Grid-based Fluid Simulation

Yunfeng Bai

5/18/2011

1 Introduction

Fluid simulation is one of the hottest research area in computer graphics. Starting from the seminal paper [8], researchers have pushed both the accuracy and efficiency far forward. Currently, there are two major types of methods for fully 3D fluid simulation, the grid method [8] and the particle method [6]. Recently, Lentine et al. [5] proposed a method to accelerate the speed of projection step in grid method: the goal of this project is to build a incompressible smoke simulator based on MAC grid [3] and experiment the technique of projection on a coarse grid proposed in their paper. Also, in this project we also tried to add several extensions in fluid simulator literature (most of which are from [1]), to reduce the dissipation as well as increase the efficiency.

2 Overview

2.1 Navior-Stokes Equation

The essence of fluid simulation is to solve the Navier-Stokes equation:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{f} + \nu \nabla \cdot \nabla \vec{u} \tag{1}$$

where $\vec{u}$ is the velocity of the fluid, ρ is the density of the fluid, $\vec{f}$ is the force field which is applied on the whole fluid, typically gravity $\vec{g}$ of our main concern and p is the pressure, and ν is the kinematic viscosity. In this project, we only consider the incompressible, inviscid fluid, so the term $\nu \nabla \cdot \nabla \vec{u}$ is removed, and we get the inviscid version of Navier-Stokes equation:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{f} \tag{2}$$

For incompressible fluid, we also need to add the incompressible condition:

$$\nabla \cdot \vec{u} = 0 \tag{3}$$

Combining 2 and 3, we get the incompressible inviscid Navior-Stokes Equation.

2.2 Solving Incompressible Inviscid Navior-Stokes Equation

To solve the incompressible Navier-Stokes equations 2 and 3, we first split it into three parts as in [1]:

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \cdot \nabla \vec{u} \tag{4}$$

$$\frac{\partial \vec{u}}{\partial t} = \vec{f} \tag{5}$$

$$\frac{\partial \vec{u}}{\partial t} = -\frac{1}{\rho} \nabla p, \text{ s.t. } \nabla \cdot \vec{u} = 0 \tag{6}$$

At each time step, we solve the three equations one by one, with the output of the previous equation being the input of the next one. We'll use semi-Lagrangian method to solve the first equation and forward Euler for the second one. Let's name the intermediate result of velocity be $\vec{u}'$ when we just finished solving equation 4 and 5. Subsequently we'll solve equation 6, whose discrete version is:

$$\frac{\vec{u}^{n+1} - \vec{u}'}{\nabla t} = -\frac{1}{\rho}\nabla p \tag{7}$$

and we can solve 7 by considering the incompressibility condition 3 and get:

$$\nabla \cdot \frac{1}{\rho}\nabla(p\triangle t) = \nabla \cdot \vec{u}' \tag{8}$$

where we call the solving process of 7 and **"projection"**. After all, we'll use MICCG(0) [1] to solve the Poisson equations 8.

2.3 Program Structure Overview

There are mainly four parts in the simulator: MAC grid, MICCG solver, smoke simulator and rendering part.
The simulator uses the MAC grid and the MICCG solver when computing, and the rendering procedure uses the output of MAC grid to display.
The simulation cycle is:

1. Set time step.
2. Advect temperature, concentration and velocities.
3. Add forces.
4. Solve pressures by projection.
5. Update velocities with the solved pressures.
6. Render smoke and export smoke density. Go to step 1.

3 MAC Grid

The MAC grid [3] is a staggered grid, meaning that the three components of velocities are stored on the faces of each grid, while other parameters, like pressures, temperatures and concentrations are stored at the centers of each grid. So if we have $NX \times NY \times NZ$ grids, then we have $(NX+1) \times NY \times NZ$ velocity components in X direction, $NX \times (NY+1) \times NZ$ velocity components in Y direction and $NX \times NY \times (NZ+1)$ velocity components in Z direction. The $NY \times NZ + NX \times NZ + NX \times NY$ extra numbers can be proved to provide the discrete differentiation more accuracy.
Sometimes we want to get the values that are not at centers or on faces, then we need to interpolate on MAC grid.

3.1 Trilinear Interpolation

Assume we want to get the pressure value at point (x, y, z). Let

$$\begin{aligned} \bar{x} &= \lfloor x \rfloor \\ \bar{y} &= \lfloor y \rfloor \\ \bar{z} &= \lfloor z \rfloor \\ dx &= x - \bar{x} \\ dy &= y - \bar{y} \\ dz &= z - \bar{z} \end{aligned}$$

and

$$\begin{aligned} v_{000} &= p(\bar{x}, \bar{y}, \bar{z}) \\ v_{001} &= p(\bar{x}, \bar{y}, \bar{z} + 1) \\ v_{010} &= p(\bar{x}, \bar{y} + 1, \bar{z}) \\ v_{011} &= p(\bar{x}, \bar{y} + 1, \bar{z} + 1) \\ v_{100} &= p(\bar{x} + 1, \bar{y}, \bar{z}) \\ v_{101} &= p(\bar{x} + 1, \bar{y}, \bar{z} + 1) \\ v_{110} &= p(\bar{x} + 1, \bar{y} + 1, \bar{z}) \\ v_{111} &= p(\bar{x} + 1, \bar{y} + 1, \bar{z} + 1) \end{aligned}$$

Then

$$\begin{aligned} p_{trilinear}(x, y, z) = \; & (1 - dx) \times (1 - dy) \times (1 - dz) \times v_{000} \\ & (1 - dx) \times (1 - dy) \times z \times v_{001} + \\ & (1 - dx) \times y \times (1 - dz) \times v_{010} + \\ & (1 - dx) \times y \times z \times v_{011} + \\ & x \times (1 - dy) \times (1 - dz) \times v_{100} + \\ & x \times (1 - dy) \times z \times v_{101} + \\ & x \times y \times (1 - dz) \times v_{110} + \\ & x \times y \times z \times v_{111} \end{aligned}$$

3.2 Catmull-Rom Interpolation

Trilinear interpolation will produce great numerical dissipation during simulation. If we use trilinear interpolation to get concentration values of the smoke, we'll notice that the smoke will disappear rapidly near boundaries. To reduce the dissipation, we implemented Catmull-Rom interpolation, introduced by [2].

3.2.1 Catmull-Rom Interpolation in 1D

First let's consider the 1D case. To interpolate the value of point x on a 1D function f, let

$$\begin{aligned} x_0 &= \lfloor x \rfloor - 1 \\ x_1 &= \lfloor x \rfloor \\ x_2 &= \lfloor x \rfloor + 1 \\ x_3 &= \lfloor x \rfloor + 2 \\ dx &= x - x_1 \end{aligned}$$

then the Catmull-Rom interpolation gives the value

$$\begin{aligned} f_{Catmull-Rom-1D}(x) &= f(x_0)*(-\frac{1}{2}dx + dx^2 - \frac{1}{2}dx^3) + f(x_1)(1 - \frac{5}{2}dx^2 + \frac{3}{2}dx^3) + \\ & f(x_2)*(\frac{1}{2}dx + 2dx^2 - \frac{3}{2}dx^3) + f(x_3)(-\frac{1}{2}x^2 + \frac{1}{2}x^3) \end{aligned}$$

After all, let's define a function $F_{Catmull-Rom-1D}(dx, f(x_0), f(x_1), f(x_2), f(x_3))$, which outputs the value of the Catmull-Rom interpolation.

3.2.2 Catmull-Rom Interpolation in 2D

Now let's consider the 2D case. We'll interpolate the value of point (x, y) on a 2D function f in Y direction. Let

$$\begin{aligned} y_0 &= \lfloor y \rfloor - 1 \\ y_1 &= \lfloor y \rfloor \\ y_2 &= \lfloor y \rfloor + 1 \\ y_3 &= \lfloor y \rfloor + 2 \\ dy &= y - y_1 \end{aligned}$$

then we can get

$$\begin{aligned} f_{Catmull-Rom-2D}(x, y) = F_{Catmull-Rom-1D}(dy, \quad & f_{Catmull-Rom-1D}(x, y_0), f_{Catmull-Rom-1D}(x, y_1), \\ & f_{Catmull-Rom-1D}(x, y_2), f_{Catmull-Rom-1D}(x, y_3)) \end{aligned}$$

where $f_{Catmull-Rom-1D}(x, y_i)$ can be calculated by 1D Catmull-Rom interpolation in X direction.

3.2.3 Catmull-Rom Interpolation in 3D

So with the same idea, we interpolate the value of point (x, y, z) on a 3D function f in Z direction. Following the same definition for dz, z_0, z_1, z_2, z_3, define

$$\begin{aligned} f_{Catmull-Rom-3D}(x, y, z) = F_{Catmull-Rom-1D}(dz, \quad & f_{Catmull-Rom-2D}(x, y, z_0), f_{Catmull-Rom-2D}(x, y, z_1), \\ & f_{Catmull-Rom-2D}(x, y, z_2), f_{Catmull-Rom-2D}(x, y, z_3)) \end{aligned}$$

where $f_{Catmull-Rom-2D}(x, y, z_i)$ can be acquired by 2D Catmull-Rom interpolation in XY direction.
One last thing to mention is that 0 will be returned if we are trying to access the value outside of the grid.

4 MICCG

MICCG is a method used to solve symmetric positive definite linear systems, e.g., $Ap = b$. One important fact is that in Poisson equation, the coefficient matrix A is a sparse matrix, with at most 7 non-zero values each row.
First of all, let's review the preconditioned conjugate gradient algorithm (from [1]) in algorithm 1.
There are several external functions in this algorithm: applyPreconditioner(v), dotProduct(a, b), applyA(v), max(v) and abs(v). applyPreconditioner(v) multiplies the preconditioner by vector v; dotProduct calculates the dot product of two vectors a and b; applyA(v) multiplies the matrix A by vector v; max(v) returns the max value of v; abs(v) returns a vector whose values are the absolute values of v. Because A is a very sparse matrix, then applyA(v) is fast.
So our main concern is applyPreconditioner(v). With modified Cholesky factorization, A can be approximated by LL^T. If we let $M = (LL^T)^{-1}$ be the preconditioner, then to calculate $z = Mv$, we just need to solve the linear equations $LL^Tz = v$.

```
if b = 0 then
  return 0;
end
Initial guess p ← 0;
residual vector r ← b;
Auxiliary vector z ← applyPreconditioner(r);
Search vector s ← z ;
σ = dotProduct(z, r);
for iteration ← 0 to Max Iteration do
    z ← applyA(s) ;
    α ← σ/dotProduct(z, s) ;
    p ← p + αs ;
    r ← r − αz ;
    if max(abs(r)) ≤ Tolerance then
      return p;
    end
    z ← applyPreconditioner(r) ;
    σ_new ← dotProduct(z, r) ;
    β ← σ_new/σ ;
    s ← z + βs ;
    σ ← σ_new ;
end
return p;
```

Algorithm 1: Preconditioned Conjugate Gradient Algorithm

In modified Cholesky factorization, $L = FE^{-1} + E$, where F is the strict lower triangle of A, and E is a diagonal matrix:

$$E(i,j,k) = \sqrt{a - b - c - d - e - f - g}$$

where

$$\begin{aligned}
a &= A_{(i,j,k),(i,j,k)} \\
b &= (A_{(i-1,j,k),(i,j,k)}/E_{(i-1,j,k)})^2 \\
c &= (A_{(i,j-1,k),(i,j,k)}/E_{(i,j-1,k)})^2 \\
d &= (A_{(i,j,k-1),(i,j,k)}/E_{(i,j,k-1)})^2 \\
e &= A_{(i-1,j,k),(i,j,k)}(A_{(i-1,j,k),(i-1,j+1,k)} + A_{(i-1,j,k),(i-1,j,k+1)})/E^2_{(i-1,j,k)} \\
f &= A_{(i,j-1,k),(i,j,k)}(A_{(i,j-1,k),(i+1,j-1,k)} + A_{(i,j-1,k),(i,j-1,k+1)})/E^2_{(i,j-1,k)} \\
g &= A_{(i,j,k-1),(i,j,k)}(A_{(i,j,k-1),(i+1,j,k-1)} + A_{(i,j,k-1),(i,j+1,k-1)})/E^2_{(i,j,k-1)}
\end{aligned}$$

it satisfies that the off-diagonal non-zero entries of A are equal to the corresponding ones of LL^T, and the sum of each row of A is equal to the sum of each row of LL^T.
To solve $LL^Tz = v$, we first solve $Lq = v$, then we solve $L^Tz = q$ to get z. Because L is a sparse matrix, solving q and z will be fast.

5 Smoke Simulator

With all the tools we have, we can start to build a smoke simulator. In this section, we'll discuss simulation steps in the simulation cycle (step 1 - 5).

5.1 Set Timestep

To make sure our solution converges to the correct answer (satisfying CFL condition), we let $\Delta t \leq \frac{\Delta x}{u_{max}}$, where u_{max} is the maximum velocity stored on the grid. However, for the simplicity of film capturing, we sometimes set Δt to $\frac{1}{24}$.

5.2 Advection

Advection step is used to solve equation 4. It basically transfers whatever we have in grid according to the velocity field.

5.2.1 Semi-Lagrangian Advection

Semi-Lagrangian advection, introduced in [8], is the classic method for advection in fluid simulation. The idea behind semi-Lagrangian method is that, if we want to get the new value at point p, $q^{n+1}(p)$, we can just trace back from p to p' according to the velocity field and use $q^n(p')$. Formally, if we use forward Euler,

$$q^{n+1}(p) = q^n(p - \Delta t \times \vec{u}(p))$$

5.2.2 MacCormack Method

MacCormack method is an extension to semi-Lagrangian advection, introduced in [7], it has the same order of accuracy as BFECC [4], but with lower computational cost. The central idea of MacCormack method is going back and forth, and use the difference between the original point to compensate the result. Formally, if we know value ϕ^n, an advection operator A (e.g. forward Euler) and its reverse operator A^R, then

$$\begin{aligned} \hat{\phi}^{n+1} &= A(\phi^n) \\ \hat{\phi}^n &= A^R(\hat{\phi}^{n+1}) \\ \phi^{n+1} &= \hat{\phi}^{n+1} + (\phi^n - \hat{\phi}^n)/2 \end{aligned}$$

5.3 Extensions for Smoke

To simulate the behavior of smoke, we add several extensions to the simulator.

1. Temperature and concentration
2. Buoyancy
3. Vorticity force

Temperature and concentration are stored at the centers of grid, and will be advected during simulation. Buoyancy is calculated by the following equation:

$$\vec{b} = [\alpha s - \beta(T - T_{amb})]\vec{g}$$

where α is the relative density difference of the smoke to air, and T_{amb} is the ambient temperature of the environment (usually set to 273K).

5.3.1 Vorticity Force

Vorticity ω is defined to be the curl of the velocity field, and it is also twice the local angular velocity, formally $\omega = \nabla \times \vec{u}$. To enhance vorticity (adding more turbulence), we can construct a normalized vector $\vec{N}$ that points to the direction where the length of vorticity changes the most:

$$\vec{N} = \frac{\nabla \|\vec{\omega}\|}{\|\nabla \|\vec{\omega}\|\|}.$$

Then we can define the vorticity confinement force to be:

$$f_{conf} = \epsilon \Delta x (\vec{N} \times \vec{\omega}).$$

To get the value of vorticity and the gradient of the length of the vorticity, we can use discrete differentiation:

$$\begin{aligned} \vec{w}_{i,j,k} &= (\frac{w_{i,j+1,k} - w_{i,j-1,k}}{2\Delta x} - \frac{v_{i,j,k+1} - v_{i,j,k-1}}{2\Delta x}, \\ &\quad \frac{u_{i,j,k+1} - u_{i,j,k-1}}{2\Delta x} - \frac{w_{i+1,j,k} - w_{i-1,j,k}}{2\Delta x}, \\ &\quad \frac{v_{i+1,j,k} - v_{i-1,j,k}}{2\Delta x} - \frac{u_{i,j+1,k} - u_{i,j-1,k}}{2\Delta x}), \end{aligned}$$

and

$$\nabla \|\vec{w}\|_{i,j,k} = (\frac{\|w\|_{i+1,j,k} - w_{i-1,j,k}}{2\Delta x}, \frac{\|w\|_{i,j+1,k} - w_{i,j-1,k}}{2\Delta x}, \frac{\|w\|_{i,j,k+1} - w_{i,j,k-1}}{2\Delta x})$$

6 Rendering

In each rendering cycle, we'll execute the following functions:

1. Render solids.
2. Render smoke.
3. Take screen shot.
4. Export smoke concentration to files.

To render smoke, we set OpenGL to BLEND mode, and use the concentration of each grid as the alpha value of the color. Then we render each grid as a cube, in which case we draw 6 faces with the color defined previously. One obvious deficiency here is that 6 faces are not enough to represent the light absorbance behavior of a solid cube, however, it's suitable for a fast hardware rendering.
I planed to render images with the ray tracer Mitsuba. Its newest stable release version is 0.2.0, however there is no tutorial nor sample about how to render a scene with participating media, so I downloaded their code and tried to figure out the file format. Eventually I was able to figure out the file format of the density file, but no luck on the scene file. So I turned to find the help from Wenzel Jakob and Shuang Zhao. They suggested me to use the newest version of Mitsuba, and provided me a sample scene file. It also took me some time to figure out the format of the density file for the newest version of Mitsuba. Because there is no released build, I had to build myself. However my build had problems rendering even a simple scene, which is perfectly rendered with the previous build. By far I'm stuck here and not able to finish ray tracing in time.

7 Projection on A Uniform Coarse Grid

In [5], they project the fine grid velocities to the coarser grid using an area-weighted average of all the fine grid velocities, which is given by $\vec{u}'_c = \frac{1}{n} \sum_{f \in faces} A_f \vec{u}'_f$, where $\vec{u}'_c$ is the projected velocity on the coarse face, $faces$ is the set of fine scale faces that overlap with the coarse face, A_f is the area of the fine face, and n is the number of elements in $faces$.
We then could use the usual projection method to solve equations 8. To map velocities back to the fine grid, [5] used $\vec{u}_f^{n+1} = \alpha \vec{u}_c^{n+1} + (1 - \alpha)(\vec{u}'_f + \vec{u}_c^{n+1} - \vec{u}'_c)$, where α is a constant between 0 and 1 and $\vec{u}_f$ is the velocity on the fine grid. Note that here they only map to the fine grid faces that are incident on the coarse grid faces.

After mapping velocities from coarse grid to fine grid, we need to fill those still unknown velocities on fine grid faces and make the velocity field in fine grid divergence-free. [5] achieved this by solving 8 within each coarse grid's domain, with known $\vec{u}_f$'s as Neumann boundary conditions and unknown ones left to be solved. If we are projecting a uniform fine grid to a uniform coarse grid, then we can use a hard-coded routine. Consider the simplest case, where a 2 by 2 by 2 fine grid whose size is 1 is projected to a 1 by 1 by 1 coarse grid, and the centers of each fine grid cell is located on $(0,0,0)$, $(0,0,1)$, $(0,1,0)$, $(0,1,1)$, $(1,0,0)$, $(1,0,1)$, $(1,1,0)$, $(1,1,1)$. Because all of the boundary conditions are Neumann conditions, there will be rank 1 degeneracy. If we define pressure to be $p(x,y,z)$ and fix $p(0,0,0)=0$, then we have the following Poisson equation:

$$\begin{bmatrix} 3 & 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & 3 & -1 & 0 & 0 & -1 & 0 \\ -1 & -1 & 3 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 3 & -1 & -1 & 0 \\ -1 & 0 & 0 & -1 & 3 & 0 & -1 \\ 0 & -1 & 0 & -1 & 0 & 3 & -1 \\ 0 & 0 & -1 & 0 & -1 & -1 & 3 \end{bmatrix} \begin{bmatrix} p(0,0,1) \\ p(0,1,0) \\ p(0,1,1) \\ p(1,0,0) \\ p(1,0,1) \\ p(1,1,0) \\ p(1,1,1) \end{bmatrix} = -\frac{\rho}{\Delta t} \begin{bmatrix} u_{0.5,0,1} - u^*_{-0.5,0,1} + v_{0,0.5,1} - v^*_{0,-0.5,1} + w^*_{0,0,1.5} - w_{0,0,0.5} \\ u_{0.5,1,0} - u^*_{-0.5,1,0} + v^*_{0,1.5,0} - v_{0,0.5,0} + w_{0,1,0.5} - w^*_{0,1,-0.5} \\ u_{0.5,1,1} - u^*_{-0.5,1,1} + v^*_{0,1.5,1} - v_{0,0.5,1} + w^*_{0,1,1.5} - w_{0,1,0.5} \\ u^*_{1.5,0,0} - u_{0.5,0,0} + v_{1,0.5,0} - v^*_{1,-0.5,0} + w_{1,0,0.5} - w^*_{1,0,-0.5} \\ u^*_{1.5,0,1} - u_{0.5,0,1} + v_{1,0.5,1} - v^*_{1,-0.5,1} + w^*_{1,0,1.5} - w_{1,0,0.5} \\ u^*_{1.5,1,0} - u_{0.5,1,0} + v^*_{1,1.5,0} - v_{1,0.5,0} + w_{1,1,0.5} - w^*_{1,1,-0.5} \\ u^*_{1.5,1,1} - u_{0.5,1,1} + v^*_{1,1.5,1} - v_{1,0.5,1} + w^*_{1,1,1.5} - w_{1,1,0.5} \end{bmatrix}$$

where $u*$, $v*$ and $w*$ are the velocities that being projected back from the coarse grid. We can solve this equation and hardcode the solution. However, due to time restriction, I haven't finished this part yet.

8 Conclusion

In this project, we implemented a fluid simulator that is specially optimized for smoke simulation by adding buoyancy and vorticity confinement forces. Also we reduced numerical dissipation by Catmull-Rom interpolation and MacCormack method. After all, the efficiency is guaranteed by solving Poisson equation with MICCG. Unfortunately, some planned goals are not accomplished, such as ray tracing rendering and coarse grid projection.

From this project I learned that writing a numerical simulation program is very error-prone, and it's almost impossible to write a correct program before having comprehensive understanding of the subject. However it's fun to see numbers become some real thing that can be displayed on the screen, also from which I had a great sense of accomplishment.

References

[1] R. Bridson. *Fluid simulation for computer graphics*. AK Peters Ltd, 2008.

[2] R. Fedkiw, J. Stam, and H.W. Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22, 2001.

[3] F.H. Harlow, J.E. Welch, et al. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of fluids*, 8(12):2182, 1965.

[4] B.M. Kim, Y. Liu, I. Llamas, and J. Rossignac. Advections with significantly reduced dissipation and diffusion. *IEEE Transactions on Visualization and Computer Graphics*, pages 135–144, 2007.

[5] M. Lentine, W. Zheng, and R. Fedkiw. A novel algorithm for incompressible flow using only a coarse grid projection. In *ACM SIGGRAPH 2010 papers*, pages 1–9. ACM, 2010.

[6] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159. Eurographics Association, 2003.

[7] A. Selle, R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac. An unconditionally stable maccormack method. *Journal of Scientific Computing*, 35(2):350–371, 2008.

[8] J. Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.