Stable Fluids in 3D

Wen Qin Ye
wenqin.ye@mail.utoronto.ca
University of Toronto
Toronto, ON, Canada

Bence Peter Weisz
ben.weisz@mail.utoronto.ca
University of Toronto
Toronto, ON, Canada

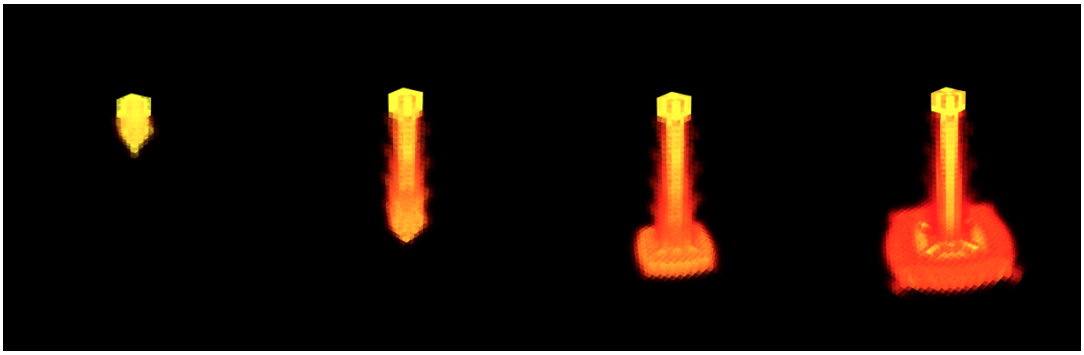

Figure 1: Stages of fluid flow

ABSTRACT

In this paper we describe our implementation of Jos Stam's Stable Fluids paper [2]. We first cover the physics based method behind stable fluids, and then describe how we implemented the simulation using the Eigen C++ library.

CCS CONCEPTS

• **Computing Methodologies** → Modeling and simulation.

KEYWORDS

Stable fluids, Navier-Stokes, fluid simulation, C++, Eigen, Libigl

1 INTRODUCTION

The Stable Fluid method [2] represents a way to simulate fluids that are unconditionally stable. We begin the paper by describing the mathematical ideas behind the Stable Fluid method in **Section 2**. More specifically, you will see that simulating a fluid involves numerically solving for the Navier-Stokes equations, which itself can be broken down into a four step process. We then explain how each step is implemented in detail in **Section 3** and also describe how to simulate the movement of a substance in the fluid. Finally we end off by displaying some results from our implemented stable fluids simulation in **Section 4**.

2 METHOD

2.1 Navier-Stokes Equation

The Navier-Stokes equations are a set of equations that describes the change in the velocity field for a fluid over time.

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{2}$$

Where $\mathbf{u}$ is the velocity field, ρ is the density of the fluid, ν is the kinematic viscosity, and $\mathbf{f}$ is an external force (represented as a vector field)

Since the Navier-Stokes equations cannot be solved analytically, we must solve approximations of the problem in four distinct steps. These steps are presented below as the Add Force, Advect, Diffuse and Project operations. Note that at a given time step, $\mathbf{w_0}$ is the equivalent to $\mathbf{u}(\vec{x}, t)$, while $\mathbf{w_4}$ is equivalent to $\mathbf{u}(\vec{x}, t+1)$.

$$\mathbf{w_0}(\vec{x}) \overbrace{\rightarrow}^{\text{Add Force}} \mathbf{w_1}(\vec{x}) \overbrace{\rightarrow}^{\text{Advect}} \mathbf{w_2}(\vec{x}) \overbrace{\rightarrow}^{\text{Diffuse}} \mathbf{w_3}(\vec{x}) \overbrace{\rightarrow}^{\text{Project}} \mathbf{w_4}(\vec{x})$$

A more in depth explanation of the steps follows.

2.2 Step 1: Add Force

The add force step involves adding an external force onto the velocity field. This step produces a new velocity vector field $\mathbf{w_1}$.

$$\mathbf{w_1} = \mathbf{w_0} + \Delta t\, \mathbf{f} \tag{3}$$

Note that $\mathbf{f}$ is a vector field of forces that can vary over time.

2.3 Step 2: Advection

The advection step propagates forces in the fluid and corresponds to the $-(\mathbf{u} \cdot \nabla)\mathbf{u}$ part of the Navier-Stokes equation. Stam [2] solves for the advection term using a technique called *Method of Characteristics* which is a technique for solving differential equations. The result of the *Method of Characteristics* technique tells us that to solve for the advection term we back trace each position in the velocity field some time Δt ago. Then for each position, we set the velocity of the field at that position to be the velocity of the field at its corresponding backtraced position. Applying this advection procedure gives us our new velocity vector field $\mathbf{w_2}$.

$$\mathbf{w_2} = \text{advect}(\mathbf{w_1}) \tag{4}$$

2.4 Step 3: Diffusion

The diffusion steps involves solving the following system to solve for the effect of viscosity on the fluid. This produces a new velocity vector field $\mathbf{w_3}$.

$$(\mathbf{I} - vt\nabla^2)\mathbf{w_3} = \mathbf{w_2} \tag{5}$$

2.5 Step 4: Projection

The projection step is required to remove any divergence from the velocity vector field. By projecting the velocity vector field we are ensuring that first equation of the Navier-Stokes equations are satisfied and that the velocity field is divergence-free.

The first step in doing so is computing the divergence of the vector field $\mathbf{w_3}$ and solving the equation below:

$$\nabla^2\mathbf{q} = \nabla \cdot \mathbf{w_3} \tag{6}$$

The solution of the system $\mathbf{q}$ is used to then remove the divergence from the velocity vector field and thus produces a new velocity field $\mathbf{w_4}$.

$$\mathbf{w_4} = \mathbf{w_3} - \nabla\mathbf{q} \tag{7}$$

3 IMPLEMENTATION

We discretize our velocity field into **DIM x DIM x DIM** many cubes, where the velocities of the field are defined at the center of each cube and each cube is of unit size. In our implementation, this 3D velocity velocity field $\mathbf{u}(i, j, k) = [u_x\ u_y\ u_z]^\top$ is flattened into 3, 1D vectors `Eigen::VectorXd` objects. We flatten the vector fields in order to be able to use linear solvers with the vector fields. Each of these vectors store the x, y, z components of the vector field $\mathbf{u}$ respectively. Each are of size **DIM3**, where **DIM3** is **DIM**^{3}. These velocity field components are named **V_field_x**, **V_field_y**, **V_field_z** respectively. Similarly the scalar field representing the density of the fluid is represented by a single flattened vector **S_field**. The scalar field is represented in the graphical space by voxels with varying colour and opacity.

Indexing into the components of the velocity field and the scalar field is done through the use of an indexing function:

```
flat_index(i,j,k)=i + j*DIM + k*DIM*DIM
```

In addition, for each of the steps that we discuss, the steps are only applied to the inner cells of the discretized velocity field. For the cells on the walls of the simulation boundary we apply a fixed boundary constraint, where we set the velocity of the cell to be the negative velocity value of its closest neighbour.

3.1 Implementing Apply Force

The force field which is added onto the velocity field, is represented in the same manner as the velocity field. That is, it is represented by 3, 1D `Eigen::VectorXd` objects. The force field acting on the fluid is free to change at any time and can vary based on user interaction. However for our implementation we hard-coded the velocity field and had it decay in time by a factor of $\frac{1}{t+1}$, where t is the global time of the simulation.

3.2 Implementing Advection

To perform advection we loop through each cell in the discretized velocity field. We back-trace the position of the cell using forward euler [1]:

$$p_{t-\Delta t} = p_t - \Delta t\, v_p t$$

Where p_{t-1} is the backtraced position, p_t is the position of the cell, and $v_p(t)$ is the velocity defined at the cell at time t.

We then obtain the velocity at the backtraced position $p_{t-\Delta t}$ in the field using trilinear interpolation and then update velocity at the cell to be the backtraced velocity.

3.3 Implementing Diffusion

To implement the diffusion process we must express equation 5 as a problem of solving a linear system of equations. In order to do this we must represent the laplacian operator as a **3·DIM3** by **3·DIM3** sparse matrix ∇^2.

In order to discretize a component of the laplacian at a local cell (x, y, z) the following approximation is used.

$$= \frac{\partial^2\vec{w}(x,y,z)_x}{\partial x^2} + \frac{\partial^2\vec{w}(x,y,z)_x}{\partial y^2} + \frac{\partial^2\vec{w}(x,y,z)_x}{\partial z^2} \tag{8}$$

$$= \frac{\vec{w}_{-x} + \vec{w}_{-x} + \vec{w}_{-y} + \vec{w}_{+y} + \vec{w}_{-z} + \vec{w}_{+z} - 6\vec{w}_0}{h^2} \tag{9}$$

Where $\vec{w}_{-x} = \vec{w}(x-1, y, z)$, $\vec{w}_{+x} = \vec{w}(x+1, y, z)$. Similarly these are defined for values in the y and z directions. $\vec{w}_0$ denotes the unperturbed location $\vec{w}(x, y, z)$.

The first **DIM3** rows of the ∇^2 matrix represents the laplacian of the x component of the velocity field. In order to compute an element of the laplacian for the n-th cell, the approximation mentioned above is used. Coefficients of $\frac{1}{h^2}$ are set for cells corresponding to the cells which are adjacent to the n-th cell in 3D space. A coefficient of $-\frac{6}{h^2}$ is set at the location in the array corresponding to the n-th cell. When multiplied by the vector field's representation, this then yields the laplacian for the given component of the n-th cell.

The rows for the y and z components for the n-th cell appear on rows n+ **DIM3** and n+ **3·DIM3** respectively. The coefficients for these rows are also offset by **DIM3** and **2·DIM3** respectively.

Having computed the laplacian operation matrix we multiply it by t an subtract the whole operation matrix from an similarly sized identity matrix. A conjugate gradient solver is then used to solve the sparse system of linear equations.

$$(I - vt\nabla^2)\mathbf{w_3}(\vec{x}) = \mathbf{w_2}(\vec{x}) \tag{10}$$

Where both $\mathbf{w_2}$ and $\mathbf{w_3}$ are stacked vectors of the components of the vector field. Together these then form a vector of size **3·DIM3**.

3.4 Implementing Projection

In order to implement projection we must additionally discretize the divergence and the gradient operators. These are represented as

a **DIM3** by **3·DIM3** matrix ∇_{div} and a **3·DIM3** by **DIM3** matrix ∇_{grad} respectively. Both of these matrices are defined in such a way that when multiplied by a vector of the appropriate size, they perform the desired operation on the input vector. For ∇_{div} this input vector is a **3·DIM3** vector of stacked vector field components, and for ∇_{grad} this is simply a flattened scalar field of size **DIM3**.

The central difference is used for a more accurate approximation of the partial derivatives of a cell than could be provided by either a forward or backward difference approximation. The central difference is computed as follows for a 3D field F in the x direction.

$$\frac{\partial \mathbf{F}(x,y,z)}{\partial x} = \frac{\mathbf{F}(x+1,y,z) - \mathbf{F}(x-1,y,z)}{2h} \tag{11}$$

Where h is the difference in world space locations.

The divergence and gradient operators are implemented as follows:

3.4.1 Divergence. The n-th row of the ∇_{div} corresponds to the divergence of the n-th cell in the simulation domain. In order to sum all of the partial derivatives of the different components of the vector field $u(\vec{i,j},k)$, coefficients for 3 central difference approximations are placed in the n-th row of ∇_{div}. Since a stacked vector of x, y, and z components of the vector field are used for its representation, the central difference coefficients are placed a at an offset of **DIM3** units from each other. Coefficients for the x partial derivative are placed in the first **DIM3** elements of row n, and so on for the y and z components in the next **2 * DIM3** elements. All other values are set to 0. This is then repeated in subsequent rows for the rest of the cells in the simulation domain. See ∇_{div} below.

$$\nabla_{div} = \begin{bmatrix} -\frac{1}{2h} & \cdots & \frac{1}{2h} & \cdots & -\frac{1}{2h} & \cdots & \frac{1}{2h} & \cdots -\frac{1}{2h} & \cdots & \frac{1}{2h} \\ \vdots & \ddots & & & & & & & & \\ -\frac{1}{2h} & \cdots & \frac{1}{2h} & \cdots & -\frac{1}{2h} & \cdots & \frac{1}{2h} & \cdots -\frac{1}{2h} & \cdots & \frac{1}{2h} \end{bmatrix} \tag{12}$$

3.4.2 Gradient. When considering one cell in the simulation domain, its directional partial derivatives must be computed to form the gradient at that cell. All of the partial derivatives of the cells in the x direction correspond to the first DIM3 rows of the ∇_{grad}. The y and z partial derivatives then follow in the next 2 * DIM rows.

For partial derivatives of the form $\frac{\partial \mathbf{q}(i,j,k)}{\partial x_n}$, the column corresponding to $\mathbf{q}(i-1,j,k)$ and $\mathbf{q}(i+1,j,k)$ will contain values $-\frac{1}{2h}$ and $\frac{1}{2h}$ respectively (h representing the difference in input between the two cells). All other coefficients are set to 0. Thus when multiplied by a vector, a finite difference approximation of the partial derivative is placed in row n.

$$\nabla_{grad} = \begin{bmatrix} 0 & \cdots & -\frac{1}{2h} & \cdots & \frac{1}{2h} & \cdots & 0 \\ \vdots & \ddots & & & & & \\ 0 & \cdots & -\frac{1}{2h} & \cdots & \frac{1}{2h} & \cdots & 0 \end{bmatrix} \tag{13}$$

To complete the projection process, first we compute a vector $\vec{b} = \nabla \cdot \mathbf{w_3}$. We then solve $\nabla^2 \mathbf{q}$ with a conjugate linear solver.

Having solved for $\mathbf{q}$ we then compute the gradient of $\mathbf{q}$ by multiplying ∇_{grad} by $\mathbf{q}$. This is then used to compute the projected velocity vector $\mathbf{w_4}$ using the second projection equation provided in **Section 2**.

3.5 Moving Substances through fluid

To visualize a substance moving through a fluid we need to create a scalar field with the same size as the velocity field. In other words we create a scalar field of size DIM3. Each number in the scalar field represents how much substance there is at a particular cell. The higher the number, the darker the cell should be when visualized, and the lower the number, the lighter the cell should be when visualized.Updating the scalar field is similar to updating the velocity field and follows a similar set of steps for each time-stepas updating the velocity field.

3.5.1 Step 1: Adding densities. This step is the same as the adding force step for the velocity field except the "force" is represented by a field of scalars rather than a field of vectors. This intuitively corresponds to adding substances at various positions in the fluid.

3.5.2 Step 2: Advection. To advect the substance through the fluid we use the same advection code for the advection of the velocity field with itself mentioned earlier. However instead of advecting the velocity field with itself, we advect the scalar field with with the velocity field.

More specifically, we still use the velocity field to get the backtraced positions, but when doing trilinear interpolation we performed the interpolation on the scalar field for the backtraced position rather than on the velocity field.

3.5.3 Step 3: Diffusion. For the diffusion step we use the same code from the diffusion step for updating the velocity field. The only change that needed to be made is to update the laplace operator, where we convert the laplace operator matrix into one that works on a scalar field.

3.5.4 Step 4: Dissipation. The dissipation step replaces the projection step when updating a scalar field. This makes sense because there is no notion of divergence for a scalar field, and so no projection is needed. For the dissipation step we update the scalar values at each cell based on the following equation:

$$a(\mathbf{x}, t+\Delta t) = \frac{a(\mathbf{x},t)}{1+\Delta T \alpha}$$

where $a(x,t)$ represents the scalar value at position x for time t and α is the dissipation rate. This step effectively dissipates a substance in the fluid by making its value smaller over time.

4 RESULTS

We will now present some results of our simulation.

4.1 Two substances colliding

In the picture below we put densities at opposite ends of simulation "walls". We then add an external force pointing in the normal direction on each one of these walls that meets each other at the center.

The magnitude of the force at each z position varies according to a sine wave. We found that this sine wave variation in the force was necessary to create an interesting visual effect.

Figure 2: Two plumes of gas colliding.

After colliding the gas eventually expands and slowly fills up the simulation box.

Figure 3: Gas filing up the simulation box.

4.2 Velocity Field Swirling

Pictured below is a 30×30 simulation domain showing only the velocity vectors. Each velocity is coloured using a mapping from the 3D normalized direction of the velocity at that cell location. The x, y, and z channels are mapped to the R, G, and B channels respectively.

An upward force is then applied at the center of the simulation domain. Upon careful inspection of **4**, swirling can be seen on either sides of the image.

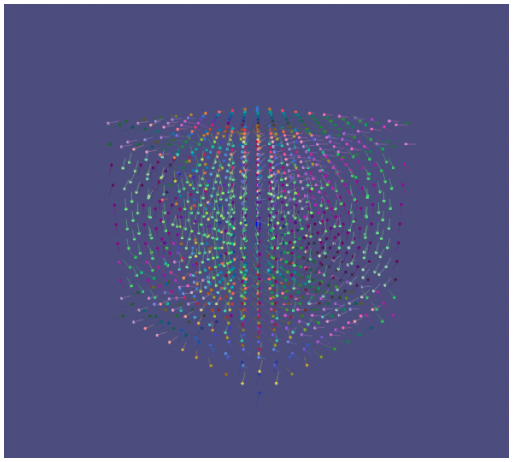

Figure 4: Swirling of the vector fields.

4.3 Downward plume

Below we can see a plume of lava like fluid flowing down from the ceiling of the simulation domain. Dissipation is set to 0.

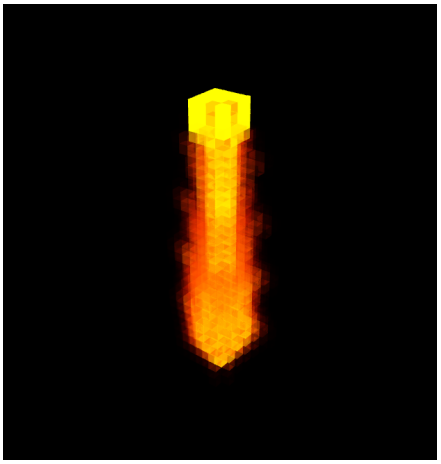

Figure 5: Downward Plume

REFERENCES

[1] Mark Harris. 2004. *Chapter 38. Fast Fluid Dynamics Simulation on the GPU.* https://developer.nvidia.com/gpugems/gpugems/part-vi-beyond-triangles/chapter-38-fast-fluid-dynamics-simulation-gpu

[2] Jos Stam. 1999. Stable Fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99).* ACM Press/Addison-Wesley Publishing Co., USA, 121–128. https://doi.org/10.1145/311535.311548