电子科技大学信息与软件工程学院

实 验 指 导 书

（实验）课程名称 图形与动画 II

电子科技大学教务处制表

目　　录

实验六 流体模拟

一、实验目的

1. 掌握流体模拟（欧拉视角）中的基本思路；
2. 掌握 Stam 流体模拟方法。

二、实验原理

(一) 流体方程

模拟流体运动的关键是找到流体在任意时刻其状态的数学表示。而在描述流体状态的众多参数中，速度则是一个最重要的参数，因为根据速度可以知道流体的运动方向。

流体的速度会随着时间和空间发生变化，对于这样一类随着时间和空间发生变化的量，常用矢量场来表征。矢量场是矢量函数在一个参数化空间上的映射。流体速度表征为**速度场**。具体来说，为了模拟一块矩形区域内流体运动，首先将该区域分成 $N \times N$ 个独立的网络单元（如图 6 - 所示），然后对网格单元中的流体状态进行计算。假设单个网格单元内部流体状态是一致的，即流体密度、速度相同。在渲染过程中，每个网格单元内的密度值将确定气体的可见性。我们用网格中心点状态代表整个网格状态。我们的目标是求得在任意时刻 t，每个方格中心位置$\boldsymbol{x} = (x, y)$处流体的速度场$\boldsymbol{u}(\boldsymbol{x}, t)$。请注意这里速度场 $\boldsymbol{u}$ 是一个与时间有关的矢量，$\boldsymbol{u}(\boldsymbol{x}, t) = (u(\boldsymbol{x}, t), v(\boldsymbol{x}, t))$。

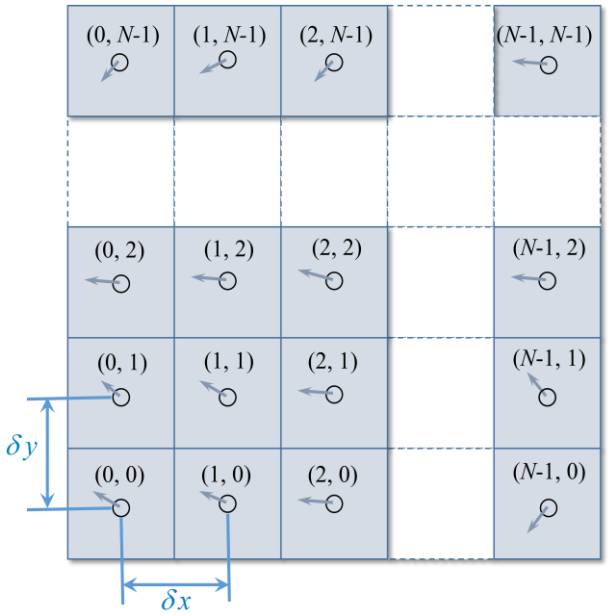


图 6 - 1 流体速度网格（箭头表示流体速度方向）

模拟流体运动的关键是在每一个时间步内，找到每个网格内流体的速度场。这样就能模拟各种各样有趣的流体现象，如烟雾密度的变化。那么如何描述流体运动呢？当然要借助流体力学中著名的Navier-Stokes方程。

1. Navier-Stokes 方程

Navier-Stokes流体动力学方程假设流体是不可压缩(体积不变)和均匀(密度不变)的。不可压缩是指流体的体积在任意时刻保持不变；均匀流体是指流体密度在空间任意位置不变。那么不可压缩(体积不变)和均匀(密度不变)的流体就意味着流体密度在时间和空间范围内保持不变。这是流体动力学中普遍的假设，适用于液体、气体等流体的模拟。

Navier-Stokes流体动力学方程描述流体状态变化如下：

$$\begin{aligned}\frac{\partial \boldsymbol{u}}{\partial t} &= -(\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\boldsymbol{u} + \boldsymbol{F} \\ \nabla\cdot\boldsymbol{u} &= 0\end{aligned} \qquad (6\text{-}1)$$

$\boldsymbol{u}$ - 流体在时刻 t，空间点 $\boldsymbol{x}=(x,y)$ 处速度， $\boldsymbol{u}(\boldsymbol{x},t)$ 是一个矢量场；

p - 流体在时刻 t，空间点 $\boldsymbol{x}=(x,y)$ 处压力， $p(\boldsymbol{x},t)$ 是一个标量场；

ρ - 流体密度。对于水来说 $\rho\approx 1000\ kg/m^3$；对于空气来说， $\rho\approx 1.3\ kg/m^3$；

ν - 动力黏度系数。动力黏度系数衡量了流体的粘稠度，即流体流动时抵抗形变的能力，更直观的说，衡量了搅拌流体的难度。通常蜂蜜具有较高的粘稠度，酒精具有较低的粘稠度。

$\boldsymbol{F}$ - $\boldsymbol{F}=(f_x,f_y)$ 表示作用于流体的外部力。

方程组（6-1）中第一个方程称为动量方程，衡量了流体速度场的变化；第二个方程称为不可压缩性条件，该约束条件保证了流体的不可压缩性，即体积不变。从方程组（6-1）看出，流体由它的矢量速度场 $\boldsymbol{u}(\boldsymbol{x},t)$和标量压力场 $p(\boldsymbol{x},t)$表示，这些量随着时间和空间的变化而变化。

由于速度场是一个矢量，那么方程组（6-1）中第一个方程实际是由两个方程构成：

$$\begin{aligned}\frac{\partial u}{\partial t} &= -(\boldsymbol{u}\cdot\nabla)u - \frac{1}{\rho}\nabla p + \nu\nabla^2 u + f_x \\ \frac{\partial v}{\partial t} &= -(\boldsymbol{u}\cdot\nabla)v - \frac{1}{\rho}\nabla p + \nu\nabla^2 v + f_y\end{aligned} \qquad (6\text{-}2)$$

公式（6-1）乍看之下非常复杂，接下来我们将其分解为几个部分，分别进行解释。

1） 平流项（Advection）

流体的运动使得流体可以传送物体，甚至包括密度、温度等。平流项比较难理解，为了方便理解可以想象将染料注入流动的液体中，那么染料将随着液体的运动而运动（传送）；同样烟雾随着空气流动而运动。方程（6-1）右边第一项就表示速度场随着自身的运动而运动，因此也被称作自身平流（self-advection），简称平流项。

2） 压力项（Pressure）

假如将流体内部看作由一个个流体分子构成，流体分子能够运动，流体分子之间相互运

动势必造成挤压和碰撞。当流体内部压力不均衡时，压力高的区域推动流体分子运动到压力低的区域，那么流体分子在压力作用下产生加速度。$-\frac{1}{\rho}\nabla p$ 表示受到压力时的加速度，称为压力项。梯度算子 $-\nabla p$ 方衡量了某点处的压力差。方程（6-1）右边第二项为压力项。

3） 扩散项（Diffusion）

根据生活经验，我们知道越粘稠的液体流动越缓慢，如蜂蜜比水流动慢很多。这是因为流体内部有一种力量在抵抗流体的形变，越粘稠的液体抵抗形变的能力越强，因而它的流动更缓慢。黏度系数（viscosity）衡量流体抵抗形变的程度。方程（6-1）第三项扩散项 $\nu\nabla^2\boldsymbol{u}$ 就是在衡量这个力，简单来说，此项可以看做流体内部的一种力量使得流体分子以周围分子的平均速度运动，以减少与周围分子速度之间的差异。这个力量造成动量的扩散，由此引起速度扩散，因此被称为扩散项。拉普拉斯算子 $\nabla^2=\nabla\cdot\nabla$ 很好地衡量了一个量与其平均值之间的差别。

4） 外力项（External forces）

前面三项与流体内部所受力相关，外力项则表示施加到流体上的外力，如风力、重力、浮力等。

因此综合上面四项，方程组（6-1）中第一个方程衡量了流体速度场的变化。根据公式（6-1），若已知流体在时刻 $t = 0$ 时速度和压力，那么根据该方程可计算流体在任意时刻的状态。

2. 数学回顾

从上节讨论中，方程（6-1）中涉及了大量的矢量微积分的知识。在本小节我们对相关知识进行回顾。

在公式（6-1）中出现最多的是符号 ∇，即微分运算符。公式中的 ∇，$\nabla\cdot$ 和 ∇^2 分别对应了梯度（Gradient）、散度（Divergence）和拉普拉斯（Laplacian）算子。

在公式（6-1）中，∇p 梯度定义为：

$$\nabla p=\left(\frac{\partial p}{\partial x},\frac{\partial p}{\partial y}\right) \tag{6-3}$$

为了便于计算机处理，需要梯度的有限微分形式：

$$\nabla p_{i,j}=\left(\frac{p_{i+1,j}-p_{i-1,j}}{2\delta x},\frac{p_{i,j+1}-p_{i,j-1}}{2\delta y}\right) \tag{6-4}$$

其中，下标 i 和 j 表示网格在坐标系中离散的位置（如图 6 - 所示），$\boldsymbol{\delta x}$ 和 $\boldsymbol{\delta y}$ 分别是 x 和 y 方向上的网格间距。

散度定义为：

$$\nabla\cdot\boldsymbol{u}=\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y} \tag{6-5}$$

我们可以简单地理解运算$\nabla\cdot\boldsymbol{u}$为向量$\nabla=\left(\frac{\partial}{\partial x},\frac{\partial}{\partial y}\right)$和向量$\boldsymbol{u}=(u,v)$的点积。散度具有重要的物理意义，它衡量着流体表面处速度的变化。流体散度为 0 意味着单位体积内流体流入量和流出量相等，这样能够保持单位体积内密度不变，也就是方程(6-1)的假设。另外，从公式（6-3）和（6-5）注意到梯度运算的结果是矢量场，而散度运算的结果是一个标量场。

散度$\nabla\cdot\boldsymbol{u}$的有限微分形式为：

$$\nabla\cdot u_{i,j}=\frac{u_{i+1,j}-u_{i-1,j}}{2\delta x}+\frac{v_{i,j+1}-v_{i,j-1}}{2\delta y} \tag{6-6}$$

拉普拉斯算子定义为：

$$\nabla^2 p=\frac{\partial^2 p}{\partial x^2}+\frac{\partial^2 p}{\partial y^2} \tag{6-7}$$

拉普拉斯算子$\nabla^2 p=\nabla\cdot\nabla p$可以简单理解为向量$\nabla=\left(\frac{\partial}{\partial x},\frac{\partial}{\partial y}\right)$和$\nabla p$的点积，也就是把散度运算应用于梯度运算的结果上。其有限微分形式为：

$$\nabla^2 p_{i,j}=\frac{p_{i+1,j}-2p_{i,j}+p_{i-1,j}}{(\delta x)^2}+\frac{p_{i,j+1}-2p_{i,j}+p_{i,j-1}}{(\delta y)^2} \tag{6-8}$$

若$\delta x=\delta y$，那么上式可化简为：

$$\nabla^2 p_{i,j}=\frac{p_{i+1,j}+p_{i-1,j}+p_{i,j+1}+p_{i,j-1}-4p_{i,j}}{(\delta x)^2} \tag{6-9}$$

拉普拉斯算子在物理学中应用非常普遍，其中最著名的是热传导方程。对于形如$\nabla^2\boldsymbol{x}=b$的方程称为泊松(Poisson)方程。$b=0$时，上述方程称为拉普拉斯方程。

(二) 求解流体方程

Navier-Stokes 方程很好的描述的流体的运动状态变化，接下来根据计算机动画的需求求解该方程。这里，采用数值积分法递增的来求解该方程，方法如下：

已知满足方程组（6-1）的速度场$\boldsymbol{u}^{(0)}$

对于时间步$n=0,1,2,\ldots$

更新时间 $t_{n+1}=t_n+\Delta t$

根据前一时刻$\boldsymbol{u}^{(n)}$求得时刻t_{n+1}下速度场$\boldsymbol{u}^{(n+1)}$

接下来关键求解方程（6-1）。我们将使用 Stam 采用的稳定流体技术[1]，对方程右边四项分别计算。在求解之前，我们先介绍一种数学变换方法，通过该方法可以方便地求解。

1. Helmholtz-Hodage 分解定律

基本矢量微积分学中提到任何矢量 $\boldsymbol{v}$ 能被分解为一系列基本矢量的和。如在笛卡尔坐标系下，矢量 $\boldsymbol{v} = (x, y)$可以表示成 $\boldsymbol{v} = x\,(1,0) + y\,(0,1)$形式。同样的，我们不但可以将矢量分解成基本矢量和的形式，还可以将矢量场分解为矢量场和的形式。假设 D 是空间内的某一区域，该区域边界平滑（即 ∂D 可求导），表面法线为 $\boldsymbol{n}$。根据 Chorin 和 Marsden1993 年提出的 Helmholtz-Hodage 分解定律[2]，我们可以将区域 D 上的矢量场 $\boldsymbol{w}$ 唯一地分解为：

$$\boldsymbol{w} = \boldsymbol{u} + \nabla p \tag{6-10}$$

其中，$\nabla \cdot \boldsymbol{u} = 0$ 即 $\boldsymbol{u}$ 是散度为 0 的矢量场并且 $\boldsymbol{u} \cdot \boldsymbol{n} = 0$ 即 $\boldsymbol{u}$ 与 ∂D 平行。关于该定律详细证明参见 Chorin 和 Marsden 在文献[2]中的第 1.3 节。

该定律指出任何的矢量场都可以分解为两个矢量场之和：

- 无散度矢量场；
- 标量场的梯度。

利用该定律，我们可以得到如下计算：

1) 根据公式（6-1）计算新的速度矢量场，假设为 $\boldsymbol{w}$。但是 $\boldsymbol{w}$ 并不一定满足（6-1）的约束条件，即散度为 0。为了得到散度为 0 的速度场，利用 Helmholtz-Hodage 分解定律解决该问题。将 $\boldsymbol{w}$ 减去压力场的梯度就可以得到满足条件的速度场 $\boldsymbol{u}$。

$$\boldsymbol{u} = \boldsymbol{w} - \nabla p \tag{6-11}$$

2) 在得到速度场 $\boldsymbol{w}$ 之前，我们还需要利用该定律计算压力场 p。将散度算子作用于方程（6-10）两边：

$$\nabla \cdot \boldsymbol{w} = \nabla \cdot \left(\boldsymbol{u} + \nabla p\right) = \nabla \cdot \boldsymbol{u} + \nabla^2 p \tag{6-12}$$

由于约束条件（6-1）规定 $\nabla \cdot \boldsymbol{u} = 0$，上式简化为：

$$\nabla^2 p = \nabla \cdot \boldsymbol{w} \tag{6-13}$$

上式（6-13）是一个 Poisson 方程式，也称为泊松压力（Poisson-pressure）方程。

根据 Helmholtz-Hodage 分解定律，我们首先计算得到更新后的 $\boldsymbol{w}$，带入方程（6-13）计算得到压力 p，然后将 $\boldsymbol{w}$ 和 p 都带入（6-11），求得到新的散度为 0 的速度场 $\boldsymbol{u}$。

2. 投射（Project）

从上面分析可以看出，要求得满足方程（6-1）条件的速度场 $\boldsymbol{u}$，需要首先计算得到 $\boldsymbol{w}$。在利用公式（6-1）计算 $\boldsymbol{w}$ 之前，我们希望对该公式进行简化。

根据点积的定义，两个向量的点积 $\boldsymbol{r} \cdot \boldsymbol{s}$（$\boldsymbol{s}$ 为单位向量）可以看做向量 $\boldsymbol{r}$ 在单位向量 $\boldsymbol{s}$ 上的投影。受此启发，我们使用 Helmholtz-Hodage 分解定律定义一个投射运算符 $\mathbf{P}$，将矢量

场 $\boldsymbol{w}$ 投射到其无散度分量 $\boldsymbol{u}$ 上。将 $\mathbf{P}$ 应用到方程(6-10)上：

$$\mathbf{P}\boldsymbol{w} = \mathbf{P}\boldsymbol{u} + \mathbf{P}(\nabla p) \tag{6-14}$$

根据 $\mathbf{P}$ 的定义，$\mathbf{P}\boldsymbol{w} = \mathbf{P}\boldsymbol{u} = \boldsymbol{u}$，因此 $\mathbf{P}(\nabla p) = 0$。根据此结果，简化(6-1)。将 $\mathbf{P}$ 应用到（6-1）两端：

$$\mathbf{P}\left(\frac{\partial \boldsymbol{u}}{\partial t}\right) = \mathbf{P}\left(-(\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\boldsymbol{u} + \boldsymbol{F}\right) \tag{6-15}$$

速度场 $\boldsymbol{u}$ 散度为 0，因此其导数也是无散度的，那么 $\mathbf{P}\left(\frac{\partial \boldsymbol{u}}{\partial t}\right) = \frac{\partial \boldsymbol{u}}{\partial t}$。另外，$\mathbf{P}(\nabla p) = 0$，去掉压力项，得到新的方程式：

$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathbf{P}\left(-(\boldsymbol{u}\cdot\nabla)\boldsymbol{u} + \nu\nabla^2\boldsymbol{u} + \boldsymbol{F}\right) \tag{6-16}$$

从上述公式可以得到整个模拟流体的算法：首先计算方程右边括号内部三项。从左至右，分别为平流项、扩散项和外力项。计算完后得到一个散度非零的速度场 $\boldsymbol{w}$，对其应用投射运算符 $\mathbf{P}$ 得到一个散度为 0 的速度场 $\boldsymbol{u}$。具体来说，求解方程（6-13）得到 p，然后根据方程(6-11)从 $\boldsymbol{w}$ 减去梯度 p。

在实际对方程（6-16）计算中，并非计算完右边每个分量后再把结果相加，而是通过状态上的变化的合成来实现的。每个分量的计算都是一个步骤，输入为一个速度场，输出为一个新的速度场。具体来说，采用如下计算步骤实施：

$$\mathbf{S}\left(\boldsymbol{u}^{(n+1)}\right) = \mathbf{P}\cdot\mathbf{F}\cdot\mathbf{D}\cdot\mathbf{A}\left(\boldsymbol{u}^{(n)}\right) \tag{6-17}$$

其中：

A - 计算平流项；
D - 计算扩散项；
F - 计算外力；
P - 投射运算。

请注意，（6-17）中我们采用的计算步骤从右至左，先计算平流项，计算结果作为扩散项输入，计算扩散结果，再考虑外力，最后对外力项输出结果进行投射。

到目前为止我们对于流体模拟的整个计算过程有了大致的了解。现在还剩下三个问题未解决：如何计算平流、扩散项？如何求解泊松方程？接下来分别解决这三个问题。

3. 平流（Advect）

平流是指流体速度的存在造成的流体本身以及流体中其他量的流动。平流项实际上表达了一个量 q 如何沿着速度场移动。我们将每个网格中抽象为一个粒子，假设粒子位置初始位于 P 点，坐标矢量为 $\boldsymbol{x}$，每个粒子在时间步 δt 内沿着速度场 $\boldsymbol{u}$ 向前移动，移动后的位置 P' 坐标为：$\boldsymbol{x} + \boldsymbol{u}(\boldsymbol{x}, t)\delta t$（如图 6 - 2 所示）。

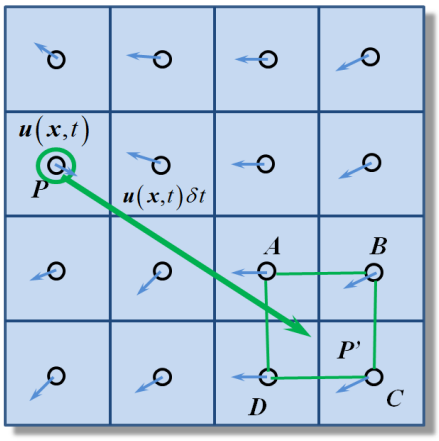


图 6 - 2 位于 *P'*点量由周围四个网格点处的量决定

流体在位置 *P* 经过δt后的量 *q*(速度、密度、温度或流体携带的任何量)等于其位于新位置 *P'*时刻 *t* 时的量$\boldsymbol{q}(\boldsymbol{x}+\boldsymbol{u}(\boldsymbol{x},t)\delta t,t)$。

$$q(\boldsymbol{x},t+\delta t)=q(\boldsymbol{x}+\boldsymbol{u}(x,t)\delta t,t) \tag{6-18}$$

$\boldsymbol{q}(\boldsymbol{x}+\boldsymbol{u}(\boldsymbol{x},t)\delta t,t)$等于 *P'*点周围紧邻的四个网格点 *A*、*B*、*C* 和 *D* 处量的插值。上述方法采用 Euler 方法更新物体位置坐标。但是该方法会产生问题，当时间步较大δt时，特别是$\boldsymbol{u}(\boldsymbol{x},t)\delta t$比单个网格单元的尺寸大时，模拟结果非常不稳定。

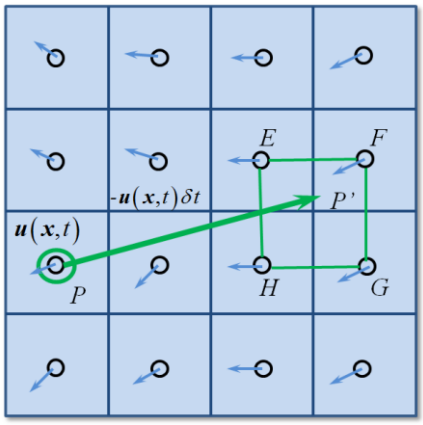


图 6 - 3 隐式的对流步骤是穿过速度场向后追踪，以确定是如何向前携带量的

一种解决方法是 Stam 提出的隐式积分法。不是通过计算粒子在当前时间步移动到哪儿来更新值，而是从每个网格单元逆时追踪粒子轨道，得到它以前的位置 *P'*，复制那个位置的量。如图 6 - 所示，要计算位置 *P* 处值，将沿此处速度方向向后追踪，找到新的位置 *P'*，此时 *P* 点处值为之前时刻 *P'*处值：

$$q(\boldsymbol{x},t+\delta t)=q(\boldsymbol{x}-\boldsymbol{u}(x,t)\delta t,t) \tag{6-19}$$

该方法对于任意时间段和任意速度值都是稳定的。*P'*处的量值由其周围四个网格点（*E*、*F*、*G* 和 *H*）处的值采用双线性插值求得。

下面简单介绍双线性插值的概念。双线性插值又称为双线性内插。在数学上，双线性插值是有两个变量的插值函数的线性插值扩展，其核心思想是在两个方向分别进行一次线性插值。为了方便理解双线性插值，先考虑一维情况下的线性插值(图 10 - 所示)：

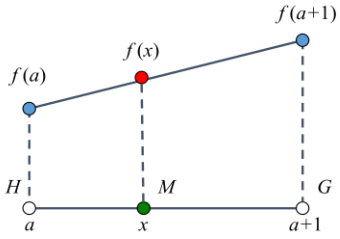


图 10 - 4 一维线性插值

已知直线上点 $H = a$ 处的函数值 $f(a)$和点 $G = a+1$ 处函数值 $f(a+1)$。假设函数值 $f(a)$到 $f(a+1)$之间是线性变化的。对于直线段 a 到 $a+1$ 点上任意一点 $M = x$ 所对应的函数值 $f(x)$ $(a \le x \le a+1)$为：

$$f(x) = f(a) + \frac{f(a+1) - f(a)}{a+1-a}(x-a) \tag{6-20}$$
$$\Leftrightarrow f(x) = f(a) \times (1+a-x) + f(a+1) \times (x-a)$$

接下来将上述公式扩展到二维空间。假设我们想得到未知函数 f 在点 $P' = (x,y)$处的函数值（如图 6 - 所示）。

已知 P'点附近四个点 E、F、G 和 H 处对应的函数值 $f(a, b+1)$、$f(a+1, b+1)$、$f(a+1, b)$和 $f(a, b)$ $(a \le x \le a+1,\ b \le y \le b+1)$。假设函数值在 x 和 y 方向是线性变化的。

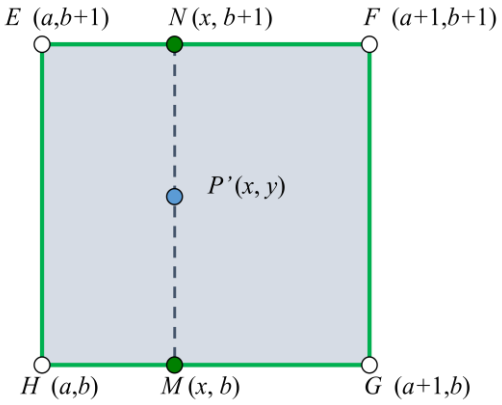


图 6 - 5 双线性插值

首先在 x 方向进行线性插值，分别求出点 $M = (x, b)$和点 $N = (x, b+1)$处函数值:

$$f(x,b) = f(a,b) \times (1+a-x) + f(a+1,b) \times (x-a) \tag{6-21}$$
$$f(x,b+1) = f(a,b+1) \times (1+a-x) + f(a+1,b+1) \times (x-a)$$

然后利用已获得的点 M 和点 N 处的函数值 $f(x, b)$和 $f(x, b+1)$，在 y 方向进行线性插值。假设 $f(x, b)$到 $f(x, b+1)$是线性变化的，那么再次利用一维线性插值求得位于点 M 和 N 之间的点 $P'(x, y)$处函数值:

$$f(x,y) = f(x,b)\times(1+b-y) + f(x,b+1)\times(y-b) \tag{6-22}$$

4. 扩散（Diffusion）

如前所述，粘性的液体对流动有着阻碍作用，这将造成速度的扩散。粘性扩散的偏微分方程为：

$$\frac{\partial \boldsymbol{u}}{\partial t} = v\nabla^2\boldsymbol{u} \tag{6-23}$$

与求解对流项一样，我们可以选择显式的 Euler 方法求解上式：

$$\boldsymbol{u}(\boldsymbol{x},t+\delta t) = \boldsymbol{u}(\boldsymbol{x},t) + v\delta t\nabla^2\boldsymbol{u}(\boldsymbol{x},t) \tag{6-24}$$

但是就像平流项计算时一样，Euler 法方法对于时间步和网格大小非常敏感，结果非常不稳定。因此，扩散项计算再次采用 Stam 的思想，使用隐式形式，他考虑这样一个问题：如果将扩散过程逆时追踪，什么速度场会产生当前速度场？于是得到如下计算公式：

$$\begin{aligned} &\boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{u}(\boldsymbol{x},t+\delta t) - v\delta t\nabla^2\boldsymbol{u}(\boldsymbol{x},t+\delta t) \\ &\Leftrightarrow (I - v\delta t\nabla^2)\boldsymbol{u}(\boldsymbol{x},t+\delta t) = \boldsymbol{u}(\boldsymbol{x},t) \end{aligned} \tag{6-25}$$

其中，I 是单位矩阵，该公式对任何时间步都是稳定的。

方程（6-25）形式是关于速度场的泊松方程。目前，我们已经得到两个泊松方程，分别为（6-13）泊松压力方程和（6-25）黏度扩散方程。泊松方程是一类常见于静电学、机械工程和理论物理的偏微分方程。通常可采用松弛迭代法，从一个近似的解开始，每迭代一次将更接近准确值。下面将介绍如何利用松弛迭代法求解泊松方程。

5. 泊松方程的解法

泊松方程可以通过迭代法求解。对于一个形如 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ 的方程，$\boldsymbol{x}$ 是需要求解的数值矢量（在方程（6-13）中对应 p，方程（6-25）中对应 $\boldsymbol{u}$），$\boldsymbol{b}$ 是一个常数的矢量，$\boldsymbol{A}$ 是一个矩阵，$\boldsymbol{A}$ 在拉普拉斯算子∇^2中隐式地表示。

迭代法首先从一个初值 $\boldsymbol{x}^{(0)}$ 开始，带入方程求得一个新的值 $\boldsymbol{x}^{(1)}$，不断重复，每一步 k 产生一个新的值 $\boldsymbol{x}^{(k)}$。我们以如下方程为例介绍迭代法原理：

$$ax = b,\ \ a \neq 0, 0 < a < 1$$

求解方程前，首先定义 $a = 1-c$，那么 $x = cx + b$ $(0< c < 1)$，给定一个初始值 $x^{(0)}$，那么

$$\begin{aligned} x^{(1)} &= cx^{(0)} + b \\ &\vdots \\ x^{(n)} &= cx^{(n-1)} + b \end{aligned}$$

经过若 n 次迭代， $x^{(n)}$ 接近真实值。

证明：假设 x^* 为解析解，则 $x^* = cx^* + b$ ，接下来我们考察 $\lim_{n\to\infty}\left(x^{(n)} - x^*\right)$：

$$\lim_{n\to\infty}\left(x^{(n)} - x^*\right) = cx^{n-1} + b - cx^* - b = c\left(x^{n-1} - x^*\right)$$
$$= c\left(cx^{n-2} + b - x^* - b\right) = c^2\left(x^{n-2} - x^*\right) = \cdots = c^n\left(x^0 - x^*\right)$$

由于 0< c < 1，则 $\lim_{n\to\infty}\left(x^{(n)} - x^*\right) = 0$ 。由此可见，通过迭代方式能够找到 x 的解析解，并且初始值 $x^{(0)}$ 选择无关。

经被证明, 对于矩阵表示的线性方程，迭代法对于方程 **A*x*=*b*** 求解照样可行。伯松压力方程, 就正好是这样的一类方程。对于一般矩阵方式方程 **A*x*=*b*** 的迭代推导可以在 Golub 和 Van Loan 于 1996 的文献[3]中找到。

利用方程（6-9）将 $\nabla^2 p$ 离散化，可求解（6-13），泊松压力方程迭代求解：

$$p_{i,j}^{(k+1)} = \frac{\left(p_{i+1,j}^{(k)} + p_{i-1,j}^{(k)} + p_{i,j+1}^{(k)} + p_{i,j-1}^{(k)}\right) - (\delta x)^2 \nabla \cdot \boldsymbol{w}_{i,j}}{4} \tag{6-26}$$

（6-25）迭代求解：

$$\boldsymbol{u}_{i,j}^{(k+1)} = \frac{\boldsymbol{u}_{i+1,j}^{(k)} + \boldsymbol{u}_{i-1,j}^{(k)} + \boldsymbol{u}_{i,j+1}^{(k)} + \boldsymbol{u}_{i,j-1}^{(k)} + \dfrac{(\delta x)^2}{\nu\delta t}\boldsymbol{u}_{i,j}}{4 + \dfrac{(\delta x)^2}{\nu\delta t}} \tag{6-27}$$

仔细观察（6-26）和（6-27）具有相似的形式：

$$x_{i,j}^{(k+1)} = \frac{\left(x_{i+1,j}^{(k)} + x_{i-1,j}^{(k)} + x_{i,j+1}^{(k)} + x_{i,j-1}^{(k)}\right) + \alpha b_{i,j}}{\beta} \tag{6-28}$$

这里α和β是常数。在泊松压力方程中，x 代表 p，b 代表$\nabla \cdot \boldsymbol{w}$，$\alpha = -(\delta x)^2$，$\beta = 4$。对于黏度扩散公式，$x$ 和 b 代表 $\boldsymbol{u}$，$\alpha = (\delta x)^2/\nu\delta t$，$\beta = 4 + \alpha$。采用共同的形式目的是使用相同的函数来求解这两个方程，在后面章节中将会看到。

6. 初始条件和边界条件

对于流体模拟初始条件，如流体初始速度、压力等值，可根据具体模拟场景设定。在这里需要重点讨论流体模拟的边界条件。

如果流体是在一个封闭的方格内流动，那么流体需要满足不穿越(no-slip)边界条件和 Neumann 边界压力条件。不穿越边界条件规定流体在边界上的速度等于 0，而 Neumann 边界压力条件要求在边界的压力法向导数为 0。边界规定位于最外围单元格和最近的内部单元之间的边上（如图 10 - 6 所示）。因此，与边界相关的有两个单元格。通过两个单元格求平均值才能得到边界值。

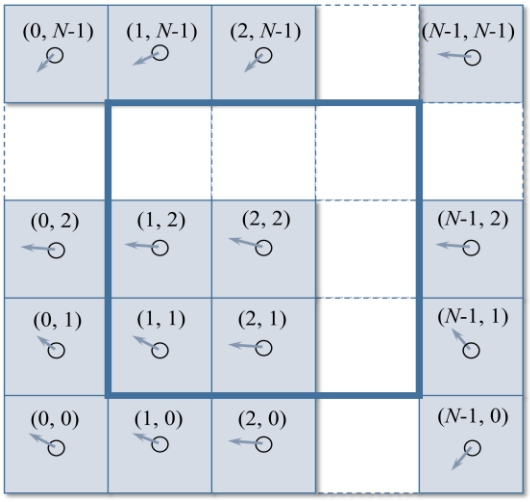


图 10 - 6 封闭方格的边界（深色粗线所示）

以左边界为例，左边边界上的速度必须满足如下条件：

$$\frac{\boldsymbol{u}_{0,j}+\boldsymbol{u}_{1,j}}{2}=0,\quad j\in[0,N-1] \tag{6-29}$$

为了满足边界条件，必须设定边界处的速度场 $\boldsymbol{u}_{0,j}$ 等于-$\boldsymbol{u}_{0,j}$。

对于边界处压力，需要满足边界处压力沿着法线变化为 0：

$$\frac{\partial p}{\delta n}=0 \tag{6-30}$$

采用求导的前向微分求得上式的近似值：

$$\frac{p_{1,j}-p_{0,j}}{\delta x}=0 \tag{6-31}$$

7. 扩展

获得流体速度场的结果并不能在视觉上表达出来。流体可见的原因在于通过它的运动造成其他物质（如烟雾等）的运动。对于烟雾这类质量较轻的物体将完全随着流体速度场运动而运动。在计算机动画中，采用粒子系统的方式模拟烟雾非常消耗计算资源，取而代之的方法同速度场一样。将空间结构分成若干个网格，每个网格内部烟雾密度均匀，密度值通常选取在 0 到 1 之间，0 代表没有烟雾，1 代表浓度最大值。烟雾密度$\boldsymbol{\rho}$和时间相关，其状态更新方程为：

$$\frac{\partial\rho}{\partial t}=-(\boldsymbol{u}\cdot\nabla)\rho+\kappa\nabla^2\boldsymbol{u}+S \tag{6-32}$$

（6-32）和 Navier-Stokes 方程非常相似，不过（6-32）是线性的，而（6-1）是非线性的。方程右边三项说明密度改变是由三个因素造成的：

- $-(\boldsymbol{u}\cdot\nabla)\rho$ 表明密度应该随着速度场运动而改变；
- $\kappa\nabla^2\boldsymbol{u}$ 表明密度将以一定速率 $\boldsymbol{u}$ 扩散，κ 是密度扩散系数；

- S 表明密度值受到外界的影响。

同计算速度扩散一样，采用迭代法，计算密度扩散。

(三) 代码实现

在了解算法基本原理后，本节将重点介绍 Stam 在其文章《Real-Time Fluid Dynamics for Games》[4]中对于流体模拟的实现。

1. 网格结构

Stam用一个$(N+2)\times(N+2)$的网格来模拟流体运动。每个网格中心处定义了该网格内流体的速度场以及密度值。假设流体在一个封闭的盒子内部流动。最外层的网格作为边界条件。

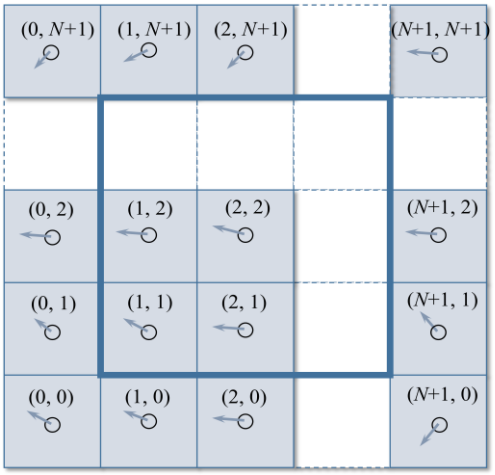


图 10 - 7 网格划分

2. 数据结构

Stam定义了两个大小为$(N+2)\times(N+2)$的数组用来存储网格中流体速度场以及密度值。出于计算效率的考虑，采用一维数组存储以上数据。

```
static float * u, * v, * u_prev, * v_prev;
static float * dens, * dens_prev;
```

其中：

- u和v为指向一维数组结构的指针，分别存储当前速度场分量值；
- u_pre和v_pre为指向一维数组结构的指针，分别存储前一时刻速度场分量值；
- dens为指向一维数组结构的指针，存储当前时刻密度值；
- dens_prev为指向一维数组结构的指针，存储前一时刻密度值。

函数allocate_data分配所需要的空间：

```
static int allocate_data ( void )
{
    int size = (N+2)*(N+2);

    u = (float *) malloc ( size*sizeof(float) );
```

```
    v = (float *) malloc ( size*sizeof(float) );
    u_prev = (float *) malloc ( size*sizeof(float) );
    v_prev = (float *) malloc ( size*sizeof(float) );
    dens = (float *) malloc ( size*sizeof(float) );
    dens_prev = (float *) malloc ( size*sizeof(float) );

    if ( !u || !v || !u_prev || !v_prev || !dens || !dens_prev ) {
        fprintf ( stderr, "cannot allocate data\n" );
        return ( 0 );
    }

    return ( 1 );
}
```

通过定义如下宏，可以根据索引值方便读取数组中元素数：

```
#define IX(i,j) ((i)+(N+2)*(j))
```

例如：为了读取当前时刻第 *i* 行和第 *j* 列中网格（*i*, *j*）中速度场第一分量值，可以简单通过 u[IX(i,j)]读取。

3. 程序结构

程序主要结构如下所示。首先函数get_from_UI利用用户点击鼠标的信息来初始化密度值和速度场，然后通过函数vel_step以及dens_step更新速度场与密度的状态，最后绘制密度。

```
static void idle_func ( void )
{
    get_from_UI ( dens_prev, u_prev, v_prev );
    vel_step ( N, u, v, u_prev, v_prev, visc, dt );
    dens_step ( N, dens, dens_prev, u, v, diff, dt );
    …
}
```

1） 更新速度场

更新速度场程序框架如图 6 - 所示。从图中可以看出，每个时间步需要计算外力项，然后计算扩散项和平流项：

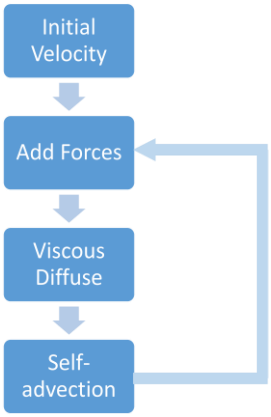


图 6 - 8 速度场更新步骤

具体实现通过函数 vel_step 实现：

```
void vel_step ( int N, float * u, float * v, float * u0, float * v0, float visc, float dt )
{
    add_source ( N, u, u0, dt ); add_source ( N, v, v0, dt );
    SWAP ( u0, u ); diffuse ( N, 1, u, u0, visc, dt );
    SWAP ( v0, v ); diffuse ( N, 2, v, v0, visc, dt );
    project ( N, u, v, u0, v0 );
    SWAP ( u0, u ); SWAP ( v0, v );
    advect ( N, 1, u, u0, u0, v0, dt ); advect ( N, 2, v, v0, u0, v0, dt );
    project ( N, u, v, u0, v0 );
}
```

函数 add_source 实现外力添加：

```
void add_source ( int N, float * x, float * s, float dt )
{
    int i, size=(N+2)*(N+2);
    for ( i=0 ; i<size ; i++ ) x[i] += dt*s[i];
}
```

完成第一步后，需要交互速度场两个分量u0、u和v0、v中的数值。

```
#define SWAP(x0,x) {float * tmp=x0;x0=x;x=tmp;}
```

SWAP是一个宏定义，交换了两个数组的指针。

接下来计算扩散项，扩散系数保存在变量diff中。根据公式（6-27）的推导，网格(i, j)中速度场与周围相邻的四个网格相关（如图6-9所示）。假设整个网格长度为1，每个网格长度为1/N，那么a=dt*diff*N*N。

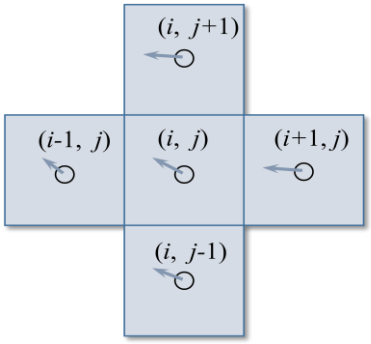


图 6 - 9 网格(i, j)中的量与周围相邻的四个网格交互

x[IX(i,j)] = (x0[IX(i,j)] + a*(x[IX(i-1,j)]+x[IX(i+1,j)]+x[IX(i,j-1)]+x[IX(i,j+1)]))/(1 + 4 * a)

扩散项由函数diffuse实现：

```
void diffuse ( int N, int b, float * x, float * x0, float diff, float dt )
{
    float a=dt*diff*N*N;
    lin_solve ( N, b, x, x0, a, 1+4*a );
}
```

Poisson 方程求解由通用函数 lin_solve 实现：

```
void lin_solve ( int N, int b, float * x, float * x0, float a, float c )
{
    int i, j, k;

    for ( k=0 ; k<20 ; k++ ) {
        FOR_EACH_CELL
            x[IX(i,j)] = (x0[IX(i,j)] + a*(x[IX(i-1,j)]+x[IX(i+1,j)]+x[IX(i,j-1)]+x[IX(i,j+1)]))/c;
        END_FOR
        set_bnd ( N, b, x );
    }
}
```

其中，FOR_EACH_CELL和END_FOR是宏定义，用于遍历所有网格。

```
#define FOR_EACH_CELL for ( i=1 ; i<=N ; i++ ) { for ( j=1 ; j<=N ; j++ ) {
#define END_FOR }}
```

函数set_bnd设置边界处值，将在后面讲解。

最后一步计算平流项，采用隐式积分法，逆时追踪粒子轨道，在新位置通过双线性插值求得平流项的值，具体实现如下：

```
void advect ( int N, int b, float * d, float * d0, float * u, float * v, float dt )
```

```
{
    int i, j, i0, j0, i1, j1;
    float x, y, s0, t0, s1, t1, dt0;

    dt0 = dt*N;
    FOR_EACH_CELL
        x = i-dt0*u[IX(i,j)]; y = j-dt0*v[IX(i,j)];
        if (x<0.5f) x=0.5f; if (x>N+0.5f) x=N+0.5f; i0=(int)x; i1=i0+1;
        if (y<0.5f) y=0.5f; if (y>N+0.5f) y=N+0.5f; j0=(int)y; j1=j0+1;
        s1 = x-i0; s0 = 1-s1; t1 = y-j0; t0 = 1-t1;
        d[IX(i,j)] = s0*(t0*d0[IX(i0,j0)]+t1*d0[IX(i0,j1)])+
                        s1*(t0*d0[IX(i1,j0)]+t1*d0[IX(i1,j1)]);
    END_FOR
    set_bnd ( N, b, d );
}
```

上面完成了外力计算、扩散项计算以及平流项计算，最后对结果进行投射。根据公式（6-26），投射由project函数实现，具体如下：

```
void project ( int N, float * u, float * v, float * p, float * div )
{
    int i, j;

    FOR_EACH_CELL
        div[IX(i,j)] = -0.5f*(u[IX(i+1,j)]-u[IX(i-1,j)]+v[IX(i,j+1)]-v[IX(i,j-1)])/N;
        p[IX(i,j)] = 0;
    END_FOR
    set_bnd ( N, 0, div ); set_bnd ( N, 0, p );

    lin_solve ( N, 0, p, div, 1, 4 );

    FOR_EACH_CELL
        u[IX(i,j)] -= 0.5f*N*(p[IX(i+1,j)]-p[IX(i-1,j)]);
        v[IX(i,j)] -= 0.5f*N*(p[IX(i,j+1)]-p[IX(i,j-1)]);
    END_FOR
    set_bnd ( N, 1, u ); set_bnd ( N, 2, v );
}
```

在project函数中，再次调用了lin_solve函数求解泊松压力方程（6-13），然后利用公式（6-11）求解出散度为0的速度场。

vel_step函数调用了两次project函数。这是因为如果能让速度场保持物质守恒，advect函数计算结果将会更准确。

最后还剩下更新边界处的速度场，由set_bnd函数实现：

```
void set_bnd ( int N, int b, float * x )
{
    int i;

    for ( i=1 ; i<=N ; i++ ) {
        x[IX(0  ,i)] = b==1 ? -x[IX(1,i)] : x[IX(1,i)];
        x[IX(N+1,i)] = b==1 ? -x[IX(N,i)] : x[IX(N,i)];
        x[IX(i,0  )] = b==2 ? -x[IX(i,1)] : x[IX(i,1)];
        x[IX(i,N+1)] = b==2 ? -x[IX(i,N)] : x[IX(i,N)];
    }
    x[IX(0  ,0  )] = 0.5f*(x[IX(1,0  )]+x[IX(0  ,1)]);
    x[IX(0  ,N+1)] = 0.5f*(x[IX(1,N+1)]+x[IX(0  ,N)]);
    x[IX(N+1,0  )] = 0.5f*(x[IX(N,0  )]+x[IX(N+1,1)]);
    x[IX(N+1,N+1)] = 0.5f*(x[IX(N,N+1)]+x[IX(N+1,N)]);
}
```

set_bnd函数中的整个for循环实现了不穿越(no-slip)边界条件（即在四条边界上的速度等于0）以及Neumann边界压力条件（即在边界的压力法向导数为0）。最后四条语句实现了对四个角点处速度场的更新（如图6-10所示）。

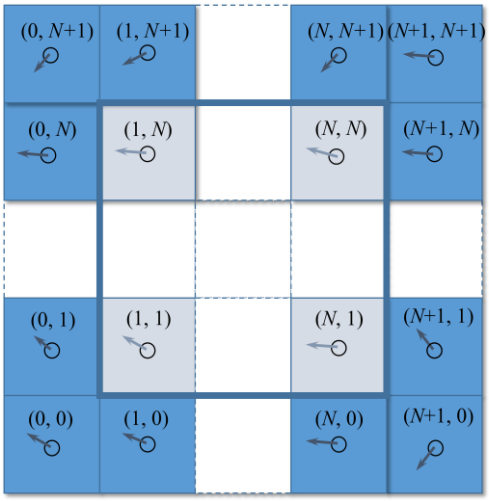


图 6 - 1 边界处速度场的更新

2） 更新密度值

根据密度更新方程(6-32),密度更新流程程序框架如图 6 - 所示。从图中可以看出，每个时间步需要计算新增加密度值项，然后计算密度的扩散项和以及密度随着速度场的移动：

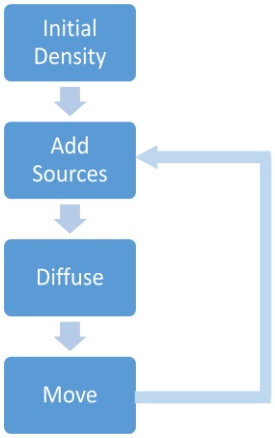


图 6 - 11 密度更新的基本架构

具体实现有 dens_step 函数实现：

```
void dens_step ( int N, float * x, float * x0, float * u, float * v, float diff, float dt )
{
    add_source ( N, x, x0, dt );
    SWAP ( x0, x ); diffuse ( N, 0, x, x0, diff, dt );
    SWAP ( x0, x ); advect ( N, 0, x, x0, u, v, dt );
}
```

计算新增加密度值仍然由 add_source 函数实现。add_source 函数利用外部源对于密度值进行修改，外部源密度存储在指针 x0 指向的数组中。

密度扩散项同速度场扩散项非常类似，因此通过 diffuse 函数模拟密度的扩散。密度扩散系数为 diff，当 diff ＞ 0 时，密度将会扩散到周围。每个网格只与其周围直接相邻的四个网格交换密度。网格内密度值随着密度扩散至相邻网格而降低，随着相邻网格密度流入而增加。

密度平流项同速度场平流项类似，因此也通过调用 advect 函数实现密度的对流项计算。

(四) 参考文献

[1] Stam, J. Stable Fluid. Proceedings of SIGGRAPH, 1999.

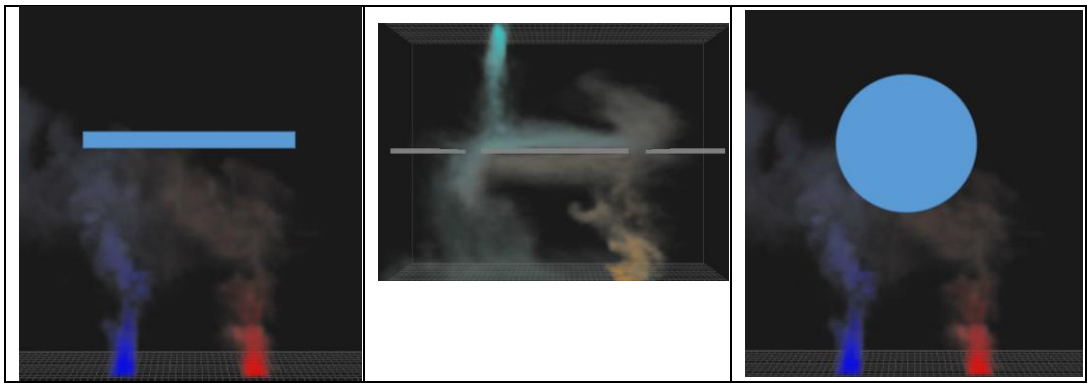
[2] Chorin, A.J., and J.E. Marsden. A Mathematical Introduction to Fluid Mechanics. 3rd ed. Springer, 1993.

[3] Colub, G.H., and C.F. Van Loan. Matrix Computations, 3rd ed. The Johns Hopkins University Press, 1996.

Stam, J. Real-Time Fluid Dynamics for Games. Proceedings of the Game Developer Conference, March 2003.

三、实验内容

1. 深入理解流体动画模拟原理以及 Stam 代码；
2. 在 Stam 代码中烟雾添加颜色，通过鼠标或者键盘控制颜色变化；
3. 在 Stam 代码中添加障碍物，可参考如下几种形式：



四、实验步骤

1. 明确项目需求；
2. 编写代码；
3. 编译代码；
4. 测试程序；
5. 根据测试结果对程序进行调试改进。