Chapter 11

Shallow Water Equations

Nils Thuerey, Peter Hess

11.1 Introduction

The *shallow water equations* (SWE) are a simplified version of the more general *Navier-Stokes* (NS) equations, which are commonly used to describe the motion of fluids. The SWE reduce the problem of a three-dimensional fluid motion to a two-dimensional description with a height-field representation. From now on, we will use the following notation (it is also illustrated in Fig. 11.1):

- h denotes the height of the fluid above zero-level.
- g is the height of the ground below the fluid (above zero-level).
- η denotes the height of the fluid above ground, $\eta = h - g$.
- $\mathbf{v}$ the velocity of the fluid in the horizontal plane.

A basic version of the SWE can be written as

$$\frac{\partial \eta}{\partial t} + (\nabla \eta)\mathbf{v} = -\eta \nabla \cdot \mathbf{v} \tag{11.1}$$

$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla \mathbf{v})\mathbf{v} = a_n \nabla h \ , \tag{11.2}$$

where a_n denotes a vertical acceleration of the fluid, e.g., due to gravity. This formulation can be derived from the NS equations by, most importantly, assuming a hydrostatic pressure along the direction of gravity. Interested readers can find a detailed derivation of these euqations in Section A.

In the following sections we will first explain how to solve these equations with a basic solver, and then extend this solver with more advanced techniques to handle open boundaries, or free surfaces.

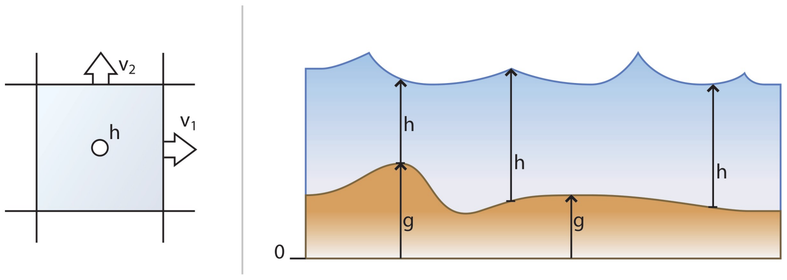

Figure 11.1: A fluid volume is represented as a heightfield elevation in normal direction **n**. The fluid velocity **v** has two components in horizontal directions.

11.2 A basic Solver

A basic solver of the SWE has to compute the change of the height and velocity values of the water surface over time. According to Eq. (11.2) and Eq. (11.1) the equations for the water height η and the velocity $\mathbf{v} = (v_1, v_2)$ can be written as:

$$\begin{aligned} \partial\eta/\partial t + (\nabla\eta)\mathbf{v} &= -\eta\nabla\cdot\mathbf{v} \\ \partial v_1/\partial t + (\nabla v_1)\mathbf{v} &= a_n\nabla h \\ \partial v_2/\partial t + (\nabla v_2)\mathbf{v} &= a_n\nabla h\,. \end{aligned} \tag{11.3}$$

In this form, the two distinct parts of the equations can be identified: the left side accounts for the advection within the velocity field **v**, while the right side computes an additional acceleration term. Here, we will use an explicit time integration scheme, as this makes the solution of the SWE signficantly more simple. Alternatively, implicit schemes, as described in [LvdP02] could be used. These, however, require a system of linear equations to be solved for the update, and, with simpler methods such as implicit Euler, introduce a significant amount of damping.

To compute a solution for these equations, we first discretize the domain with n_1 cells in x-direction, and n_2 cells in y-direction. For simplicity, we assume in the following that the cells have a square form with side length Δx. The gravity force a_n is assumed to act along the z-axis, e.g. $a_n = 10$, and the size of a single time step is given by Δt. The overall height of the water, and the strength of the gravity will later on determine the speed of the waves travelling on the surface. To represent the three unknowns with this grid we use a staggered grid. This means that the pressure is located in the center of a cell, while the velocity components are located at the center of each edge, as shown in Fig. 11.1. The staggered grid is commonly used for fluid solvers, and prevents instabilities that would result from a discretization on a co-located grid. An update step of the shallow water solver consisits of the following parts: first all three fields are advected with the current velocity field. Afterwards, the acceleration terms are computed for the height and velocity fields. The following pseudo-code illustrates a single step of the simulation loop of a simple shallow water solver:

```
Shallow-water-step($\eta, \mathbf{v}, g$)
(1) $\eta = \text{Advect}(\eta, \mathbf{v})$
(2) $v_1 = \text{Advect}(v_1, \mathbf{v})$
(3) $v_2 = \text{Advect}(v_2, \mathbf{v})$
(4) Update-height($\eta, \mathbf{v}$)
(5) $h = \eta' + g$
(6) Update-velocities($h, v_1, v_2$)
```

Note that the three calls of the *Advect*(...) function return a value that is assigned back to the original input grid (e.g., in line 1 η is a parameter of the call, and used in the assignment). This should indicate that the advection requires a temporary array, to which the advected values are written, and which is copied back to the original grid after finishing the advection step.

To compute the advection, we can use the semi-Lagrangian method [Sta99] to compute a solution without having to worry about stability. This algorithm computes the advection on a grid by essentially performing a backward trace of an imaginary particle at each grid location. Given a scalar field s to be advected, we have to compute a new value for a grid cell at position $\mathbf{x}$. This is done by tracing a particle at this position backward in time, where it had the position $\mathbf{x}^{t-1} = \mathbf{x} - \Delta t \mathbf{v}(\mathbf{x})$. We now update the value of s with the value at $\mathbf{x}^{t-1}$, so the new value is given by $s(\mathbf{x})' = s(\mathbf{x}^{t-1})$. Note that although $\mathbf{x}$ is either the center or edge of a cell in our case, $\mathbf{x}'$ can be located anywhere in the grid, and thus usually requires an interpolation to compute the value of s there. This is typically done with a bi-linear interpolation, to ensure stability. It guarantees that the interpolated value is bounded by its source values from the grid, while any form of higher order interpolation could result in larger or smaller values, and thus cause stability problems. The advection step can be formulated as

```
Advect($s, \mathbf{v}$)
(1) for $j = 1$ to $n_2 - 1$
(2)     for $i = 1$ to $n_1 - 1$
(3)         $\mathbf{x} = (i \cdot \Delta x, j \cdot \Delta x)$
(4)         $\mathbf{x}' = \mathbf{x} - \Delta t \cdot v(\mathbf{x})$
(5)         $s'(i,j) = \text{interpolate}(s, \mathbf{x}')$
(6)     endfor
(7) endfor
(8) return($s'$)
```

Note that, due to the staggered grid, the lookup of $\mathbf{v}(\mathbf{x})$ above already might require an averaging of two neighboring velocity components to compute the velocity at the desired position. This is also, why the three advection steps cannot directly performed together - each of them requires slightly different velocity interpolations, and leads to different offsets in the grid for interpolation.

The divergence of the velocity field for the fluid height update can be easily computed with finite differences on the staggered grid. So, according to Eq. (11.3), the height update is given by

```
Update-height($\eta, \mathbf{v}$)
(1)  for $j = 1$ to $n_2 - 1$
(2)      for $i = 1$ to $n_1 - 1$
(3)          $\eta(i,j) -= \eta(i,j) \cdot \left( \frac{(v_1(i+1,j) - v_1(i,j))}{\Delta x} + \frac{(v_2(i,j+1) - v_2(i,j))}{\Delta x} \right) \Delta t$
(5)      endfor
(6)  endfor
(7)  return($\eta'$)
```

In contrast to the advection steps, adding the accelerations can be directly done on the input grids. Similarly, the acceleration term for the velocity update is given by the gradient of the overall fluid height. Note that in this case, the total height above the zero-level is used instead of the fluid height above the ground level. This is necessary, to, e.g., induce an acceleration of the fluid on an inclined plane, even when the fluid height itself is constant (all derivatives of η would be zero in this case). The parameter a for the velocity update below is the gravity force.

```
Update-velocities($h, v_1, v_2, a$)
(1)  for $j = 1$ to $n_2 - 1$
(2)      for $i = 2$ to $n_1 - 1$
(3)          $v_1(i,j) += a(\frac{h(i-1,j) - h(i,j)}{\Delta x})\Delta t$
(5)      endfor
(6)  endfor
(7)  for $j = 2$ to $n_2 - 1$
(8)      for $i = 1$ to $n_1 - 1$
(9)          $v_2(i,j) += a(\frac{h(i,j-1) - h(i,j)}{\Delta x})\Delta t$
(10)     endfor
(11) endfor
```

This concludes a single step of a basic shallow water solver. Note that the steps so far do not update the values at the boundary of the simulation domain, as we cannot compute any derivatives there. Instead, special boundary conditions are required at the border, and can be used to achieve a variety of effects. These will be the topic of the next section.

11.3 Boundary Conditions

In the following, we will describe different types of boundary conditions: reflecting and absorsorbing boundaries, as well as a form of free surface boundary conditions. The former can be used to model a wall that reflects incoming waves. The second type can be used to give the effect of an open water surface, as waves will simply leave the computational domain. Free surface boundary conditions can be used once the fluid should, e.g., flow through a landscape. Although the boundary conditions will be described to handle the outermost region of the computational domain, they can likewise be used to, e.g., create a

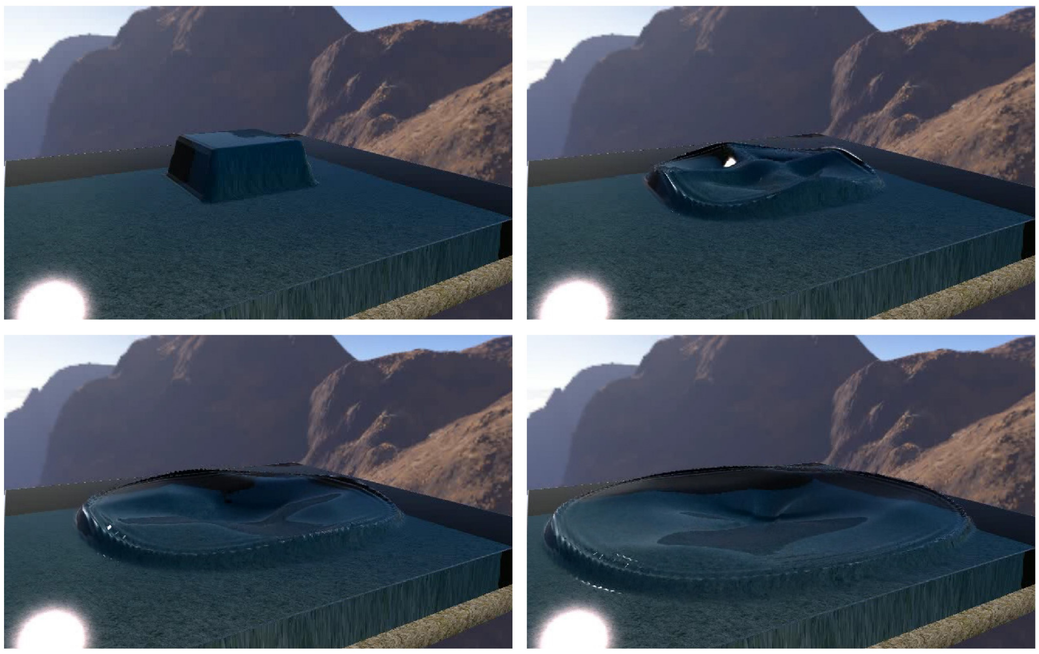
Figure 11.2: Example of a wave spreading in a basic shallow water simulation.

wall in the middle of the domain. We will not consider periodic boundary conditions here. They are commonly used in engineering applications, but give the visually unnatural effect that a wave leaving the domain at the right side re-enters it at the left. The results shown in this section where created by Peter Hess [Hes07].

Reflecting Boundaries

In the following, we will, without loss of generality, consider the boundary conditions for cells at the left boundary. Reflecting boundary conditions are achieved by setting the velocities at the boundary to zero (after all, there should be no flux through the wall). In addition, we mirror the height of the fluid in the outermost layer. We thus set:

$$\begin{aligned} h(0,j)' &= h(1,j) \\ v_1(1,j)' &= 0 \\ v_2(0,j)' &= 0\,. \end{aligned} \tag{11.4}$$

Note that we do not modify the y-component v_2 of the velocity field. The fluid is thus allowed to move tangentially to a wall. Theoretically, we could also enforce different behaviors for the tangential velocities, but in practice this does not make a noticeable difference. Also note, that we only set $v_1(1,j)$, as $v_1(0,j)$ is usually never accessed during an computation step.

Absorbing Boundaries

Surprisingly, it is more difficult to achieve absorbing boundaries than reflecting ones. The problem of boundaries simulating an infinite domain is already known for a long time (see, e.g., [Dur01] for details). A commonly used method to achieve this, is the *perfectly matched layer* introduced by [Ber94], requires an additional layer of computations around the actual domain.

This is why we chose to use the Higdon boundary conditions [Hig94] which are less accurate but can be more efficiently computed than PML. Below is the pth order Higdon boundary condition, where the velocities c_j are chosen to span the range of incoming wave velocities.

$$\left(\prod_{j=1}^{p}\left(\frac{\partial}{\partial t}+c_j\frac{\partial}{\partial x}\right)\right)h=0 \tag{11.5}$$

This boundary condition can be problematic for higher order approximations, but as the wave propagation speed in shallow water is known to be $c=\sqrt{g\eta}$, this allows us to use to use the 1st order boundary condition

$$\left(\frac{\partial}{\partial t}+c\frac{\partial}{\partial x}\right)h=0\,. \tag{11.6}$$

This boundary condition actually requires temporal derivatives, so we assume the current heightfield is given by h^t, while the heights of the previous step are stored in h^{t-1}. Hence, we can set the boundary values to:

$$\begin{aligned}
h(0,j)' &= \frac{\Delta x\, h(0,j)^{t-1}+\Delta t\ c(1,j)^t h(1,j)^t}{\Delta x+\Delta t\ c(1,j)^t}\\
v_1(1,j)' &= v_1(1,j)^{t-1}-a\frac{h(1,j)^t-h(0,j)^t}{\Delta x}\Delta t\\
v_2(1,j)' &= 0
\end{aligned} \tag{11.7}$$

Note that the update of v_1 is essentially the same acceleration term on the left hand side of Eq. (A.16). To further suppress any residual reflections at the boundary, we can apply a slight damping of the height field in a layer around the boundary.

Fig. 11.3 shows the effect of these boundary conditions compared to reflecting ones.

For boundaries where fluid should flow into or out of the domain, we can reuse the two types above. Inflow boundary conditions can be achieved by specifiying reflecting ones, with an additional fixed normal velocity. For outflow boundary conditions, absorbing ones with free normal velocities are more suitable.

Free Surfaces

Often, shallow water simulations assume a completely fluid domain, since this makes solving the SWE quite straightforward. Once applications like a river, or fluid filling an arbitrary terrain are needed, this is not sufficient anymore. Such applications require a distinction between areas filled with fluid, and empty or dry areas. An example can be seen in Fig. 11.4. In the following we will consider this as a problem similar to free surface handling for full fluid simulations. Shallow water simulations naturally have an interface, and thus a free

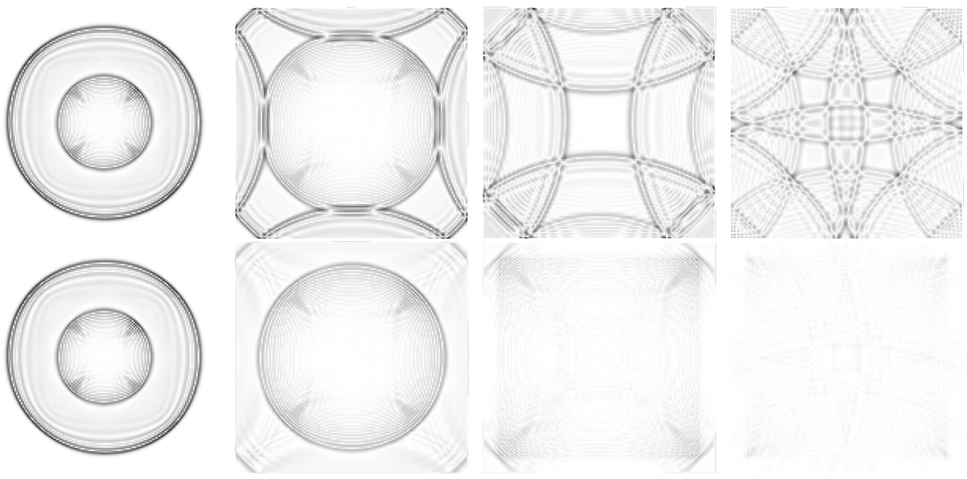
Figure 11.3: A comparison between reflecting boundary conditions (upper row of pictures), and absorbing ones (lower row).

surface, in the simulation plane between fluid below and a second gas phase above the fluid. In addition, we will now prescribe boundary conditions with such a free surface within the simulation plane itself. From the mathematical point of view, a distinction between fluid and dry would not be necessary since the SWE still work if η is zero. Distinguishing fluid and dry cells, however, brings some advantages. Foremost, computational time can be saved if large parts of the domain are dry. Therefore, we introduce cell flags $f(i,j)$, that determine the type of each cell, and update them once per time step after updating the heights. This allows us to quickly identify wet and dry cells. Besides the computational advantage, controlling the transition between wet and dry cells also gives us some control over the spreading velocity. Without free surface tracking, the fluid boundary would expand exactly one cell per time step, regardless of cell size and time step length. The height of this advancing boundary would be very small, but this behavior is usually not desired. In addition, we will compute a fill value r for each cell, as this allows us to track a smoothly moving surface line between the fluid and empty cells.

To determine the cell's flag f we have to compute the minimal and maximal ground level h_{min} and h_{max} as well as the maximal fluid depth η_{max} on the cell's edges.

$$h_{min}(i,j) = \frac{h(i,j) + \min\ h(\mathbf{p})}{2} \qquad \mathbf{p} \in \mathcal{N}(i,j) \tag{11.8}$$

$$h_{max}(i,j) = \frac{h(i,j) + \max\ h(\mathbf{p})}{2} + \varepsilon_H \qquad \mathbf{p} \in \mathcal{N}(i,j) \tag{11.9}$$

$$\eta_{max}(i,j) = \frac{\eta(i,j) + \max\ \eta(\mathbf{p})}{2} \qquad \mathbf{p} \in \mathcal{N}(i,j) \tag{11.10}$$

where $\mathcal{N}(i,j)$ is the set of the four direct neighbors of cell (i,j). Note that we add a small value ε_h to h_{max} to prevent h_{min} to be equal to h_{max} in flat areas. With these three values we can now determine f as well as the fill ratio r which indicates the cell's fill level in dependence of the local ground topology $h_{min}(i,j)$ and $h_{max}(i,j)$. r can be used to compute

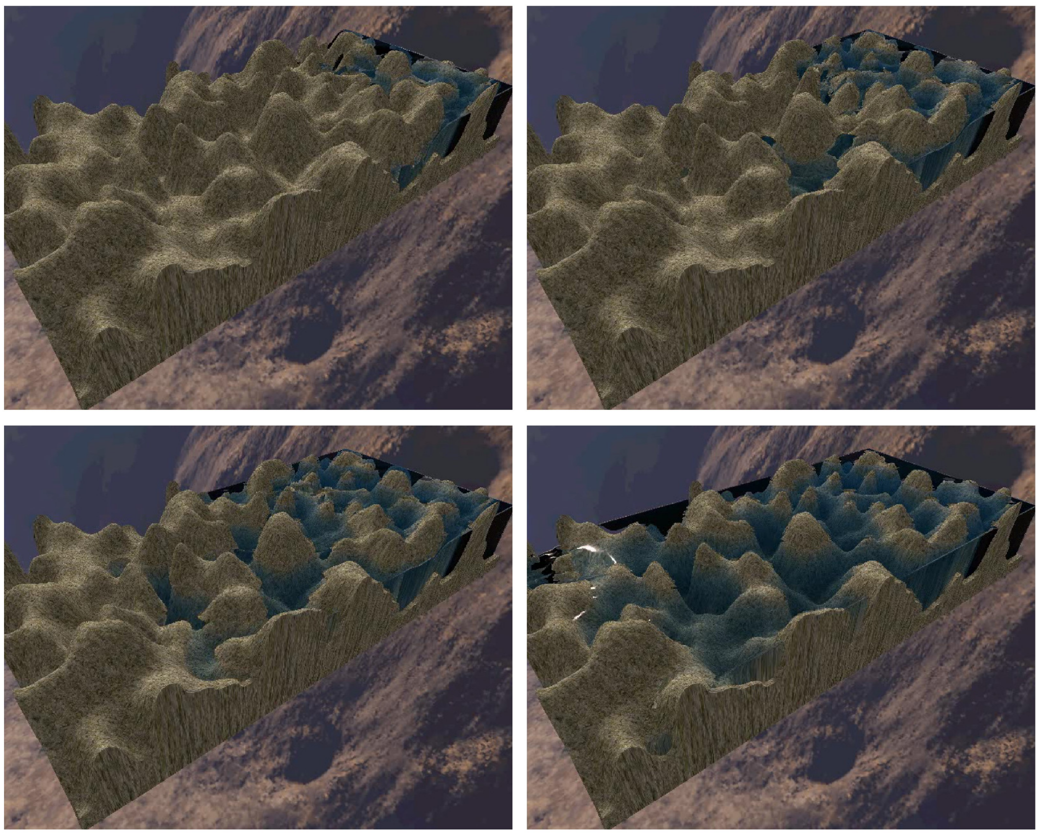
Figure 11.4: A shallow water simulation with free surface boundary conditions fills a terrain.

an isoline which defines the border of the rendered fluid surface for rendering the water surface. The following pseudo code shows how f and r are calculated:

```
Compute-flags$(i,j)$
(1)  if $h(i,j) \le h_{min}(i,j)$ and $\eta_{max}(i,j) < \varepsilon_{\eta_{max}}$
(2)      $f(i,j) = DRY$
(3)      $r(i,j) = 0$
(4)  else if $h(i,j) > h_{max}$
(5)      $f(i,j) = FLUID$
(6)      $r(i,j) = 1$
(7)  else
(8)      $f(i,j) = FLUID$
(9)      $r(i,j) = \Big(h(i,j) - h_{min}(i,j)\Big) / \Big(h_{max}(i,j) - h_{min}(i,j)\Big)$
(10) endif
```

A cell is marked as dry if its surface height is not higher than the lowest ground value in the cell and if there is no neighbor cell from which fluid could flow into this cell. The fill

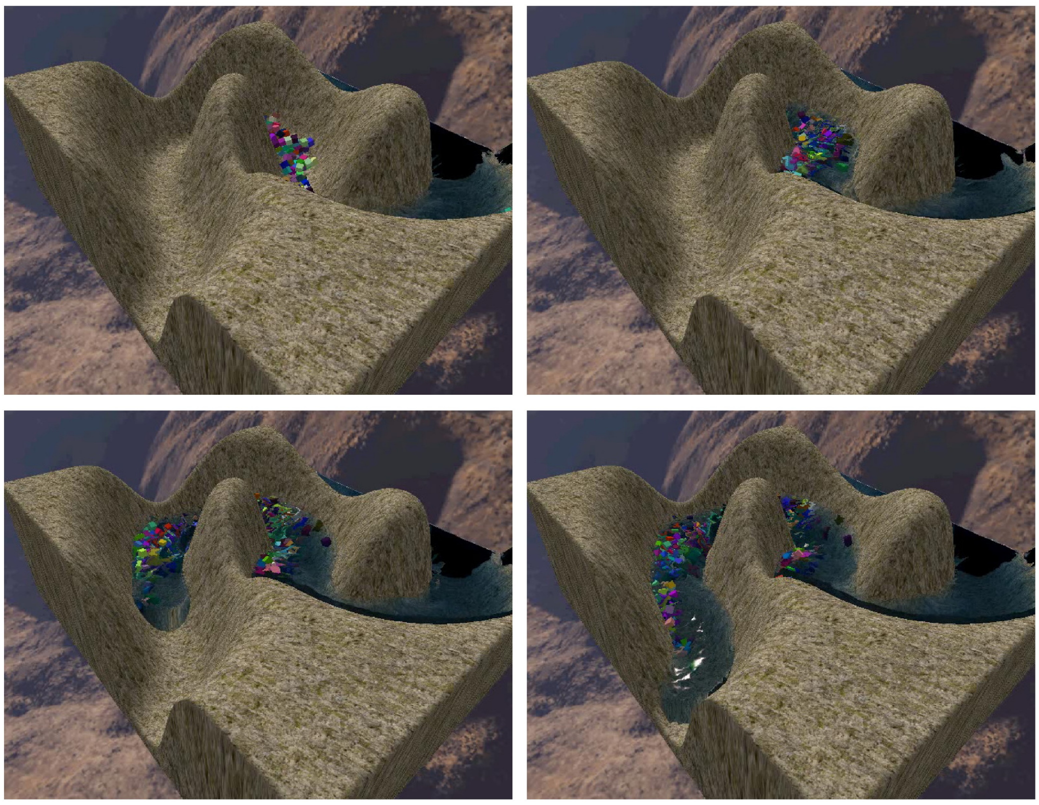

Figure 11.5: A wave flows through an S-shaped river bed, and pushes a large number of rigid bodies along with the flow.

ratio is then set to zero. ε_η can be seen as a threshold which allows inflow from a neighbor cell only if this neighbor has a large enough amount of fluid. This effectively limits the spread of thin layers of fluid. So this could be seen as a simple way of simulating surface tension. A cell is completely filled if its surface height is higher than the ground at any position in the cell. The fill ratio is then set to one. The cell is also marked as fluid if the surface height is only in parts higher than the ground level. In this case however the fill ratio is the ratio between minimal ground level, fluid surface height and maximal ground level.

Note, that with this definition cells may have negative depth values η even if they are marked as fluid. There are cases were the cell center itself is dry, so the value of η is negative at this point, while the whole cell still contains fluid at the edges of a cell.