbf{y}} & u - \sqrt{gh}\mathbf{n}\cdot\hat{\mathbf{x}} & u + \sqrt{gh}\mathbf{n}\cdot\hat{\mathbf{x}} \\ -\mathbf{n}\cdot\hat{\mathbf{x}} & v - \sqrt{gh}\mathbf{n}\cdot\hat{\mathbf{y}} & v + \sqrt{gh}\mathbf{n}\cdot\hat{\mathbf{y}} \end{bmatrix} & (0.11)\\ L =& \begin{bmatrix} -(u\mathbf{n}\cdot\hat{\mathbf{y}} - v\mathbf{n}\cdot\hat{\mathbf{x}}) & \mathbf{n}\cdot\hat{\mathbf{y}} & -\mathbf{n}\cdot\hat{\mathbf{x}} \\ (u\mathbf{n}\cdot\hat{\mathbf{x}} + v\mathbf{n}\cdot\hat{\mathbf{y}})/2\sqrt{gh} + \frac{1}{2} & -\mathbf{n}\cdot\hat{\mathbf{x}}/2\sqrt{gh} & -\mathbf{n}\cdot\hat{\mathbf{y}}/2\sqrt{gh} \\ -(u\mathbf{n}\cdot\hat{\mathbf{x}} + v\mathbf{n}\cdot\hat{\mathbf{y}})/2\sqrt{gh} + \frac{1}{2} & \mathbf{n}\cdot\hat{\mathbf{x}}/2\sqrt{gh} & \mathbf{n}\cdot\hat{\mathbf{y}}/2\sqrt{gh} \end{bmatrix} & (0.12) \end{aligned}$$

Where R is the right eigenvector matrix and L is the left eigenvector matrix.

Applications

We now have all the tools needed to solve the shallow water equations using a finite volume scheme. Using (0.5) we apply the Roe solver to $J_{i,j}$ and compute over each of the three cell sides.

To demonstrate a simple application of the scheme we consider the classical problem of a circular dam using the mesh given by Figure 1. We now need to apply the proper boundary and initial conditions. Since we are dealing with an inviscid flow we have the no-slip boundary condition $\mathbf{u} \cdot \hat{n} = 0$ at the boundary. For the simple case of a rectangular box these conditions are obtained by choosing an appropriate flux through the boundary. This is given by

$$h_- = h_+ \qquad (uh)_- = \pm(uh)_+ \qquad (vh)_- = \mp(vh)_+$$

where the minus sign is chosen when the uh or vh are perpendicular to the wall. For initial condition, we simply chose an initial height of $h = 1.9$, see Figure 2. The simulation was run on three settings with varying numbers of triangles for 1000 timesteps. The plotted figures demonstrate the most refined mesh used which has roughly 5000 triangles. As can be seen in Figure 4, the triangular mesh is able to deal with the circular geometry very easily and no loss of the flux is obtained due to edge effects. For a video of the dam see [BH12a]

Rotterdam

For this scenario, we want to examine the behaviour of the Nieuwe Waterweg in Holland. This is a shipping canal created to allow passage of ships from the North Sea to the Europoort in Rotterdam, which is one of the world's busiest ports. The Netherlands is at great risk of flooding from the North Sea, and a surge down the Nieuwe Waterweg would prove disasterous to Rotterdam and the surrounding region. As part of Holland's Delta Works plan to construct flood barriers, dams, and surge protectors, a movable barrier was constructed in the Nieuwe Waterweg - the Maeslantkering (see Figure). This barrier sits in drydock most of the time, but when a surge is imminent its halves will swing out to save Rotterdam. The arms of the barrier take about 2 hours to close, and begin closing if the North Sea is likely to generate surges of 3 metres or more. First the drydocks are flooded and the wedges float out into the water and begin to move towards each other. After the gates have closed, they are filled with water (causing them to submerge) and then function to block the surges. In this simulation, we want to investigate the situation in which the Maeslantkering is hit with a wave while being hit from the sea. As opposed to the image of the Maeslantkering where it is fully opened, we consider ours midway closed to demonstrate the effects of how the structure breaks the waves. Additionally, we have simplified the geometry of the structure for modelling purposes.

0.0.1 Implementation

There are three boundaries we need to account for: the open water at the ends of the channel, the banks of the river, and the surge gate [CDMW98]. For the open sea, we fix u_B, which is the horizontal velocity coming from the North Sea. We are simulating a storm surge, so enforcing that the flow from the sea is always towards Rotterdam is reasonable. We also assume that the flow from the sea is always subcritical. In the field of fluid mechanics, a subcritical flow exists when the flow velocity is less than the wave velocity. Supercritical flows have the opposite property. A supercritical flow is analogous to a supersonic wave in air - we assume that a similar condition does not exist at the end of the waterway. At the other end, we assume that there is no flow from Rotterdam. This makes sense since the continent lies in that direction - the flows from the Rhine-Meuse-Scheldt delta are relatively small compared to those from the North Sea.

On triangles with edges facing the open sea, we fix the value for $\mathbf{u_r}$ as the boundary u_B, force $v_R = 0$ (we assume no perpendicular flow at the interface), and solve for h using:

$$u_r = u_L - \sqrt{g}(\sqrt{h_r} - \sqrt{h_l}) \tag{0.13}$$

We send the boundary condition to the Riemann solver by passing it as part of `adj_tri_info`, the matrix which contains the information about the triangles the current one interfaces with.

For the riverbank, we assume that the interface causes perfect reflection of the vertical flow and does not affect the horizontal flow (there are no eddy currents). Since we only examine inviscid flow, this is somewhat reasonable - in this case, there would be no boundary layer to affect the horizontal flows near the edge. However, it is not a very physical assumption. Water is not an ideal liquid and isn't inviscid, so in the Nieuwe Waterweg there will be boundary layers which our simulation doesn't take into account.

On the gate, we assume that if the height of the water is less than the height of the gate, then the gate also reflects the flows perfectly. However, we face an additional complication here - the gate faces are neither perfectly horizontal nor perfectly vertical. In order to find the resulting fluxes from the reflection, some algebra is necessary. Let $\mathbf{u}_r$ and $\mathbf{v}_r$ be the reflected flows at the interface with the gate. Since perfect reflection occurs, $|\mathbf{u}_r| = |\mathbf{u}|$ and $|\mathbf{v}_r| = |\mathbf{v}|$. We also specify that θ is the angle between the $\hat{x}$ direction and the normal vector at the interface with the gate.

$$\begin{aligned}
\mathbf{u}\cdot\mathbf{u}_r =& u^2\cos 2\theta & \mathbf{v}\cdot\mathbf{v}_r =& v^2\cos(2\theta-\frac{\pi}{2}) = v^2\sin 2\theta\\
u\cdot -u_{r,x} =& u^2\cos 2\theta & v\cdot -v_{r,y} =& u^2\sin 2\theta\\
u_{r,x} =& -u\cos 2\theta & v_{r,y} =& -v\sin 2\theta\\
u =& \sqrt{u_{r,x}^2+u_{r,y}^2} & v =& \sqrt{v_{r,x}^2+v_{r,y}^2}\\
\Rightarrow u_{r,y} =& u\sin\theta & \Rightarrow v_{r,x} =& v\cos\theta
\end{aligned}$$

So that the interface fluxes u_R and v_R are:

$$u_R = -u\cos 2\theta - v\sin 2\theta \tag{0.14}$$

$$v_R = u\sin 2\theta + v\cos 2\theta \tag{0.15}$$

Where $\theta = \arctan(n_y/n_x)$.

When sending the wave down the waterway, we simulate a square wave by dropping the height of the sea after a few timesteps. Although the sea doesn't produce square waves, the wave "spreads out" due to our Riemann solver, creating a reasonable facsimile of an ocean wave. See Figure 6 for the wave after 15 timesteps

Results

For a video of the results, again see [BH12a]. Of particular interest is the reflective behaviour of the gates and the extremely dampened wave that makes it though the gap between them - the Maeslantkering seems to be an effective surge barrier, provided the gates themselves are not overwhelmed by a very tall wave.

Conclusion

We have implemented finite volume using triangular meshes with a Roe solver to obtain the evolution of a fluid in two geometries. The implementation provides a very robust numerical scheme that can be easily applied to many non-trivial geometries that simple rectangular finite volume methods are not able to handle well. Future work that can be done is to implement the shallow water wave equations with viscous effects as done in [AC97]. Additionally, we've neglected sea geometry and other force effects that might be present. All the code run, along with documentation can be found at [BH12b].

Appendix

In this section we describe how the bookkeeping methods operate. We have made an effort to get the code to run in the open source version of MATLAB, Octave, however Octave did not have the available tools that MATLAB has and we were unable to get it to run properly.

Creating the triangular mesh is done by using the MATLAB toolbox `pdetools` which, unfortunately, does not come standard with the student edition of MATLAB. In order to get around this, the matrices are provided. However, it will not be possible to experiment with other geometries without such toolboxes. Additionally, the way MATLAB does bookkeeping of the triangles is not very intuitive and much bookkeeping must be done before the numerical model is applied.

Initially we export the geometry in terms of three matrices, `p,e,t`. The `p` matrix contains the co-ordinates of the verticies of each the triangle. The matrix `t` contains the vertex labels of each the triangles in a counter-clockwise order. Thus in order to find the co-ordinates of a given triangle, one would look at the first three entries of `t` which tell us which entries of `p` to look at. For example, if we are considering the 15th triangle, the co-ordinates of the verticies are given by `p(:,t(1,15)),p(:,t(2,15)),p(:,t(3:15))`. Unfortunately, MATLAB does not order the triangles in any particular order, i.e. triangle 15 is not next to triangle 16 in `t`. Thus we must search through all the triangle verticies in order to determine which triangles are beside each other. This is done by a brute force search using the function `edgefind`. The way this function works is by simply taking the ith triangle and checking the vertices of one edge of the triangle with all the others. If the triangle is a boundary, this is handled approptiately. The resulting matrix `EdgeMatrix` tells us that if we have, say the ith triangle, it shares boundaries with triangles a,b,c. However the labelling is no particular order and it is now sorted via `order_triangles_b` (for triangles that lie on the boundary) and `order_trinagles_nb` otherwise. Thus we now have that in `EdgeMatrix` for the ith triangle, the normal vector of the first edge shares the boundary with first triangle in `EdgeMatrix` and so on. Finally we must assign, properly, the boundaries. This is contained in the matrix `e`. This matrix contains only the boundaries and the associated boundary number assigned to it by `pdetools`. Throughout we use this matrix to check whether or not the triangle is on the boundary. For more information on how MATLAB maintains all the triangle information see the help file.

The main parts of the programme are `TriInfo`,`TriData`. We have that

```
TriInfo(1) = Triangle Index
TriInfo(2,3) = components of n1 vector
TriInfo(4,5) = components of n2 vector
TriInfo(6,7) = components of n3 vector
TriInfo(8,) = whether the triangle considered is on the boundary
TriInfo(9,10,11) = lengths of n1,n2,n3
TriInfo(12) = area of triangle
```

which contains all the information about each triangle while `TriInfo` contains

```
TriData(1) = Triangle index
TriData(2) = h
TriData(3) = uh
TriData(4) = vh
```

which corresponds to φ.

References

[AC97] K. Anastasiou and C.T. Chan, *Solution of the 2d shallow water equations using the finite volume method on unstructured triangular meshes*, Int. J. Numer. Meth. Fluids **24** (1997), 1225–1245.

[Ach90] D.J. Acheson, *Elementary fluid dynamics*, Oxford Applied Mathematics and Computing Series, 1990.

[BH12a] L. Bovard and K.S. Hyatt, 2012.

[BH12b] ______, 2012.

[CDMW98] S. Chippada, C.N. Dawson, M.L. Martinez, and M.F. Wheeler, *A Godunov-type finite volume method for the system of shallow water equations*, Computer Methods in Applied Mechanics and Engineering **151** (1998).

[KC04] P. Kundu and I. Cohen, *Fluid mechanics*, Elsevier Academic Press, 2004.

[Lev04] R.J. Leveque, *Finite-volume methods for hyperbolic problems*, Cambridge University Press, 2004.

[Ray] D. Raymond.