# LAB REPORT: LAB 6

TNM079, MODELING AND ANIMATION

Linnea Bergman
linbe810@student.liu.se

Saturday 21st July, 2018

**Abstract**

Simulations are a common way to generate special effects such as water, smoke and fire. In the lab a simple fluid simulation system using the Navier-Stokes equations was implemented and tested. The semi-Lagrangian self advection was also implemented.

## 1 Introduction

This lab will focus on simulating water, which is a free surface fluid. Using the Stable Fluids approach of the Navier-Stokes equations the simulation was computed. The Navier-Stokes equations describe how a fluid changes over time, were the flow is described by a vector field. The fluid solver was extended with self advection using semi-Lagrangian integration.

## 2 Assignments

This section contains a more detailed explanation to the completed tasks. All modifications in the following sections was done in the file FluidSolver.cpp.

### 2.1 Assignment Implement basic functionality for the fluid solver

The first task was to complete the implementation of the basic steps for solving the Navies-Stokes equations in FluidSolver.cpp. The basic steps consists of three parts, external forces, Dirichled boundrary conditions and projection.

Step one was to add all external forces to the velocity field, which describes the flow of the simulation. These were added to the function ExternalForces() by explicit Euler integration. The external forces consisted mainly accelerations.

$$\hat{A}x = b \tag{1}$$

where $A = \nabla^2, x = q$ and $b = \nabla \mathbf{V}_2$. Equation 1 was used to compute the external forces. ExternalForces() input was the timestep $dt$. If there are no external forces exit the function. The world coordinates was gathered from the variable mVoxel. Each point in the voxel was tested and computed. First, the point was checked if it was in a fluid or not with the help of IsFluid()-function. If the point was within a fluid the external forces field was sampled, using

World coordinates, then the velocity field mVelocityField was updated by Euler integration. With Equation 1 the new values for the velocity field was computed and updated.

Step two was to enforce the Dirichlet boundary conditions in the function EnforceDirichlet-BoundaryCondition().

$$\mathbf{V} * \mathbf{n} = 0 \tag{2}$$

this is the Dirichlet boundary condition which states that that there can be no flow into or out of the boundary surface to which **n** is the normal. The following was done for each point in the mVoxel, along the $x-, y-$ and $z-$dimension. If the point $(i, j, k)$ is in the fluid, which was checked with IsFluid(), then the neighbours of the point was checked to see whether or not the point is next to a solid boundary. If the neighbour was solid, IsSolid()-function was used to see whether or not it was, and the velocity field for that point was none then the point was at the boundary and should not move. The velocity was set to zero along the given dimension to project the velocity of the boundary plane.

The last step for the basic functionality of the fluid solver was to compute the projection for the volume preservation in the function Projection(). The projection should preserve the volume and was computed using two equations, divergence and the Neumann boundary condition. Divergence of the velocity field was computed with:

$$\nabla \mathbf{V}_{i,j,k} = \frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y} + \frac{w_{i,j,k+1} - w_{i,j,k-1}}{2\Delta z} \tag{3}$$

where $u, v$ and $w$ ate the $x, y$ and $z$ components of the vectors in the vector field $\mathbf{V}$. The other equation is for the Neumann boundary condition which represents solids.

$$\nabla^2 q_{i,j,k} = \frac{1}{\Delta x^2} \begin{bmatrix} 1 & 1 & 1 & \text{-5} & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{q}_{i+1,j,k} \\ \mathrm{q}_{i-1,j,k} \\ \mathrm{q}_{i,j+1,k} \\ \mathrm{q}_{i,j,k} \\ \mathrm{q}_{i,j-1,k} \\ \mathrm{q}_{i,j,k+1} \\ \mathrm{q}_{i,j,k-1} \end{bmatrix} \tag{4}$$

where $(i, j, k)$ corresponds to the voxel at those grid coordinates. In Projection() there were no input values. To begin with the number of elements in the grid was counted. A sparse matrix with the same dimensions as the number of elements was created which had seven non-zero elements per grid point, representing the last part in Equation 4. Then two vectors, one for the $x$-values and one for the divergence $b$, of the same size as the number of elements, was created. $b$ was in the linear system of Equation 1. The following was done to every point in the voxel. First it was checked if the point $(i, j, k)$ were in the fluid. If not, then nothing was altered on that point. However, if the point was in the fluid then the linear indices of $(i, j, k)$ and its neighbours were computed. These are needed to get the right index of the sparse matrix and the vectors $b$ and $x$. The entry for vector $b$, the divergence of the velocity field, was computed using Equation 3. Then the entries for the sparse matrix, which represent the discrete Laplacian operator, was computed. The boundary conditions were enforced if the point was not next to a solid, which allows no change of flow in that direction. Equation 4 was used to compute the value of the fluid which were added to the sparse matrix. After all points had been checked the sparse matrix structure was rebuilt. With the conjugate gradient Equation 1 was solved. For each point $(i, j, k)$ in the voxel it was once again checked whether or not the point was a fluid. If it was then the linear indices of $(i, j, k)$ and its neighbour was computed. Using central differencing the gradient of $x$ at $(i, j, k)$ was computed with Equation 3 and subtracted from the velocity field. Thereby removing divergence which preserves the volume.

To test the basic functionality a template was added the the scene in the GUI. The outer box of the template was set as "solid" while the inner box-shape to "fluid". Propagate was used to see the solution ni time, one stable timestep at a time.

## 2.2 Assignment Implement semi-Lagrangian self advection

To extend the current non-physical fluid solver the self advection term was solved using semi-Lagrangian integration. The code modified was in the function SelfAdvection() in FluidSolver.cpp. This solves the Euler equations which model inviscid fluids. Input value to the function was the timestep and the number of steps to be performed. The current velocity field was copied. For every point $(i, j, k)$ in the voxel it was checked if the point was in a fluid or not. If it was in a fluid the following was done. The current velocity field at $(i, j, k)$ was sampled and then a particle at initial position $(i, j, k)$ was traced back in time through the velocity field the number of steps times. Each step is $(\frac{\partial t}{steps})$ time units long. Since the velocities were in world coordinates but the tracing was to be performed in grid space, the velocities were scaled accordingly to fit. When a particle was traced the velocities inbetween the grid points were interpolated using trilinear interpolation. When all points had been checked then the current velocity field was updated.

## 3 Results

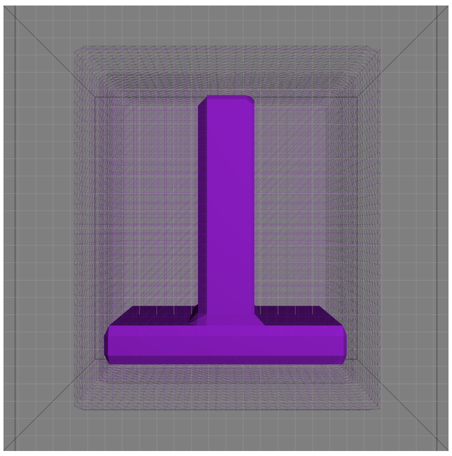
To visualize the result a video would be a better alternative since that shows how the fluid changes over time. Instead a collection of images was used to visualize the movement of the fluid.

Figure 1 displays the movement of the fluid after different number of propagations. Figure 1(a) is the original, the template that was used without any propagations or alternations. Figure 1(b) to 1(e) shows the movement of the fluid after 50, 100, 190 and 290 propagations.

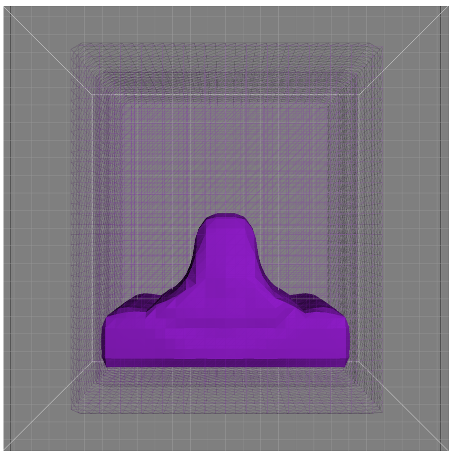
Figure 2 shows the movement of the fluid with self advection. Figure 2(a) is the original template without any modifications. Figure 2(b) to Figure 2(f) visualized the fluid movements with self advection after 50, 100, 170, 190 and 290 propagations.

## 4 Conclusion

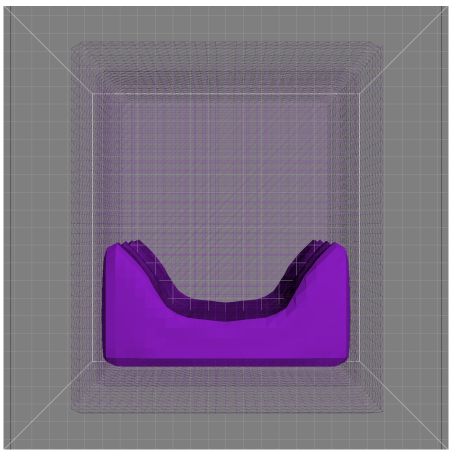
If the images in Figure 1 are studied carefully note then that the movement are stiff and does not behave as fluids usually do in reality. The fluid moves in general as a fluid should move but is disjointed and cubic. The movement of the fluid gets smoother with the self advection added, Figure 2, and generates a more realistic fluid motion. With the self advection term the fluid does not move as splash up the walls as much as it does without, compare Figure 1(c) and Figure 2(c), but it results in a more evenly spread fluid, see Figure 1(e) and Figure 2(f). However, the self advection fluid, which should be an inviscid fluid, was very viscous and behaved more like jelly than water. This is a bi-effect of the backwards tracing from the stable fluids approach in the self advection.



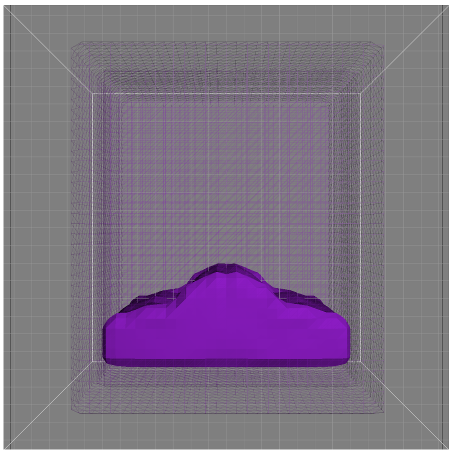
(a) Template at start



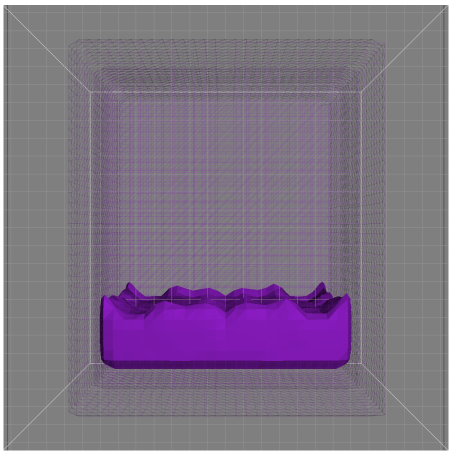
(b) Template after 50 iterations



(c) Template after 100 iterations

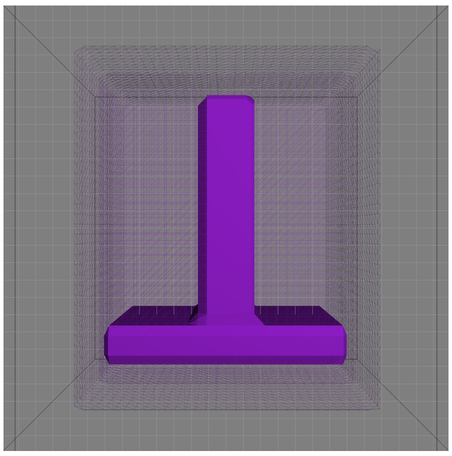


(d) Template after 190 iterations

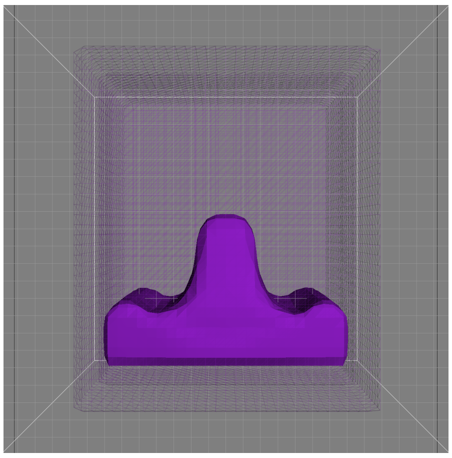


(e) Template after 290 iterations

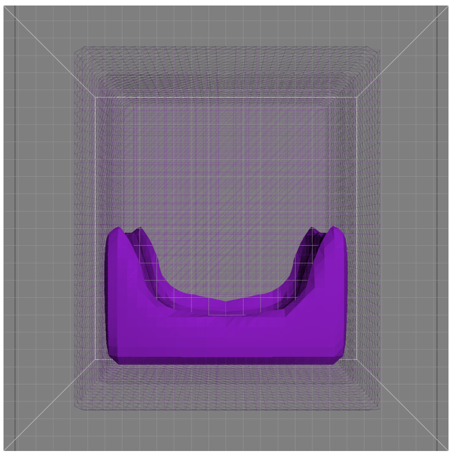
*Figure 1*
Result from assignment 2.1 showing the template used and its changes over time. The unaltered the template is shown in 1(a) and the movement of the fluid can be seen in Figure 1(b) to 1(e).



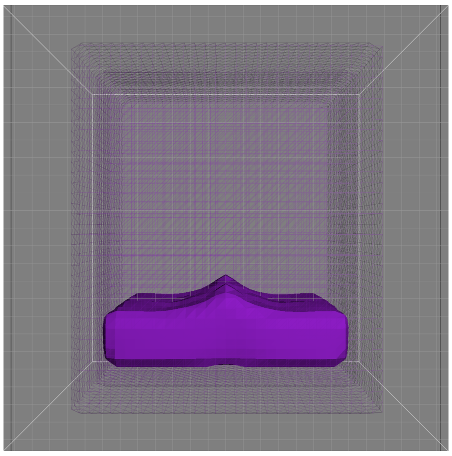
(a) Template at start



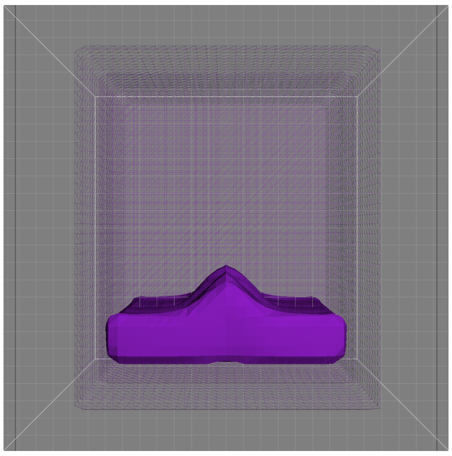
(b) Template after 50 iterations with self advection



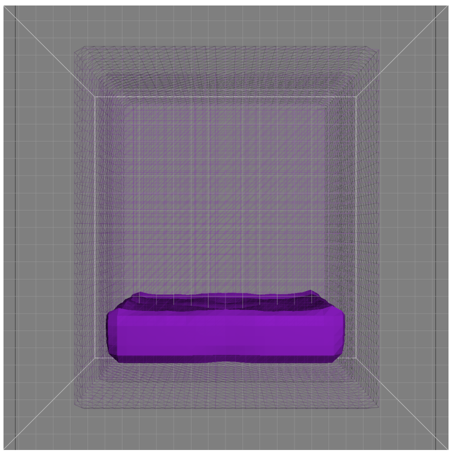
(c) Template after 100 iterations with self advection



(d) Template after 170 iterations with self advection



(e) Template after 190 iterations with self advection



(f) Template after 290 iterations with self advection

*Figure 2*

Result from assignment 2.2 showing the template used and its changes over time. The unaltered the template is shown in 2(a) and the movement of the fluid, with self advection, can be seen in Figure 2(b) to 2(f).

## 5 Lab partner and grade

This lab was completed together with Rebecca Cedermalm, rebca973. All assignments for grade 3 and grade 4 was completed which should result in grade 4.