FLUID SIMULATION

MODELING AND ANIMATION, LAB 6

Mark Eric Dieckmann Erik Junholm Emma Broman Robin Skånberg

Thursday 27th April, 2023

Abstract

The simulation of smoke, fire and water has become a common component of modern special effects movies. These phenomena are typically difficult to direct in real life and a fullscale experiment is often impossible. The aim of this lab is to get a hands-on experience of working with a simple fluid simulation system using the Navier-Stokes equations.

1 Introduction

In this lab we will go into detail regarding how to solve the Navier-Stokes equations in practice using the *Stable Fluids* [1, 2, 4] approach. Each term in the Navier-Stokes equations will be discussed in turn and a solution strategy as well as the tools required for that strategy will be described. The lab will focus on simulating free surface fluids - like water. The simulation of wind and smoke can however easily be done using the same techniques. The exception is the diffusion term which we will ignore for this lab. This term also results in a poisson equation and its solution is very similar to the projection step described below. More information on diffusion and smoke can be found in [4].

2 The Navier-Stokes equations

The famous Navier-Stokes equations describe how a fluid flow changes over time. The flow is described by $\boldsymbol{V}$, a *vector field*. A vector field is an entity that assigns a vector to every point in space. In the Navier-Stokes equations this vector field denotes the velocity of the fluid at every point in space and thus we usually refer to $\boldsymbol{V}$ as the *velocity field* of the fluid. The components of the velocity field are usually referred to as (u, v, w).

The equations for incompressible flow are

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

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

The individual terms are:

$(\boldsymbol{V}\cdot\nabla)\boldsymbol{V}$**,** this is called the *self-advection* term. It is responsible for “moving the flow with itself”. This non-linear term is very important for the behaviour of swirls and vortices.

$\boldsymbol{F}$**,** the *external force* term. It applies forces such as gravity, wind and buoyancy.

$\nu\nabla^2\boldsymbol{V}$**,** the *viscous stress* or ”viscosity”. This term corresponds to internal friction inside the fluid. It controls the thickness of the fluid. Honey has a high viscosity and water has a low one.

$\frac{\nabla p}{\rho}$**,** the pressure field p and the constant density ρ.

$\nabla\cdot\boldsymbol{V} = 0$**,** is a constraint that enforces a divergence free solution. This means that the volume of the fluid must remain constant over time.

$\frac{\nabla p}{\rho}$ and $\nabla\cdot\boldsymbol{V}$**,** are *together* responsible for maintaining the *incompressibility* of the solution. Due to the method used this is called the *projection step*.

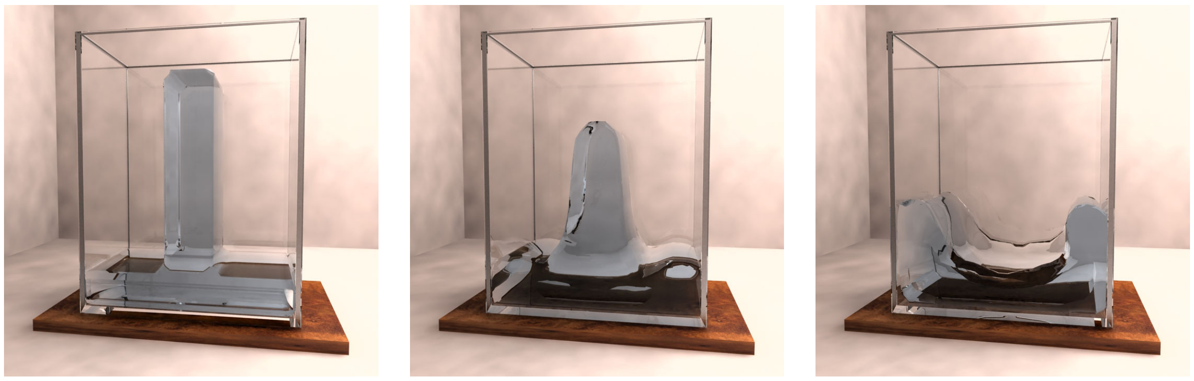


Figure 1: Ray traced example output of standard lab code without improvements. Courtesy of M. Popescu.

The Navier-Stokes equations are somewhat overwhelming at first and a direct solution is difficult to compute. But, using a technique known as *operator splitting* we can solve the Navier-Stokes equations in parts, i.e. we can solve for one term at a time. Given the initial vector field $\boldsymbol{V}_0$ at some point in time t_0 we calculate the vector field $\boldsymbol{V}_{\Delta t}$ at time $t_0 + \Delta t$ by solving a series of manageable sub-problems. These are not in the same order as in equation (1).

$$\boldsymbol{V}_0 \overset{(\boldsymbol{V}\cdot\nabla)\boldsymbol{V}}{\rightarrow} \boldsymbol{V}_1 \overset{\boldsymbol{F}}{\rightarrow} \boldsymbol{V}_2 \overset{\nu\nabla^2\boldsymbol{V}}{\rightarrow} \boldsymbol{V}_3 \overset{\frac{\nabla p}{\rho},\nabla\cdot\boldsymbol{V}}{\rightarrow} \boldsymbol{V}_{\Delta t}$$

In text the outline reads:

1. Begin by calculate the temporary field $\boldsymbol{V}_1$ from $\boldsymbol{V}_0$ by solving for the self-advection term, $(\boldsymbol{V} \cdot \nabla)\boldsymbol{V}$.
2. Then calculate $\boldsymbol{V}_2$ from $\boldsymbol{V}_1$ by adding the effect of external forces, $\boldsymbol{F}$.
3. Now calculate $\boldsymbol{V}_3$ from $\boldsymbol{V}_2$ by solving for the diffusion term, $\nu\nabla^2\boldsymbol{V}$.
4. Finally 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 $\nabla \cdot \boldsymbol{V}$.

In this lab we will focus on simulating water and since water has very low viscosity we choose to ignore the diffusion term. The Navier-Stokes equations without viscosity is called the *Euler equations*.

The algorithm for solving Euler's equation is

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

We proceed by examining these three terms in detail. Then we pay special attention to the projection step as it constitutes the most difficult part. Both in terms of programming and efficiency.

2.1 The self-advection term

The self-advection term is the non-linear $-(\boldsymbol{V} \cdot \nabla)\boldsymbol{V}$ term of the Navier-Stokes equations. This term is important since it allows for non-linear phenomena like vortices. Using straightforward methods for representing the self-advection term will unfortunately result in solvers that easily become unstable. The self-advection term can be interpreted as *the motion of the velocity field along itself* and thus we achieve unconditional stability by modeling this term as pure advection instead. This approach is true under the assumption that the velocity field $\boldsymbol{V}$ is *not time dependant*. However, since we know that the velocity field is time dependent this approach to solving the self-advection term will always be approximate, regardless of spatial resolution! In order to calculate the new temporary velocity field $\boldsymbol{V}_1$ from $\boldsymbol{V}_0$ we need to solve the following partial differential equation:

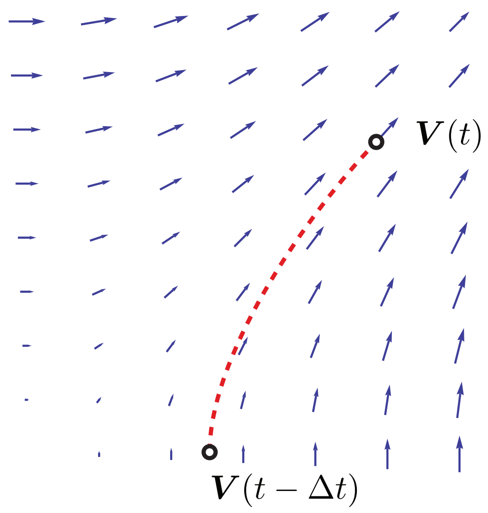


Figure 2: Self-advection using backwards tracing from the stable fluids approach.

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

These equations can be solved by the *method of characteristics*. In essence this method allows us to calculate $\boldsymbol{V}_1$ by following stream-lines of $\boldsymbol{V}_0$. Given the velocity field $\boldsymbol{V}_0$ we create $\boldsymbol{V}_1$ by using the velocity of $\boldsymbol{V}_0$ at the position that corresponds to the position a zero-mass particle would have had Δt time-units ago as is seen in figure 2.

If we define the particle trace operator $\mathbf{T}$ we can write this step as the equation

$$\boldsymbol{V}_1(\boldsymbol{x}) = \boldsymbol{V}_0(\mathbf{T}(\boldsymbol{x}, -\Delta t)) \tag{4}$$

This method of solution relies heavily on interpolation. The choise of interpolant is thus important to the quality of the solution. In this lab we use simple trilinear interpolation which is fast and easy to implement but results in numerical viscosity.

2.2 The force term

Solving the external force term is quite straightforward. We simply define the force-field $\boldsymbol{F}$ and create $\boldsymbol{V}_2$ from $\boldsymbol{V}_1$ by solving the equation

$$\frac{\partial \boldsymbol{V}_2}{\partial t} = \boldsymbol{F} \tag{5}$$

We can for example do this by means of simple first order Euler time integration:

$$\frac{\boldsymbol{V}_2 - \boldsymbol{V}_1}{\Delta t} = \boldsymbol{F} \Rightarrow \tag{6}$$

$$\boldsymbol{V}_2 = \boldsymbol{V}_1 + \Delta t \cdot \boldsymbol{F} \tag{7}$$

2.3 The projection step

The final step in the Navier-Stokes solver algorithm is to enforce incompressibility. The term $\nabla \cdot \boldsymbol{V} = 0$ is responsible for this. A divergence-free vector field is volume-conserving and thus incompressible.

The constraint is kept by applying the projection operator to the vector field $\boldsymbol{V}_2$. We do this by first calculating the curl-free part of $\boldsymbol{V}_2$ by means of the *Helmholtz-Hodge decomposition*. It states that it is always possible to split a vector field into a *divergence free* part $\boldsymbol{V_{df}}$ and a *curl free* part $\boldsymbol{V_{cf}}$:

$$\boldsymbol{V}_2 = \boldsymbol{V_{df}} + \boldsymbol{V_{cf}}. \tag{8}$$

We can identify our final divergence-free vector field $\boldsymbol{V}_{\Delta t}$ as $\boldsymbol{V_{df}}$ and construct a curl free field as the gradient of some scalar field q (note that a gradient field is always curl-free):

$$\boldsymbol{V}_2 = \boldsymbol{V}_{\Delta t} + \nabla q. \tag{9}$$

We rearrange and arrive at:

$$\boldsymbol{V}_{\Delta t} = \boldsymbol{V}_2 - \nabla q. \tag{10}$$

Now for the tricky part. We assign the value of q to the pressure field p and fit the projection step into the Navier-Stokes equations (1) by subtracting *the pressure field which makes the solution incompressible*:

$$\boldsymbol{V}_{\Delta t} = \boldsymbol{V}_2 - \frac{\nabla p}{\rho}, \quad p = q, \text{and } \rho = 1. \tag{11}$$

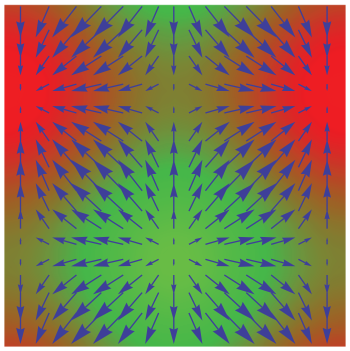


Figure 3: Visualization of the divergence of a vector field $\boldsymbol{V}(x,y) = \sin(x), \cos(y)$. Green = positive divergence (sources) and red = negative (sinks).

But we do not yet know the value of q. To find it we apply the divergence operator, $\nabla\cdot$, to all terms in equation (9):

$$\begin{aligned} &\nabla \cdot \boldsymbol{V}_2 = \cancelto{0}{\nabla \cdot \boldsymbol{V}_{\Delta t}} + \nabla \cdot \nabla q \Rightarrow \\ &\nabla \cdot \boldsymbol{V}_2 = \nabla^2 q. \end{aligned} \tag{12}$$

Make sure that you understand the cancellation. Since $\boldsymbol{V}_2$ is known this allows us to find the q that fulfills the Helmholtz-Hodge decomposition (9).

Equation (12) belongs to a class of equations called *Poisson equations* and its solution is the field q that defines the curl-free part of $\boldsymbol{V}_2$. We discretize equation (12) by using numerical approximations to the divergence and laplacian operators.

2.3.1 Discrete divergence

The divergence is an operator that measures whether the vector field at a point is "expanding" or "contracting", and the magnitude of that expansion/contraction. The point in question is labeled a source or sink, respectively. See figure 3 for a visualization of the concept.

We discretize the divergence operator using cen-

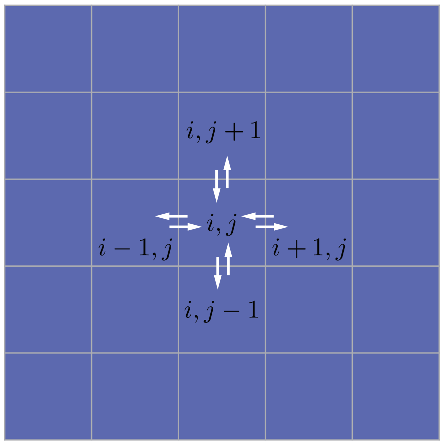


Figure 4: Visualization of the exchange described by (14).

tral differencing:

$$\nabla \cdot \boldsymbol{V}_{i,j,k} = \begin{aligned} &\frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} + \\ &\frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y} + \\ &\frac{w_{i,j,k+1} - w_{i,j,k-1}}{2\Delta z}. \end{aligned} \tag{13}$$

where u, v and w are the x, y and z components of the vectors in the vector field $\boldsymbol{V}$.

2.3.2 Discrete Laplacian

The Laplacian applied on a scalar field q is defined as the divergence of the gradient of q. The discretized laplacian can be written as

$$\nabla^2 q_{i,j,k} = \begin{aligned} &\frac{q_{i+1,j,k} - 2q_{i,j,k} + q_{i-1,j,k}}{(\Delta x)^2} + \\ &\frac{q_{i,j+1,k} - 2q_{i,j,k} + q_{i,j-1,k}}{(\Delta y)^2} + \\ &\frac{q_{i,j,k+1} - 2q_{i,j,k} + q_{i,j,k-1}}{(\Delta z)^2} \end{aligned} \tag{14}$$

The physical interpretation of this expression is that the laplacian describes the exchange of material between the voxel (i, j, k) and its neighbors, see figure 4. The material being exchanged is whatever quantity that is described by the field q.

Since we use a uniform grid we know that $\Delta x = \Delta y = \Delta z$ and equation (14) can be reduced to

$$\nabla^2 q_{i,j,k} = \frac{1}{(\Delta x)^2} \cdot \begin{aligned}(&q_{i+1,j,k} + q_{i-1,j,k} + \\ &q_{i,j+1,k} + q_{i,j-1,k} + \\ &q_{i,j,k+1} + q_{i,j,k-1} - 6q_{i,j,k})\end{aligned} \quad (15)$$

The laplacian can be represented in vector notation:

$$\nabla^2 q_{i,j,k} = \frac{1}{\Delta x^2} \begin{bmatrix} 1 & 1 & 1 & -6 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} q_{i+1,j,k} \\ q_{i-1,j,k} \\ q_{i,j+1,k} \\ q_{i,j,k} \\ q_{i,j-1,k} \\ q_{i,j,k+1} \\ q_{i,j,k-1} \end{bmatrix} \quad (16)$$

2.3.3 Solving the Poisson Equation

By using the discrete laplacian and divergence operator (13) the poisson equation (12) becomes

$$\mathbf{A}x = b \quad (17)$$

where $\mathbf{A} = \nabla^2$, $x = q$ and $b = \nabla \cdot \boldsymbol{V}_2$. This system can be solved by finding the inverse to the *Poisson matrix* $\mathbf{A}$, or more efficiently by using an iterative algorithm as discussed in the next section.

3 The projection step in detail

Making the fluid incompressible is the most difficult task, both mathematically and algorithmically. For example, applying the discrete laplacian on equation (12) will result in a linear system of equations with one unknown for each voxel that contains fluid. For a 3D simulation grid with 100x100x100 voxels this means that we can have up to 10^6 unknowns. This will in turn result in a gigantic 10^6 x 10^6 element matrix for which we need to find the inverse.

The matrix is sparse and contains many zeros. Hence we should use a solver that can take advantage of this property. In this lab we will use the conjugate gradient (CG) [3] method to manage this. The conjugate gradient method is an iterative approach to solving linear equation systems. Each iteration of the CG algorithm will improve the solution and an exact solution will be found in $O(\sqrt{n})$ iterations where n is the number of unknowns. In order to speed things up we will not iterate until we have found the perfect solution. Instead we iterate until we have obtained a solution where the total error is smaller than some small value ϵ. By using the conjugate gradient method we can reduce the storage requirement of the matrix from $O(n^6)$ scalar elements to $O(n^3)$ scalar elements.

3.1 Boundary conditions

When solving the Poisson equation (12) we also need to take two *boundary conditions* into account. Boundary conditions are constraints on a PDE that rule out solutions to that equation that are not allowed. In the case of fluid simulation, we can use boundary conditions to model interactions between the fluid and other objects in the world. The two types of boundary conditions we need to take into account are the *Dirichlet* boundary condition and the *Neumann* boundary condition.

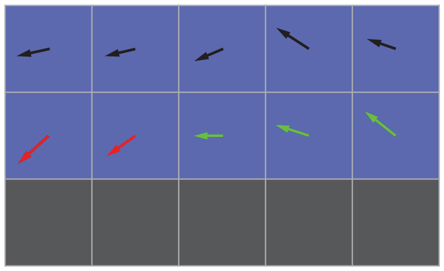
3.1.1 Dirichlet boundary condition

The Dirichlet boundary condition is

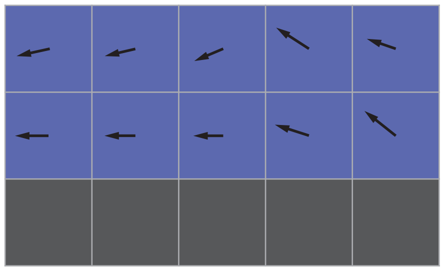
$$\boldsymbol{V} \cdot \boldsymbol{n} = 0 \quad (18)$$

and states that there can be no flow into or out of the boundary surface to which $\boldsymbol{n}$ is the normal. We enforce this boundary condition directly on the fields $\boldsymbol{V}_2$ and $\boldsymbol{V}_{\Delta t}$. I.e. *before* and *after* we project the velocity field onto its divergence-free part. We enforce the Dirichlet boundary condition by finding velocity vectors that neighbor a solid object and if the vector points toward the solid object we project the vector onto the tangent plane of the surface. This is accomplished in a discretized manner by the following algorithm outlined for the x-component:

```
procedure ENFORCEDIRICHLET( )
    for all $\phi_{i,j,k} \in$ fluid do
        if $\phi_{i-1,j,k} \in$ solid and $\boldsymbol{V}_{i,j,k}.x < 0$ then
```



(before)



(after)

Figure 5: The velocity field before and after application of the dirichlet boundary condition. Only fluid flow *into* solid (red) is prevented. Blue cells = water, gray cells = solid.

```
            $V_{i,j,k}.x \leftarrow 0$
        else if $\phi_{i+1,j,k} \in$ solid and $V_{i,j,k}.x > 0$ then
            $V_{i,j,k}.x \leftarrow 0$
        end if
        ⋮                                   ▷ Same for y, z-components
    end for
end procedure
```

We have now guaranteed that no velocity vectors will ever point into solids - thus we guarantee that the fluid will not flow into any solid objects!

3.1.2 Neumann boundary condition

The Neumann boundary condition is

$$\frac{\partial \boldsymbol{V}}{\partial \boldsymbol{n}} = 0 \tag{19}$$

and states that there shall be no change of flow along the normal $\boldsymbol{n}$ of a boundary surface. This condition is an essential complement to equation (18). Equation (19) can be enforced by eliminating the connection between solid boundary cells and fluid cells when building the linear equation system for the discrete Poisson equation. Assuming that the grid cell at position i, j, k contains fluid while the grid cell $i-1, j, k$ belongs to a solid object we approximate the Neumann boundary condition as

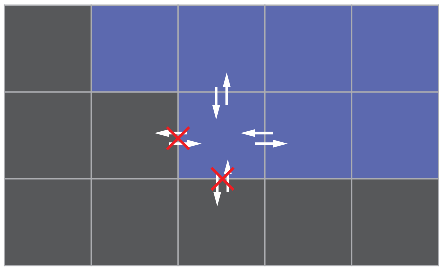


Figure 6: Visualization of the application of the Neumann boundary condition as described by equation (19).

$$u_{i-1,j,k} - u_{i,j,k} = 0 \tag{20}$$

when discretizing the Laplace operator for grid cell i, j, k. I.e. we state that there shall be no exchange between the solid cell $i-1, j, k$ and the neighboring fluid cell i, j, k.

3.2 Building the poisson matrix

In order to solve equation (17) we need to create the matrix $\mathbf{A}$. As previously stated this matrix is the discrete representation of the Laplace operator ∇^2. In the projection step (17) this matrix is multiplied by the field q. You can think of q as a pressure field. During the projection step, we calculate a pressure field such that the pressure differences will force the vector field $\boldsymbol{V}_2$ to become divergence-free (i.e. volume conserving). The matrix $\mathbf{A}$ will contain one row for each voxel that is classified as containing fluid. Due to the discretization, each row will only contain at most 7 elements that are not zero: One element for each of the neighboring voxels in the x, y and z direction and one element for the voxel itself. You can assure yourself of this by studying the discrete laplacian (15).

Now the question becomes what value should each of these non-zero elements contain? This can be determined by the classification of the voxel that corresponds to the matrix element. Remember: each element in the matrix represents a voxel in the computational grid. The center element is always a fluid voxel, but the neighbors may also be solid or empty.

3.2.1 What is solid and what is fluid?

When dealing with free surface fluid simulation we will need two kinds of level sets: a fluid level set representing the fluid surface and one or more solid level sets representing solid boundaries. In the fluid solver we then need to classify each voxel as being either fluid, solid, or empty. We will use the following classification:

$$\text{class} = \begin{cases} \text{fluid}, & \phi_f \leq \Delta x/2 \\ \text{solid}, & \phi_s \leq 0 \\ \text{empty}, & \text{otherwise.} \end{cases} \tag{21}$$

Here ϕ_f and ϕ_s are the level set distance functions describing the fluid and solid.

Assume that we are building the matrix row corresponding to the voxel at grid coordinates (i, j, k). If either of the 6 immediate neighbor voxels contains water we should allow fluid to be exchanged between the center voxel (i, j, k) and the neighbor voxel. This is achieved by setting the row element corresponding to the neighbor voxel to 1.

What if the neighbor voxel is empty (i.e contains air) instead? Once again we want to allow fluid to flow into this region and we set the row element corresponding to this voxel to 1.

The final case is that the neighbor voxel is inside a solid. Since solids represent boundaries we now need to take the Neumann boundary condition into account. Assume that the neighbor voxel $(i, j, k + 1)$ belongs to a solid object. According to (20) we can enforce this boundary condition by maintaining the relation $u_{i,j,k+1} - u_{i,j,k} = 0$. If we substitute this expression into the expression for the discrete laplacian (15) we obtain

$$\nabla^2 q_{i,j,k} = \frac{1}{\Delta x^2} \begin{bmatrix} 1 & 1 & 1 & -5 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_{i+1,j,k} \\ q_{i-1,j,k} \\ q_{i,j+1,k} \\ q_{i,j,k} \\ q_{i,j-1,k} \\ q_{i,j,k+1} \\ q_{i,j,k-1} \end{bmatrix} \tag{22}$$

We see that the Neumann boundary condition simply results in that the constant in front of voxel $(i, j, k + 1)$ becomes 0 instead of 1 and the constant in front of the center element (i, j, k) becomes -5 instead of -6, i.e. it has been increased by 1. These constants are the values that should be written in each matrix row. Since only the neighbor and center voxels are represented we know implicitly that the row values on the position for that voxel will be zero. This is the reason why the Poisson matrix is so sparse. Each row has one value for each voxel in the grid, but for each row all values except 7 (at most) are non-zero!

4 Advecting the fluid interface

In order to advect any level set along a velocity field we must guarantee that the velocity field is defined at least in the narrow band. Unfortunately, this is not the case with the field generated when solving the Navier-Stokes equations. The solution only calculates velocities for voxels that contain fluid. This translates to all cells on or inside the interface. This means we must find some way to extend the velocity field to the outside region of the interface so that the level set can be moved properly. One way to do this is using a method known as velocity extension. The idea is to propagate velocity information out in the normal direction from the interface the same way as distance information is being propagated during level set re-distancing. To do this we use the general equation for extrapolating some characteristic S in the direction of the normal $\boldsymbol{N}$:

$$\frac{\partial S}{\partial \tau} + \boldsymbol{N} \cdot \nabla S = 0 \tag{23}$$

In our case S will be a field of velocity vectors leading to the corresponding equation for extrapolating the velocity field $\boldsymbol{V}$ in the direction of $\boldsymbol{N}$:

$$\frac{\partial \boldsymbol{V}}{\partial \tau} = -\boldsymbol{N} \cdot \nabla \boldsymbol{V} \tag{24}$$

It should be noted that this equation only extrapolates velocities into the outside region of the level set. We can extrapolate velocities in both directions by adding the sign of the distance field in front of the right-hand side (RHS) of equation (3.27). The final equation for velocity extension into both the inside and outside region of the level set thus becomes

$$\frac{\partial \boldsymbol{V}}{\partial \tau} = -\operatorname{sgn}(\phi) \boldsymbol{N} \cdot \nabla \boldsymbol{V} \tag{25}$$

We will use the information on the interface as boundary condition, thus extrapolating the velocity field on the interface outwards along the normal field provided by the level set. In order to successfully do this we need to use upwind differentiation. Upwind differentials are used both when calculating the normal and when calculating the gradient of the velocity field in equation (25) according to the following scheme: We loop through all cells in the band structure. We choose to differentiate in the direction of the smallest distance, i.e. the direction of the closest interface point:

```
procedure PickUpwinding( )
    if $\phi_{i+1,j,k} < \phi_{i,j,k}$ and $\phi_{i-1,j,k} < \phi_{i,j,k}$ then
        if $\phi_{i+1,j,k} > \phi_{i-1,j,k}$ then
            return $\phi_x^+$
        else
            return $\phi_x^-$
        end if
    else if $\phi_{i+1,j,k} \leq \phi_{i,j,k}$ then
        return $\phi_x^+$
    else if $\phi_{i-1,j,k} < \phi_{i,j,k}$ then
        return $\phi_x^-$
    else
        return $\phi_x = 0$ ▷ Direction indeterminable, no information should flow into this point. Set all differentials to zero.
    end if
end procedure
```

This example is along the x-axis but the algorithm looks the same for the y- and z-axis.

After calculating the normal vector and the velocity gradient matrix at each cell we can calculate the right hand side of equation (25) and use first order Euler time integration to update the field one time-step. As for the re-initialization equation we iterate these steps until the RHS of equation (25) approaches zero and we have reached a steady state.

When solving equation (25) one must also take a CFL condition into account. A time step of 0.5 works fine, but the more aggressive time-step of 0.7 is used to get faster convergence.

5 Assignments

5.1 Grading

The assignment (3) is mandatory to pass the lab, grade 3. Completing assignment (4) gives you the grade 4 and completing (4,5) give you the grade 5.

Implement basic functionality for the fluid solver (3)

The class `FluidSolver` contains the basic steps for solving the Navier Stokes equations presented in the text. Your task is to complete the implementation of the steps:

1. **External forces**: Add all external forces (accelerations) to the velocity field by explicit Euler integration in `FluidSolver::ExternalForces()`
2. **Dirichlet boundary conditions**: Enforce the Dirichlet boundary conditions in `FluidSolver::EnforceDirichletBoundaryCondition()`
3. **Projection**: Compute the projection for volume preservation in `FluidSolver::Projection()`

Study the existing code in `FluidSolver` to get an understanding of the complete pipeline, starting in `FluidSolver::Solve()`. The GUI has a menu option "Add template 1" which contains a test setup for your fluid. Add this template and set the outer box as "solid" and the inner box-shape as "fluid". Then you can click "Propagate" to propagate the solution in time. If you set the time to zero in the text box, it will propagate the solution one stable timestep. You can study the velocity field by "Visualize velocity field" which adds a cut plane. Do this after completing each of the steps above to see how your solution behaves. You can use the "Add template 2" and "Add template 3" to construct your own scene. Maybe a fluid teapot?

Implement semi-Lagrangian self advection (4)

Extend your current non-physical fluid solver by solving the self advection term using semi-Lagrangian integration. Add your code to the function `FluidSolver::SelfAdvection()`. You are now solving the Euler equations which model inviscid (zero viscosity) fluids. Do you notice any difference compared to the results in the previous task? As you can see your inviscid fluid is depressingly viscous. Can you figure out an explanation for why this is?

Improved volume conservation (5)

As you may have noticed, in spite of all your efforts, your fluid is still losing a lot of volume. This is mainly caused by numerical diffusion of sharp features on the level set. A particle level set will almost completely remove this problem, but there is a simple trick that can significantly improve the visual appearance of your fluid. Since we are losing mass the logical thing would be to add the lost mass back again! So how do we do this? Remember that the projection step is supposed to create a divergence-free (i.e. mass conserving) field. So what if we could make the projection create a field that actually adds a small amount of mass whenever necessary? The right-hand side of the Poisson equation ($\nabla \cdot \boldsymbol{V}_2$) actually corresponds to sources and sinks so we can add an artificial source term that tries to counter the volume loss by adding volume everywhere in the fluid. The fluid solver stores the original level set volume in the member variable `mInitialVolume` and the current volume in the variable `mCurrentVolume`. The unit of a source is $[m^3/s]$. Use this knowledge to try and restore all lost volume during each timestep. In order for this to work well you may have to play a bit with scaling your source term so that enough mass is injected each timestep.

6 Acknowledgements

The lab scripts and the labs were originally developed by Gunnar Läthén, Ola Nilsson, Andreas Söderström and Stefan Lindholm.

References

[1] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 23–30. ACM SIGGRAPH, ACM Press / ACM SIGGRAPH, August 2001.

[2] Nick Foster and Demitri Metaxas. Realistic animation of liquids, May 1996.

[3] M. R. Hestenes and E. L. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards Sect. 5*, 49:409–436, 1952.

[4] Jos Stam. Stable fluids. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 121–128, August 1999.