서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

Term project Report

soheon yoo (2021-22823)
yumk0717@snu.ac.kr
Seoul National University - Advanced Graphics (Spring 2023)

Abstract

Visual simulation of smokes was implemented based on Semi-Lagrangian method fluid system. In the force term calculation for smoke fluid, temperature term was considered. And vorticity confinement force was also implemented. For rendering 3D texture was used for grid density. DirectX 11 and Cpp17 was used for implementation. For fast calculation of matrix and vector Eigen library and DirectXMath was used.

1 Introduction

Fluid simulation and rendering is important part in computer graphics. There are eulerian method and lagrangian method for simulating fluids. Both method is for calculating navier-stokes equation. Eulerian method is based on grid and space and lagrangian method is based on particles. In this term project, the fluid system is based on semi-lagrangian method introduced by stam[1]. Actually this is method eulerian based, in other words grid based. Vorticity confinement was also introduced for smoke simulation[2]. And Rendering method for smoke system is based on fedkiw's paper[2].

2 Methods

Smoke simulation was implemented using cpp. For rendering DirectX 11 and HLSL was used. And Big-size matrix calcultion was done using Eigen. All vector math was done in XMMATRIX and XMVECTOR included in DirectXMath for SIMD

3 Model explanation

The basic system is based on Stam's stable fluid system[1]. Basic equation for fluid is Navier-Stokes equation.

$$\nabla \cdot u = 0$$

$$\frac{du}{dt} = -(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + v\nabla^2 u + f$$

There is force term, diffusion term, advection term and pressure term(projection) in Navier-Stokes equation. All the calculation for u is not done by once, but step by step. For smoke calculation viscosity term(diffusion term) was disregarded, because air have very low viscosity unlike liquid. Model is comprised of 3 steps. Advect step, Force step and pressure step.

3.1 Advect step

In advect step, semi-lagrangian method was used. For advecting velocity grid and density grid, calculate position certain time ago. And change value's to old value at old page. Let's say w(x) is grid value at position x.

$$w_2(x) = w_1(p(x, -\Delta t))$$

3.2 Force step

This step is very simple, firstly set force and change the velocity grid.

$$w_2(x) = w_1(x) + \Delta t f(x, t)$$

In the Smoke simulation Temperature term was included. If temperature is high smoke tend to go upward and low T smokes tend to sink. So force for smokes was modeled as below.

$$f_{buoy} = \alpha\rho z + \beta(T - T_{amb})z$$

alpha is gravity constant and beta can be changed for modeling. T_{amb} is ambient Temperature. And vorticity confinement forces was introduced. It will be explained section 5

3.3 Pressure step

pressure term calculation is done by solving poisson equation.

$$\nabla^2 P = \nabla \cdot w_1(x)/\Delta t$$

$$\frac{\partial P}{\partial N} = 0$$

$$w_2(x) = w_1(x) - \Delta t \nabla P$$

firstly calculate pressure grid using known velocity grid and update velocity grid using pressure gradient.

4 Algorithm

for *all Grid* i **do**
  Initialize all variable for i
end
while *simulating* **do**
  for *all Grid* i **do**
    Advect velocity u_i
    Advect density ρ_i
    Advect Temperature $Temp_i$
  end
  for *all Grid* i **do**
    Reset Force
    Calculate Vorticity confinement force for i
    update velocity
    $u_i = \Delta t f_i$
  end
  for *all Grid* i **do**
    Calculate pressure p_i update velocity
    $u_i = u_i - \Delta t \nabla P$
  end
  Render with grid density
end

Algorithm 1: Smoke simulation Algorithm

5 Rendering

Rendering of smoke is done using grid density value. If light collides with smoke, photon is either scattered or absorbed. So light is attenuated if ray pass through positive density field. Let's say T_{ray} is chagned(attenuated) by is transparency of ray and T_{vox} is transparency of voxel. and L means radience. If ray meets voxel T_{ray} is chagned(attenuated) by T_{vox}. And radience of each voxel can be calculated as below.

$$L_{vox} = L_{ray}(1 - T_{vox})T_{ray}$$

For inputting density in shader, I used 3D dynamic texture. In the eyeDirection to pixel there will be many voxel. And

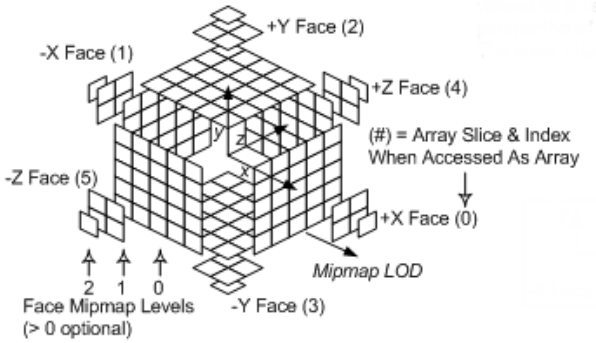

Figure 1: **3D texture in DX**

each voxel have its own radiance and transparency. It also will be attenuated in the pass from voxel to eye. So it was implemented in double loop in HLSL code.

6 Vorticity confinement

This metho is not about physicall correctness. It is for the liveliness of smokes in visually. Getting vorticity(omega) in grid was done by below code. befor using this code average velocity should be calculated.

Listing 1: Code for getting vorticity

```
1  omg_x[IDXto1D(i, j, k)] = (avg_w[IDXto1D(i, j + 1, k)] − ↩
   avg_w[IDXto1D(i, j − 1, k)] −avg_v[IDXto1D(i, j, k + 1)] ↩
   + avg_v[IDXto1D(i, j, k − 1)]) / 2VOXELSIZE;
2
3   omg_y[IDXto1D(i, j, k)] = (avg_u[IDXto1D(i, j, k + 1)] −↩
   avg_u[IDXto1D(i, j, k − 1)] − avg_w[IDXto1D(i + 1, j, k)] ↩
   + avg_w[IDXto1D(i − 1, j, k)]) /2VOXELSIZE;
4
5   omg_z[IDXto1D(i, j, k)] = (avg_v[IDXto1D(i + 1, j, k)] − ↩
   >avg_v[IDXto1D(i − 1, j, k)] − avg_u[IDXto1D(i, j + 1, k)↩
   ] + avg_u[IDXto1D(i, j − 1, k)]) /2VOXELSIZE;
```

Then normalized vorticity location vectors are computed.

$$N = \frac{\mu}{|\mu|}(\mu = \nabla|\omega|)$$

The force for the vorticity can be calculated by below equation.

$$f_{conf} = \epsilon h(N \times w)$$

h is voxel size and epilson can be changed for visualization. We can add or decrease vorticity as we want.

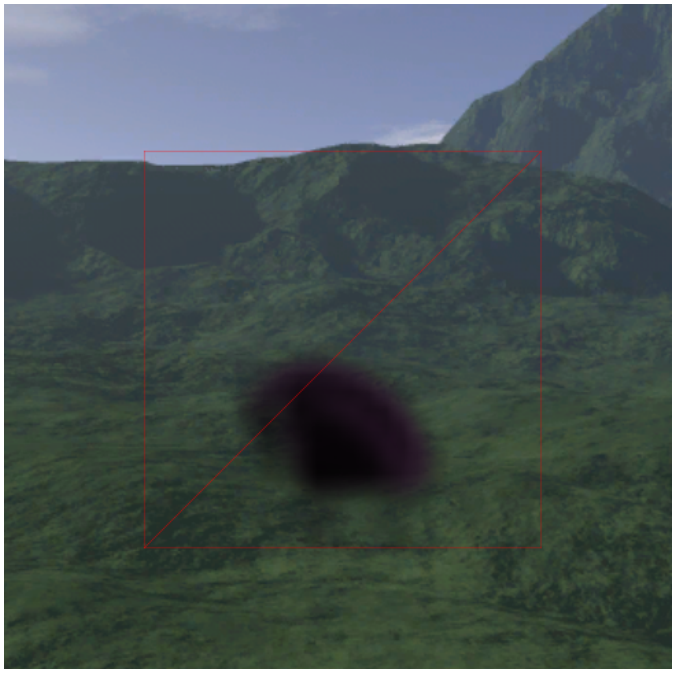

Figure 2: **3D smoke in DX11**

7 Discussion

At first i planned to make real-time smoke system based on Navier-Stokes equation. But Avecting and pressure step take too much time. So fps was almost one ,even if it is 20x20x20 very small grid system. For fast calcultion all the vector calculation was changed to xna API for SIMD but real-time

physics system was really hard.
When Interpolating in semi- lagrangian step(advection), Monotonic cubic interpolation [2] was used. This is revised version of Hermite interpoation. Becasue cubic Hermite interpolation makes overshoot, some revision was done. when we calculate f(t) , (t is between tk and tk+1)
$f(t) = a_3(t-t_k)^3 + a_2(t-t_k)^2 + a_1(t-t_k) + a_0$
where

$$a_3 = d_k + d_{k+1} - \Delta_k$$
$$\mathsf{a}_2 = 3\Delta_k - 2d_k - d_{k+1}$$
$$\mathsf{a}_1 = d_k$$
$$\mathsf{a}_0 = f_k$$

and

$$d_k = (f_{k+1} - f_{k-1})/2, \Delta_k = f_{k+1} - f_k$$

for handling overshoot, change the sign of d_k, d_{k+1} to sign of Δ_k, Then interpolation become monotonic without overshooting.

References

[1] J. Stam, "Stable fluids," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 121–128, 1999.

[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*, pp. 15–22, 2001.