Fluid Sim Notes (still in progress!)

These notes are based on the following resources:

- Fluid Flow Tutorial by Karl Sims
- Coding Adventure: Simulating Smoke by Sebastian Lague
- Stable Fluids by Jos Stam

The Theory and Implementation

Implementation notes are written in *C#*.

Navier Stokes Equations for Incompressible fluids

Stable fluids assumes that the simulated fluid is uniformly viscous and non-compressible e.g its density is constant across the domain and time. Although real fluids are not incompressible, it is still a good approximation for simulating many real fluids such that water or gases which usually have low compressibility factor when the conditions such as pressure or temperature are not fluctuating too much.

Navier-Stokes equations are partial differential equations that are for this kind of fluid stated as

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = \frac{-\nabla p}{\rho} + \frac{\vec{F}_{\text{external}}}{\rho} + \nu \nabla^2 \vec{u}, \tag{1}$$

$$\nabla \cdot \vec{u} = 0, \tag{2}$$

where the terms are the following:

- $\vec{x}$ - let that be the position vector which in 2D is just $\vec{x} = (x, y)$.
- ρ - constant density of the fluid,
- ν - the viscosity constant (the rate of internal friction),
- $\vec{u} = \vec{u}(\vec{x}, t)$ - the vector field of velocities,
- $p = p(\vec{x}, t)$ - the scalar-valued pressure field,
- $\vec{F}_{\text{external}}$ - vector field of external forces such as gravity, bouyant forces or other forces.

The NS equations (1) - (2) say the following:

$$\underbrace{\frac{\partial \vec{u}}{\partial t}}_{\text{the acceleration}} + \underbrace{(\vec{u} \cdot \nabla)\vec{u}}_{\text{self-advection}} = \underbrace{\frac{-\nabla p}{\rho}}_{\text{the pressure gradient}} + \underbrace{\frac{\vec{F}_{\text{external}}}{\rho}}_{\text{applied external force}} + \underbrace{\nu \nabla^2 \vec{u},}_{\text{the viscous force}}$$

$$\underbrace{\nabla \cdot \vec{u} = 0,}_{\text{zero-divergence condition}}$$

which in words can be stated as the acceleration of a fluid is equal to the sum of the negative pressure gradient, external forces and the viscosity (e.g the internal friction) and the fluid is contiuous. The zero-divergence condition that fluid is continuous e.g that in the simulation there are no unintentional sources or sinks of fluids (these will be intentionally programmed).

Let the time be fixed ($t = \text{const.}$). Then let $\vec{u}(x, y) = (u_x(x, y), u_y(x, y))$ is a vector field with a vector argument representing vector field of velocities at that fixed time, and let $p(x, y)$ be a scalar field representing pressures of the fluid at that time. The vector calculus operators that are used in NS equations are the gradient of a scalar field which is a vector of directional derivatives in each basis vector direction

$$\nabla f(x, y) = \begin{pmatrix} \frac{\partial \vec{V}}{\partial x} \\ \frac{\partial \vec{V}}{\partial y} \end{pmatrix}, \tag{3}$$

the divergence of a vector field is a scalar field which is a dot product between a nabla operator vector (for 2D $\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$) with a vector vector field

$$\nabla \cdot \vec{u}(x, y) = \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y}, \tag{4}$$

and a $\nabla^2 \vec{u}$ is a Laplacian of a velocity field produces which is in explicit form

$$\nabla^2 \vec{u} = \left(\frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2}, \frac{\partial^2 u_2}{\partial x^2} + \frac{\partial^2 u_2}{\partial y^2} \right). \tag{5}$$

Although these are analytical expressions that can be applied for analytically defined functions, the evolution of fluids can almost never be analytically defined especially in simulations where the fluid is represented discretely. Therefore these oprators aree applied on discretely defined scalar fields using the finite difference methods. For that, the fields $\vec{u}(x, y)$ and $p(x, y)$ must first be discretized.

Discrete Representation of Variables and Operators

Discretization Procedure for a Simple Scalar Field

Let's say we want to discretely evaluate (solve) in a rectangular spatial domain R and time $t > 0$ some type of function $f(x, y, t)$ which is governed by some partial differential equation. Let R be defined as

$$R = \{(x, y) \in \mathbb{R} |\ 0 < x < a, 0 < y < a\}, \quad a = \text{const.} > 0.$$

In finite-difference methods for solving PDE-s in 2D, the spatial domain is usually discretized by some fixed smallest spatial step for all coordinates as $\Delta x = h_x > 0, \Delta y = h_y > 0$. Time is also discretized to consist of smallest possible timesteps $\Delta t > 0$. To assure the integer number of smallest spaces in a domain R, the smallest steps are defined by the integer

numbers of cells in each direction. Let that be an integer constant $N > 0$ for both axes. Then the following can be stated about the spatial discretization:

$$\Delta x = \Delta y = \frac{a}{N}$$

and the position vector $\vec{x} = (x, y)$ coordinates in the domain can be expressed for some integer $0 \leq i \leq N$ as

$$\begin{aligned} x_i &= i\Delta x, \\ y_i &= i\Delta y, \end{aligned}$$

therefore there are $N + 1$ nodes from $n = 0$ to $n = N$ where a scalar field $f(x, y) = f(x_i, y_j)$ can be evaluated therefore all the $(N + 1)(N + 1)$ values of $f(x_i, y_i)$ can be represented as a matrix

$$f_{ij} = f(x_j, y_i) = \begin{pmatrix} f(x_0, y_0) & f(x_1, y_0) & \dots & f(x_n, y_0) \\ f(x_0, y_1) & f(x_1, y_1) & \dots & f(x_n, y_0) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_0, y_n) & f(x_1, y_n) & \dots & f(x_n, y_n) \end{pmatrix}. \tag{6}$$

Discretization Procedure for a Velocity Field

In computational fluid dynamics, the pressure is (implicitly) discretized such as a scalar field in equation (6). However, to ensure the numerical ability to create 0-divergence, a staggered grid will be used to map velocity field components on the edges of cells that surround some pressure values. Therefore, if pressure for example takes on a matrix $(p)_{i,j}$ with dimensions $(N_y) \times (N_x)$ in the form

$$(p)_{i,j} = \begin{pmatrix} p_{00} & p_{01} & \cdots & p_{0,N_x-2} & p_{0,N_x-1} \\ p_{10} & p_{11} & \cdots & p_{1,N_x-2} & p_{1,N_x-1} \\ \dots & \dots & \ddots & \vdots & \\ p_{N_y,0} & p_{N_y,1} & \cdots & p_{N_y-2,N_x-2} & p_{N_y-2,N_x-1} \\ p_{N_y+1,0} & p_{N_y+1,1} & \cdots & p_{N_y-1,N_x-2} & p_{N_y-1,N_x-1} \end{pmatrix} \in \mathbb{R}^{N_y \times N_x},$$

then the corresponding velocity fields $(u_x)_{k,l}$ and $(u_y)_{m,n}$ will be created such that each pressure value $p_{ij} = (p)_{i,j}$ will have two vertical neighbors that correspond to the vertical velocities $(u_y)_{i,j}$ its cell and $(u_y)_{i+1,j}$ below, and two horizontal neighbors that correspond to the horizontal velocities $(u_x)_{i,j}$ from the left and $(u_x)_{i,j+1}$ from the right.

Therefore

$$\begin{aligned} (u_x)_{k,l} &\in \mathbb{R}^{N_y \times (N_x+1)}, \\ (u_t)_{m,n} &\in \mathbb{R}^{(N_y+1) \times N_x}. \end{aligned}$$

```
public class FluidGrid{
    //number of cells in each direction on a rectangular grid
```

```
    public readonly int Nx;
    public readonly int Ny;
    public readonly float h; //"infinitesimal" value dx=dy=h

    public readonly float[,] u_x;
    public readonly float[,] u_y;

    //initialization method
    public FluidGrid(int cellCountX, int cellCountY, float delta)
    {
        Nx = cellCountX;
        Ny = cellCountY;
        h = delta;

        //intialize velocity fields
        u_x = new float[N_y,N_x+1];
        u_y = new float[N_y+1, N_y];
    }
}
```

Update Loop of the Stable Fluids Algorithm

The simulation will start with initial conditions $\vec{u}(\vec{x},0)=\vec{u}_0$. Let the solution of the algorithm after some time t be $\vec{u}(\vec{x},t)=\vec{w}_0(\vec{x})$.Then the update loop of the Stable Fluids simulation algorithm proceeds as follows:

$$\vec{w}_0(\vec{x}) \overbrace{\rightarrow}^{\text{add force}} \vec{w}_1(\vec{x}) \overbrace{\rightarrow}^{\text{advect}} \vec{w}_2(\vec{x}) \overbrace{\rightarrow}^{\text{diffuse}} \vec{w}_3(\vec{x}) \overbrace{\rightarrow}^{\text{project}} \vec{w}_4(\vec{x}). \tag{7}$$

The first step will be the **addition of external force**:

$$\vec{w}_1(\vec{x})=\vec{w}_0(\vec{x})+\Delta t\vec{f}(\vec{x},t),$$

where $\vec{f}$ is some external force field normalised by the density.

The second step will be the **advection** which in numerical implementations is simply shifting the velocity field values along the velocity field lines by some small amount using *the method of characteristics*:

$$\vec{w}_2(\vec{x}) \leftarrow \vec{w}_1(\vec{p}(\vec{x},t-\Delta t)).$$

For the next step **viscous diffusion** will be implemented which is equivalent to a diffusion equation

$$\frac{\partial \vec{w}_2}{\partial t}=\nu\nabla^2\vec{w}_2$$

which will be solved using standard implicit methods that result in a sparse linear equation

$$(\mathbf{I}-\nu\Delta t\nabla^2)\vec{w}_3(\vec{x})=\vec{w}_2(\vec{x}).$$

As the last step in the calculation rule (7) the **projection** occurs. The resulting $\vec{w}_3$ gained diverging part ∇q in addition to the desired field component that contributes the curl $\vec{w}_3$ in the form

$$\vec{w}_3 = \vec{w}_4 + \nabla q,$$

where q is a scalar field that obeys a Poisson equation

$$\nabla^2 q = \nabla \cdot \vec{w}_3.$$

A solution of this equation in the domain R with neumann boundary condition $\frac{\partial q}{\partial n} = 0$ will be used to derive the projection for $\vec{w}_4$ in the form

$$\vec{w}_4 = \mathbf{P}\vec{w}_3.$$

Update Loop Functions in Code - TODO

```
public void Awake(){
    //initialization process
}

public void Update(){
    //order of steps proposed by the Stable Fluids paper
    //there are other kinds of which I'll look into (multiple times?)
    //arguments and definitions of the methods will be added next commit!
    addForces();
    advect();
    diffuse();
    project();
}
```