# Eulerian Grid Smoke Simulation

CIS 563 final project write up
Lan Lou

May 6, 2018

## 1 Introduction

This project is based on the paper *Visual simulation of smoke* [1] and the *2007 SIGGRAPH fluid course notes* [2] and also some of Aline Normoyle 's fluid simulation tutorial, the fundamental theory across the project is a staggered MAC Grid (Marker-and-Cell Method) view,in this view, instead of treating the simulated targets as Lagrangian particles, we evaluate all the physics based on a constant Eulerian grid which is mac grid.

The code of this project is using the base code provided by the TAs, and is written in C++, the system I'm using is ubuntu virtual machine running with 6 Gigabytes memory and 6 Core virtual processor of intel i7-6700HQ.

I render the smoke in the form of particles in houdini, including a few scenarios: two smoke colliding, one smoke propagating, and one smoke meet a moving sphere.

The entire simulation process can be divided into a few steps:

- generate resource particles with initial velocities, temperature, density.
- advection of velocity, using semi-Lagrangian advection and RK2.
- compute two external forces: vorticity confinement force, the buoyancy force and use force update velocities.
- apply projection to keep the divergence free and use the pressure result update the velocities again.
- advection of temperature , density also using semi-Lagrangian advection and RK2.

Now I will explain each step in detail

## 2 Simulation process

### 2.1 The MAC-Grid

First I have to explain the mac grid data structure

The data that is going to be stored are : velocities on three faces in each cell, the temperature, density, and pressure stored in the center of each cell. As shown in Figure 1, this is a 2X2X1 grid example, there are three faces in x and y direction but only two faces in z direction, similarly,

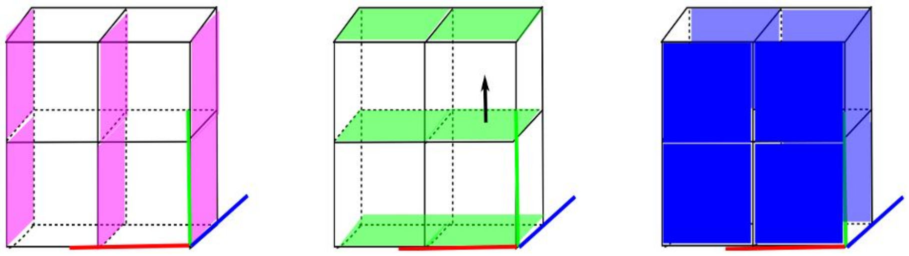

Figure 1: The face order in three directions.

when we are storing $X \times Y \times Z$ cells, the velocities in X directions can be traversed using the $(X+1) \times Y \times Z$ , so different from the FOR_EACH_FACE provided by the original code, I used FOR_EACH_FACE_X, FOR_EACH_FACE_Y, and FOR_EACH_FACE_Z to do iterations for data in faces of each directions respectively, as for the pressure, density, temperature, they are simply stored in the cell center, can be traversed using FOR_EACH_CELL.

## 2.2 Advection

Since the advection of velocity, density and temperature are similar, I will just put the explanation together:

In short, the pseudo code of advection can be written as this:

FOR_EACH_FACE

- currentpt = position at center of current face ;
- currentvel = velocity at center of current face ;
- midpt = currentpt - dt * currentvel / 2.0 ;
- midvel = interpolate velocity at mipt ;
- oldpt = currentpt - dt * midvel ;
- newvel = interpolate velocity at old location ;
- store one component of newvel depending on face type ;

Similarly , if you want to advect density temp etc which are stored in cell, center, you only need to change FOR_EACH_FACE into FOR_EACH_CELL and change the interpolation to the corresponding type, also ,one thing to mention is that the interpolation of this project is using a cubic interpolation using the surrounding 4 cells or faces.

## 2.3 The buoyancy force

Like many other physics simulation, smoke simulation also have to take the gravity and temperature etc into consideration, this leads us to the derivation of the buoyancy force for smoke, one governing equation for this behavior is

$$F_buoyan = -\alpha * Density_face + \beta * (T - T_ambient)$$

where $\alpha$ and $\beta$ are user defined parameters, Density face indicates the average density on the current face that you are computing the velocity on, T ambient is set to 0 by default.

## 2.4 The vorticity confinement force

One feature that distinguish smoke's behavior from simple particle material is the vorticity confinement, also , I will write a short description for this part :

FOR_EACH_CELL

- compute central velocity of the cell, and store it in the cell center, the central vel is calculated using the average value of the adjacent face velocities.
- compute the Omega $\omega$ :
$$\omega = \nabla \times u$$
- then find the direction of the vortices
$$N = \frac{\nabla|\omega|}{||\nabla|\omega|||}$$
- finally compute the force :
$$f_conf = \epsilon \Delta_x (N \times \omega)$$
- the last step : update velocity on the face using
$$\Delta_v = \frac{dt * (f_frontcell - f_backcell) * TheCellsize}{Density_fluid}$$

## 2.5 Projection

The main purpose of the projection step is to make sure the fluid maintain divergence free :

$$\nabla.u = 0$$

and in the meantime follow the possion equation :

$$u^{n+1} = u^* - \delta_t \frac{1}{\rho} \nabla_p$$

then we get :

$$(6p_{i,j,k} - pi+1,j,k - pi-1,j,k - pi,j+1,k - pi,j,k+1 - pi,j,k-1) = \frac{-\nabla_x^2}{\nabla_{t\rho}}(\nabla.u)$$

finally we will be able to represent these in the form of a matrix multiplication : Ap = b in the left hand side, A is a matrix following these orders :

- each element in A indicate a cell
- Pi,j,k indicate the current cell, and the others are the one surrounding it
- the coefficient for pi,j,k is the number of the neighbors, and for the others the coefficients is -1
- non neighbors have coefficients equal 0

now that we have constructed A we have to construct vector b , basically vector b is simply the divergence of the eulerian grid, the sum of it have to be strictly zero since we are obeying divergence free.

To be more specific, I will display a pseudo code of this :

- gridData divergence
- FOR_EACH_CELL
- divergence(i,j,k) $= -(mU(i+1,j,k) - mU(i,j,k) + mV(i,j+1,k) - mV(i,j,k) + mW(i,j,k+1) - mW(i,j,k))/THEcellsize$

Now , I have A and b, I can finally compute p, I used the PCG provided by the base code to iteratively solve the linear equations, I add the calculate preconditioner code based on the fluid course[2] page 36 ,figure 4.2.

Then, we use the gradient of P to update velocities.

## 3 Some extra featrues

In the demo cases, I tried a moving sphere object as the collision target for the smoke, I managed to do this by applying boundary conditions in two sections

- the advection of particles (since I'm mainly focusing on the particle rendering), for each particles, I will evaluate the distance of it to the sphere center, if it is smaller than the radius I set before, then it will be pushed along the direction of the current position to the sphere center with a specific distance that place it exactly on the sphere's surface. if the distance is larger than the radius, then no operation would be applied.
- the second section is applied to the getvelocity function, wich means just before doing the interpolation of the velocity to get the velocity of an arbitrary position, again, if the distance larger than radius, nothing happens, however, because in the former step, we have already pushed the particles onto the sphere surface, so there will not be particles inside the sphere, so we only need to consider the particle on the sphere surface, next step is quite simple, I simply let the particle's velocity's (particle to sphere center) projection component be 0.

## 4 Links

- github : https://github.com/LanLou123/SmokeSim
- video : https://www.youtube.com/watch?v=ctjgf8bTLxg

## References

[1] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 15–22, New York, NY, USA, 2001. ACM.

[2] Matthias Muller-Fischer Robert Bridson. Fluid simulation siggraph 2007 course notes. 14(3):1–93, 2007.