Fluid Simulations - Lab 6 TNM079

Jonathan Bosson
jonbo665@student.liu.se

Sunday 11th September, 2016

Abstract

The paper discusses how to simulate smoke, water, and fire with the Navier-Stokes equations in a meshgrid data structure. This method can be used in special effects movies or video games to portrait fluid like models in an effective and quick way.

1 Introduction

This lab goes through the details how to solve the Navier-Stokes equations using *Stable Fields*. Although this lab only focuses on how to effectively simulate water can effects such as wind or fire easily be achieved using the same techniques. Smoke requires a diffusion term which is ignored in this lab, however its' poisson equation is very similar to the projection step used in this implementation.

2 Background

The Navier-Stokes equations are fairly wellknown and it describes how a velocity field, $\boldsymbol{V}$, in a fluid changes over time. The equations for incompressible flow are:

$$\frac{\partial \boldsymbol{V}}{\partial t} = \boldsymbol{F} + \nu \nabla^2 \boldsymbol{V} - (\boldsymbol{V} \cdot \nabla)\, \boldsymbol{V} - \frac{\nabla p}{\rho} \tag{1}$$

$$\nabla \cdot \boldsymbol{V} = 0, \tag{2}$$

where $\boldsymbol{V}$ is the velocity field, $\boldsymbol{F}$ is the external force term, ρ the constant density and p the pressure field.

By using a technique called *operator splitting* it is possible to solve the Navier-Stokes equations one term at a time. In essence, first can the temporary field $\boldsymbol{V}_1$ be calculated from $\boldsymbol{V}_0$ by solving for the self-advection term, $(\boldsymbol{V} \cdot \nabla)\boldsymbol{V}$. After that calculate $\boldsymbol{V}_2$ from $\boldsymbol{V}_1$ by adding the surface force, $\boldsymbol{F}$ and compute $\boldsymbol{V}_3$ from $\boldsymbol{V}_2$ by solving for the diffusion term, $\nu \nabla^2 \boldsymbol{V}$. Lastly calculate $\boldsymbol{V}_{\Delta t}$ from $\boldsymbol{V}_3$ by projecting the velocity field onto its divergence free part, this includes the terms $[\frac{\nabla p}{\rho}]$ and $\nabla \cdot \boldsymbol{V}$.

The viscous term is ignored in the lab since it aims to visualise water, which has a very low viscosity. Navier-Stokes equations without the viscosity term is called the *Euler equations* and can be solved through following scheme:

$$\boldsymbol{V}_0 \overset{(\boldsymbol{V}\cdot\nabla)\boldsymbol{V}}{\rightarrow} \boldsymbol{V}_1 \overset{\boldsymbol{F}}{\rightarrow} \boldsymbol{V}_2 \overset{\frac{\nabla p}{\rho}, \nabla\cdot\boldsymbol{V}}{\rightarrow} \boldsymbol{V}_{\Delta t}.$$

The self-advection is described in Navier-Stokes equations as $-(\boldsymbol{V} \cdot \nabla)\boldsymbol{V}$. It is clear that this term is non-linear, which is an important property that allows Navier-Stokes equations to model vortices or swirls in the fluid. The self-advection can be seen as moving the velocity field with itself and thus will straightfoward methods with pure advection to solve it cause instabilities. This is true if $\boldsymbol{V}$ wouldn't have been time dependant, but since it is can the self-advection be determined by solving following partial differential equation:

$$\frac{\partial \boldsymbol{V_1}}{\partial t} = -\left(\boldsymbol{V_0} \cdot \nabla\right) \boldsymbol{V_0} \tag{3}$$

This method calculates $\boldsymbol{V}_1$ by following the streamlines of $\boldsymbol{V}_0$ and thus is very dependant on

interpolation. Different interpolation methods will render vastly different results, in the lab a simple trilinear interpolation was used which is efficient and easy to implement but generates numerical viscosity.

The external force, $\boldsymbol{F}$, is equal to the time derivative of $\boldsymbol{V}_2$, and can thus be solved through a straightforward approach with Euler time integration.

$$\frac{\boldsymbol{V}_2 - \boldsymbol{V}_1}{\Delta t} = \boldsymbol{F} \Rightarrow \tag{4}$$

$$\boldsymbol{V}_2 = \boldsymbol{V}_1 + \Delta t \cdot \boldsymbol{F} \tag{5}$$

The last step is to prevent the fluid from compressing by enforcing $\nabla \cdot \boldsymbol{V} = 0$. A divergence free vector field defines a field where nothing is added or removed, and thus the pressure (as well as density) is constant which is equivalent to incompressibility. This is upheld by applying the projection step to $\boldsymbol{V}_2$ using *Helmhotz-Hodge decomposition* which states that each vector field $\boldsymbol{V}$ can be divided into a curl free part $\boldsymbol{V}_{cf}$ and a divergence free part $\boldsymbol{V}_{df}$.

$$\boldsymbol{V}_2 = \boldsymbol{V_{df}} + \boldsymbol{V_{cf}}. \tag{6}$$

$\boldsymbol{V}_{\Delta t}$ is defined as divergence free, and thus set to equal $\boldsymbol{V}_{df}$. The gradient of a scalar field, q, is always curl free which allows us to rewrite the equation to:

$$\boldsymbol{V}_{\Delta t} = \boldsymbol{V}_2 - \nabla q. \tag{7}$$

The divergence operator $\nabla\cdot$ is used to estimate value of q, resulting in the cancellation of the divergence free part $\boldsymbol{V}_{\Delta t}$ in equation 6.

$$\nabla \cdot \boldsymbol{V}_2 = \cancelto{0}{\nabla \cdot \boldsymbol{V}_{\Delta t}} + \nabla \cdot \nabla q \Rightarrow$$

$$\nabla \cdot \boldsymbol{V}_2 = \nabla^2 q. \tag{8}$$

The ∇q can be seen as the pressure field $\frac{\nabla p}{\rho}$. Since $\boldsymbol{V}_2$ is known can q be calculated through a poisson equation. The divergence operator is approximated through a numerical central differencing scheme:

$$\nabla \cdot \boldsymbol{V}_{i,j,k} = \frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y} + \frac{w_{i,j,k+1} - w_{i,j,k-1}}{2\Delta z}. \tag{9}$$

where u, v and w are the x, y and z components of the vectors in the vector field $\boldsymbol{V}$. Computing $\nabla^2 q$ would result in a rather complex equation. However since a uniform grid is used along all axises, $\Delta x = \Delta y = \Delta z$, can it be reduced to:

$$\nabla^2 q_{i,j,k} = \frac{1}{\Delta x^2} \begin{bmatrix} 1 & 1 & 1 & -6 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} q_{i+1,j,k} \\ q_{i-1,j,k} \\ q_{i,j+1,k} \\ q_{i,j,k} \\ q_{i,j-1,k} \\ q_{i,j,k+1} \\ q_{i,j,k-1} \end{bmatrix} \tag{10}$$

or in a more abstract description as

$$\mathbf{A}x = b \tag{11}$$

where $\mathbf{A} = \nabla^2$, $x = q$ and $b = \nabla \cdot \boldsymbol{V}_2$. The equation is solved by finding the *Poisson inverse matrix* of $\boldsymbol{A}$, or more efficiently by using an iterative parallel algorithm.

Two boundary conditions are important to consider when solving the poisson equation 8. These constraints are used such that the PDE rules out solutions to the equation that are not allowed, or in this case to allow fluid to interact with objects in the world. The first condition is the *Dirichlet boundary* which states that there can be no flow, in or out, of the boundary surface to which $\boldsymbol{n}$ is normal. In essence it forbids the fluid to flow into solid objects.

$$\boldsymbol{V} \cdot \boldsymbol{n} = 0 \tag{12}$$

The second boundary is the *Neumann boundary condition*:

$$\frac{\partial \boldsymbol{V}}{\partial \boldsymbol{n}} = 0 \tag{13}$$

This equation forbids any flow along the normal $\boldsymbol{n}$ of a boundary (solid) surface and is a required compliment to equation 12.

3 Results

The implementation of this lab was divided into three steps. The first step was to implement the external forces function, which went through every voxel with fluid and set the new velocity value at (i, j, k) according to equation 4.

Second step was to enforce the Dirichlet boundary condition such that there could not be any flow into a solid surface. In essence the function goes through all voxels with fluid and checks if it has any solid voxel neighbours. If so, the part of the velocity field in that direction was set to 0. For example:

if $\phi_{i-1,j,k} \in$ solid and $\boldsymbol{V}_{i,j,k}.x < 0$ **then**
  $\boldsymbol{V}_{i,j,k}.x \leftarrow 0$
end if

Last step was to implement the projection function, which solves the inverse poisson matrix $\boldsymbol{A}^{-1}$ discussed in equation 11. The divergence of the velocity field at position (i, j, k) is first calculated through central differencing. To maintain the Neumann boundary condition are the neighbouring voxels checked to see if they are solid. This is since the $\nabla^2 q_{i,j,k}$ can be described in a discrete laplacian frame. The result is a translated matrix containing seven entries. Each entry corresponds to a specific neighbour to voxel (i, j, k) and is equal to 1 if the neighbour is solid and 0 if not. The middle entry is equal to the negative sum of all neighbours. All entries in the matrix is then divided by $\frac{1}{\Delta x^2}$. For example:

$$\frac{1}{\Delta x^2}\begin{bmatrix}1 & 1 & 1 & -5 & 1 & 0 & 1\end{bmatrix} \tag{14}$$

Lastly the new velocity field at voxel (i, j, k) is calculated like equation 8 by using central differencing on the approximated q in the projection step and subtract it from the original velocity.

The result is an efficient but rough solver for fluid simulation, which can be seen in figures 1 to 4.

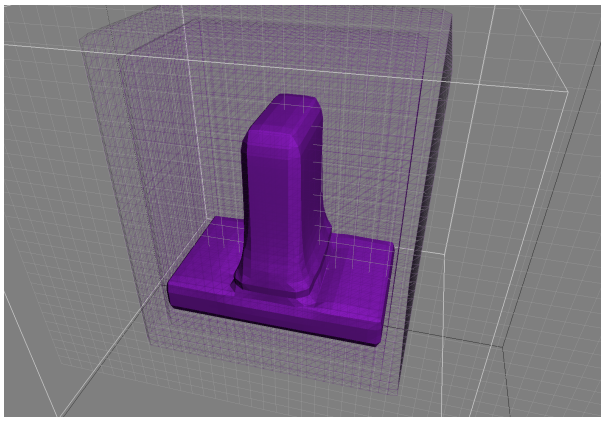

Figure 1: Snapshot of fluid simulation

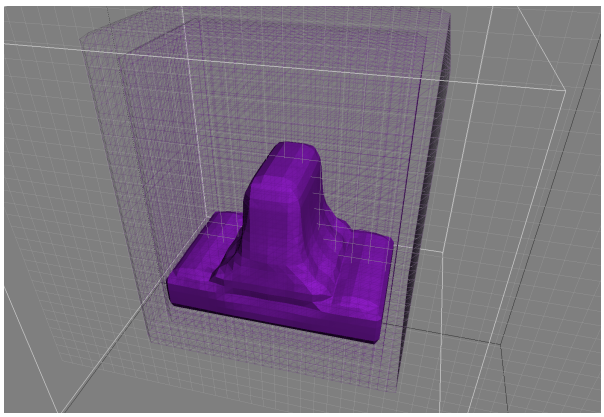

Figure 2: Snapshot of fluid simulation

This report aims for grade 3.

References

[1] M. E. Dieckmann, *Lecture Slides for TNM079*, **Lecture 8**, 2016.

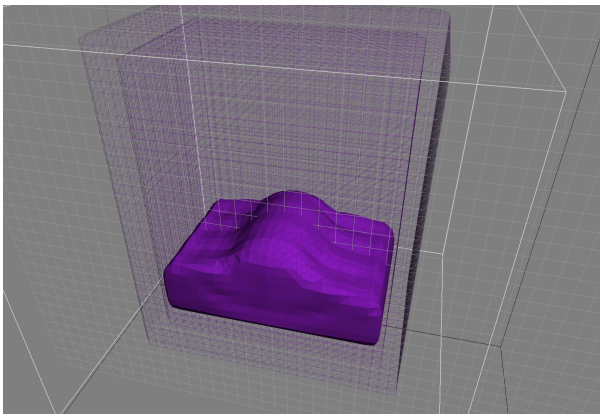

Figure 3: Snapshot of fluid simulation

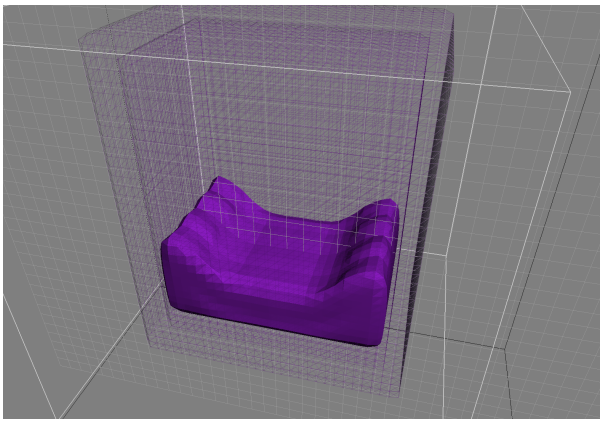

Figure 4: Snapshot of fluid simulation