Smoke Simulation Report **Linghan Zheng**
CIS 563
Prof. Jiang Date: 05/07/18

Introduction

This project implemented an incompressible smoke simulator using Marker-and-Cell (MAC) method. The main references are mthe smoke simulation paper, *Visual Simulation of Smoke* [1], *SIGGRAPH 2007 Fluid Simulation Course Notes* [2], and my own notes. All the codes are written in C++ on Ubuntu 16.04 virtual machine, and all the results are rendered in Houdini.

Basically, there are three main steps during the simulation: advection, force computation, and projection. Advection is the process of solving $\frac{\partial q}{\partial t} + \underline{v} \cdot \bigtriangledown q = 0$, where q can be velocity, temperature or density in this program. Force Computation is the step to add external forces including buoyancy and vorticity confinement. Finally, Projection will make sure the smoke is incompressible (i.e. $\bigtriangledown \cdot \underline{v} = 0$). I will explain the implementation in detail later.

MAC Grid

The main data structure used in this project is MAC Grid, which is used to model the smoke as a velocity field. For a $X \times Y \times Z$ grid, there are $(X+1) \times Y \times Z$ X-faces, $X \times (Y+1) \times Z$ Y-faces and $X \times Y \times (Z+1)$ Z-faces. We can store u-, v-, w-component of velocity on X-, Y-, Z-faces respectively (class GridDataX/Y/Z in code), but store pressure, temperature and density at cell centers (class GridData).

Advection

To advect velocity, we need to loop all X-, Y-, Z-faces. If the face is a boundary between smoke and static solid, we can set the stored velocity component as 0 directly. Otherwise, we need to get the full velocity at center of the face and then use either Forward Euler or RK2 method to trace back old position. Finally, we can store the u-, v-, w- component at clipped traced position in this face. The pseudo-code is shown as Algorithm 1 (take X-face as example):

The advection of temperature and density are quite similar, however, instead of looping each face, we traverse each smoke cell now. In a short, the process can be represented as $T_{new} \leftarrow interpolateTattraceBack$(curPosition, curVelocity, dt).

Force Computation

In this project, we mainly consider two forces: buoyancy and vorticity confinement. If want to make the results more interesting, we may also add some constant "wind force" to adjust velocity.

Buoyancy force can be easily computed using the following equation, which only affect velocity's v-component.

$$f_{buoy} = (0, -\alpha \cdot cellDensity + \beta(T - T_{amb}), 0) \tag{1}$$

```
input  : $dt$
Result: update velocity on each face
for each X-face do
    if boundary face then
        mU(i, j, k) ← set 0 directly;
    else
        curVelocity ← {mU(i, j, k), avg(mV of 4-nb Y-faces), avg(mV of 4-nb Z-faces};
        if RK2 then
            midPosition ← curPosition - 0.5 $*$ $dt*$ curVelocity;
            midVelocity ← interpolate velocity at midPosition;
            oldPosition ← curPosition - $dt*$ midVelocity;
        else
            oldPosition ← curPosition - $dt*$ curVelocity;
        end
        mU(i, j, k) ← u-component of interpolated velocity at oldPosition;
    end
end
```

Algorithm 1: Advect Velocity

Vorticity confinement is used to boost the strength of vorticies and make the smoke more lively and interesting. All the useful equations are shown below, where h is the cell size, $\underline{\omega} = \bigtriangledown \times \underline{v}$ and $\underline{N} = \bigtriangledown|\underline{\omega}|/\| \bigtriangledown |\underline{\omega}|\|$.

$$f_{conf} = \epsilon h(\underline{N} \times \underline{\omega}) \tag{2}$$

We need to compute the force for each cell first, and then update the velocity at each non-boundary face as (still take X-face as example):

$$mU(i,j,k) += 0.5 * dt * (f_{conf}.X(i-1,j,k) + f_{conf}.X(i,j,k)) \tag{3}$$

Projection

To make smoke incompressible, one step called *project* is needed, where we subtract off the pressure gradient from the intermediate velocity we computed above (note as $\underline{v}^*$). The goal of projection is to ensure $\bigtriangledown \cdot \underline{v} = 0$ for each cell. So for cell(i,j,k), we want:

$$\frac{mU(i+1,j,k) - mU(i,j,k)}{h} + \frac{mV(i,j+1,k) - mV(i,j,k)}{h} + \frac{mW(i,j,k+1) - mW(i,j,k)}{h} = 0 \tag{4}$$

Then because $\underline{v} = \underline{v}^* - \frac{dt}{\rho} \bigtriangledown p$, we can construct a equation for each cell, and then by combining all these equations, we can get a linear system with the format as $\underline{A}\underline{p} = \underline{d}$. Here $\underline{A}$ is a sparse $n \times n$ matrix, each row of which corresponds to one smoke cell. For example, for row l, $\underline{A}(l,l)$ is the number of cell-l's non-solid neighbors, and if any cell-m is its neighbor, $\underline{A}(l,k) = -1$. Obviously, this is a symmetric matrix, however, it is singular.

In this project, we use Preconditioned Conjugate Gradient (PCG) algorithm with Modified Incomplete Cholesky (MIC) to solve the system. The detail can be found in [2]. For a 64×64×64 MAC grid, it converges after about only 30 iterations.

Extra Feature

Besides simple rising smoke and two sources injecting towards one another, I have also implemented obstacles in the way to make the results more interesting. The main challenges to add the extra feature include enforcing more complex boundary conditions and adjusting the linear system in projection stage. For the boundary, we can just treat them as the original wall, and keep the velocity to zero all the time. However, for the linear system, things are getting a little bit tricky because we need to construct different $\underline{A}$ based on new environment now, and the way of mapping (i, j, k) to $\underline{A}$'s row is more complex. I have used two different methods to do this:

1) Construct smaller $\underline{A}$ for only non-solid cells. This method is more natural because we can still follow the rule mentioned above to calculate $\underline{A}$'s entries. However, it is not consistent with the provided base code, where the dimension of $\underline{A}$ is fixed and the storage of $\underline{A}$ is not in a traditional way. What's more, extra effort is needed for the mapping between cell index and matrix index. Hence, in my implementation, I first used $std :: map$ to hash (i, j, k) to $index$. Then, rather than use given *GridDataMatrix*, I created a *Eigen::SparseMatrix* to store $\underline{A}$. And we also need to convert *GridData* to $Eigen :: VectorXd$ during computation.

2) Still use $n \times n$ matrix, but the only non-zero entry for a row corresponding to solid cell is 1 on the diagonal, and the corresponding right-hand side is 0. This method is more convenient to implement with our base code. So I changed the $calculateAMatrix()$ function to construct the new matrix. Nevertheless, with a bigger matrix, it takes more time to converge than first method.

In addition, I also tried to add a **moving obstacle** in the way, which makes the environment even more complex. However, considering the larger computation and the limited ability of my virtual machine, I only displayed an example of 16 $\times$ 16 $\times$ 16 grid to show the feature.

Links

Video Link: https://www.youtube.com/watch?v=RUM1To7Pcd8&feature=youtu.be

Github Link: https://github.com/linghan0919/SmokeProj

Acknowledgement

I would like to show my gratitude to Prof Jiang for giving interesting and great lectures. And I am also grateful to two TAs for their help in this semester.

References

[1] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. "Visual simulation of smoke." Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM, 2001. Available from: (http://physbam.stanford.edu/ fedkiw/papers/stanford2001-01.pdf).

[2] Robert Bridson, and Matthias Mller-Fischer. "Fluid simulation: SIGGRAPH 2007 course notes Video files associated with this course are available from the citation page." ACM SIGGRAPH 2007 courses. ACM, 2007.