FLUID SIMULATIONS

Adarsh Krishnamurthy (cs184-bb)
Bela Stepanova (cs184-bs)

OBJECTIVE

The basic objective of the project is the implementation of the paper "Stable Fluids" (Jos Stam, SIGGRAPH 99). The final project will focus on simulation of gases and fluid like flows in 2D and 3D with rigid or open boundary conditions.

SECONDARY OBJECTIVES

Flow past objects in 2D
Multiple fluids
Make the simulations real-time

INTRODUCTION

Simulation of fluid like motions is of great use in many applications. It allows one to animate the appearance and behavior of fluids such as smoke, water, clouds, wind and fire. It is also a good method to be introduced to the basics of physics based animation.

THEORY

We first started with a 2D explicit time integration solver for non-free surface problems. These simulations are based on the Navier-Stokes equations for fluid flow that specify both conservation of momentum and mass. They are given by the two equations

$$\nabla \cdot u = 0$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \upsilon \nabla^2 u + f$$

We implemented the simulator using an implicit solver as this approach has a number of advantages. First, it won't be subject to blow ups if the velocities become large. Second, it is really simple and easy to debug, so the conversion to 3D will be much less painful.

The Navier Stokes solver is based on the principle of operator splitting. It splits the Navier Stokes equation into four basic steps; add force, advect, diffuse and project for each time step. The sequence of steps is performed for the three components of the velocity. Also this velocity field is then used to advect and diffuse any density introduced into the field.

$$w_0(x) \xrightarrow{\text{add force}} w_1(x) \xrightarrow{\text{advect}} w_2(x) \xrightarrow{\text{diffuse}} w_3(x) \xrightarrow{\text{project}} w_4(x)$$

The first step in the solution is the addition of external force. If we assume that the force does not vary considerably during the time step, then

$$w_1(x) = w_0(x) + \Delta t\, f(x,t)$$

is a good approximation of the effect of the force on the field over the time step.

The next step accounts for the effect of advection (or convection) of the fluid on itself. A disturbance somewhere in the fluid propagates according to the expression $-(u \cdot \nabla)u$. At each time step all the fluid particles are moved by the velocity of the fluid itself. Therefore, to obtain the velocity at a point x at the new time t+Δt, we back trace the point x through the velocity field w_1 over a time Δt. The new velocity at the point x is then set to the velocity that the particle, now at x, had at its previous location a time Δt ago, which is shown in Figure 1. This method makes the solver unconditionally stable as the maximum value of the new field is never larger than the largest value of the previous field.

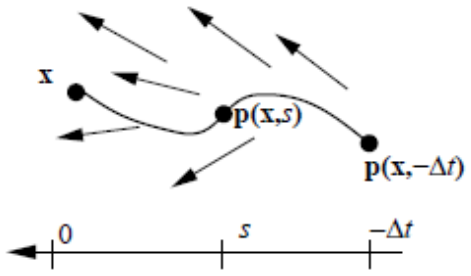

Figure 1: Advection

The third step solves for the effect of viscosity and is equivalent to a diffusion equation

$$\frac{\partial w_2}{\partial t} = \upsilon \nabla^2 w_2$$

We use an implicit solver

$$(I - \upsilon \Delta t \nabla^2) w_3(x) = w_2(x)$$

where I is the identity operator. When the diffusion operator is discretized, this leads to a sparse linear system for the unknown field w_3.

The fourth step involves the projection step, which makes the resulting field divergence free.

$$\nabla^2 q = \nabla \cdot w_3, \;\; w_4 = w_3 - \nabla q$$

We use Gauss-Seidel relaxation to solve for the pressure and make the velocity field divergent free.

IMPLEMENTATION

The major work in implementing the code was to get discrete finite difference counterparts of the equations. We used a 2D or 3D regular grid with constant spacing. However, the model became inefficient when we used the same number of cells for all dimensions as outlined in the paper. So we implemented a modified version of the solver which takes a different number of cells in each direction. This helped in running both the 2D and 3D solver in real time as we can adjust the grid size as required in each direction.

Add Force

This is the simplest term to implement. The Forcing matrix is used to add appropriate forces at different positions.

$$X[i, j, k] = X0[i, j, k] + \Delta t \cdot F[i, j, k]$$

Diffuse

For diffusion the 2D and 3D equation are

$$X[i, j] = \frac{1}{1+4a}\left(X0[i, j] + a\left(X[i-1, j] + X[i+1, j] + X[i, j-1] + X[i, j+1]\right)\right)$$

$$\begin{aligned} X[i, j, k] = \frac{1}{1+6a}(& X0[i, j, k] \\ & + a(X[i-1, j, k] + X[i+1, j, k] + X[i, j-1, k] \\ & + X[i, j+1, k] + X[i, j, k-1] + X[i, j, k+1])) \end{aligned}$$

where a = Δt*diff/(h*h), h is the grid spacing, Δt is the time step and diff is the diffusion coefficient.

Project

For projection the 2D and 3D equation are

$$X[i, j] = \frac{1}{4}\left(X0[i, j] + X[i-1, j] + X[i+1, j] + X[i, j-1] + X[i, j+1]\right)$$

$$\begin{aligned} X[i, j, k] = \frac{1}{6}(& X0[i, j, k] \\ & + X[i-1, j, k] + X[i+1, j, k] + X[i, j-1, k] \\ & + X[i, j+1, k] + X[i, j, k-1] + X[i, j, k+1]) \end{aligned}$$

Advect

In Stable Fluids, advection is calculated by tracing a particle positioned at each square's center back along the velocity field to see where it would have been at the end of the last time step. The easiest way to do this is to linearly interpolate the particle's position using its current velocity. This approach will no longer work when arbitrary boundaries are used. So we calculate how much time will be required for the particle to move half a square-width given its current speed, and do a linear interpolation using that value as Δt. We then recalculate the particle's velocity based on its new position in the velocity field, and repeat this pattern until we don't have enough time left to make it another half square width. Then we use up the remainder of the time on one last linear interpolation.

Boundary Conditions

We used rigid boundary conditions for the objects and the top and bottom edges. We used open boundaries in the flow direction. We also used no-slip conditions for the rigid bodies.

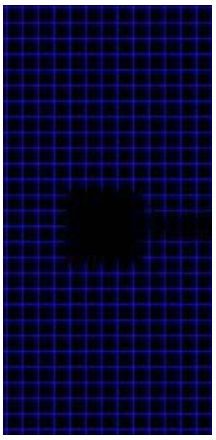

Figure 2: Wireframe view of a flow showing boundary conditions imposed inside the rigid body

In Figure 2 the value of the field is forced to be zero inside the rigid body and also the values just inside the boundary were made to be the negative of the values just outside to enforce no-slip condition.

Blending

The solver treats each density matrix separately. Hence, we can use many different fluids, each with different density matrix and see the interaction between them. We can use this to create good pseudo-random textures as shown in Figure 3.

Figure 3: Blending of two different gases

RESULTS

FLOW SIMULATIONS

Figure 4 show a flow simulation with a square object in the path of the flow. We used no-slip boundary conditions for the object and for the top and bottom edges. We can see the formation of the Von-Karman vortex street as the fluid flows past the object.

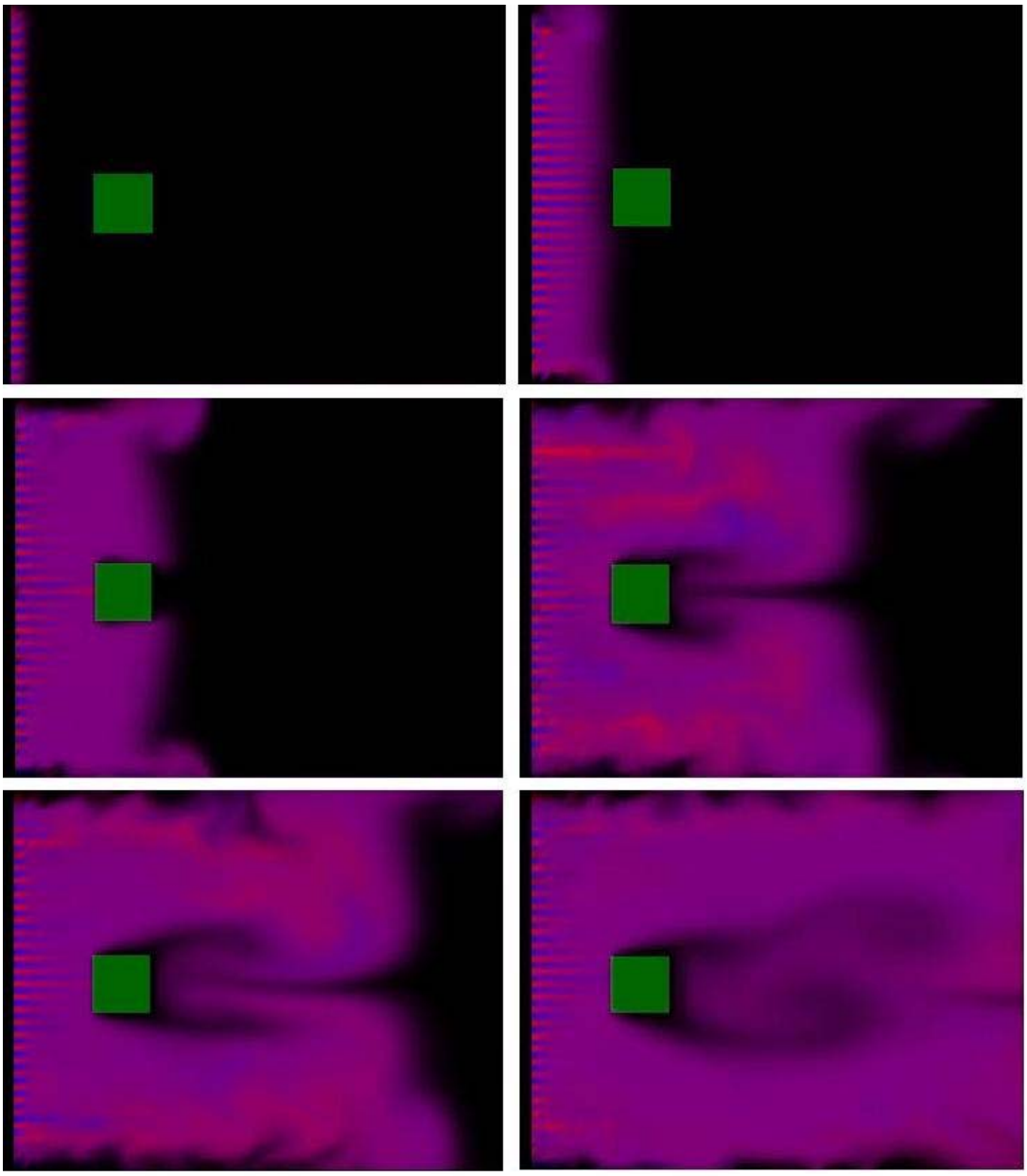

Figure 4: Flow past a square object

MULTIPLE FLOWS

Figure 5 shows the interaction between two different flows introduced at two opposite boundaries. We can see the mixing of the flows and also the interaction between them.

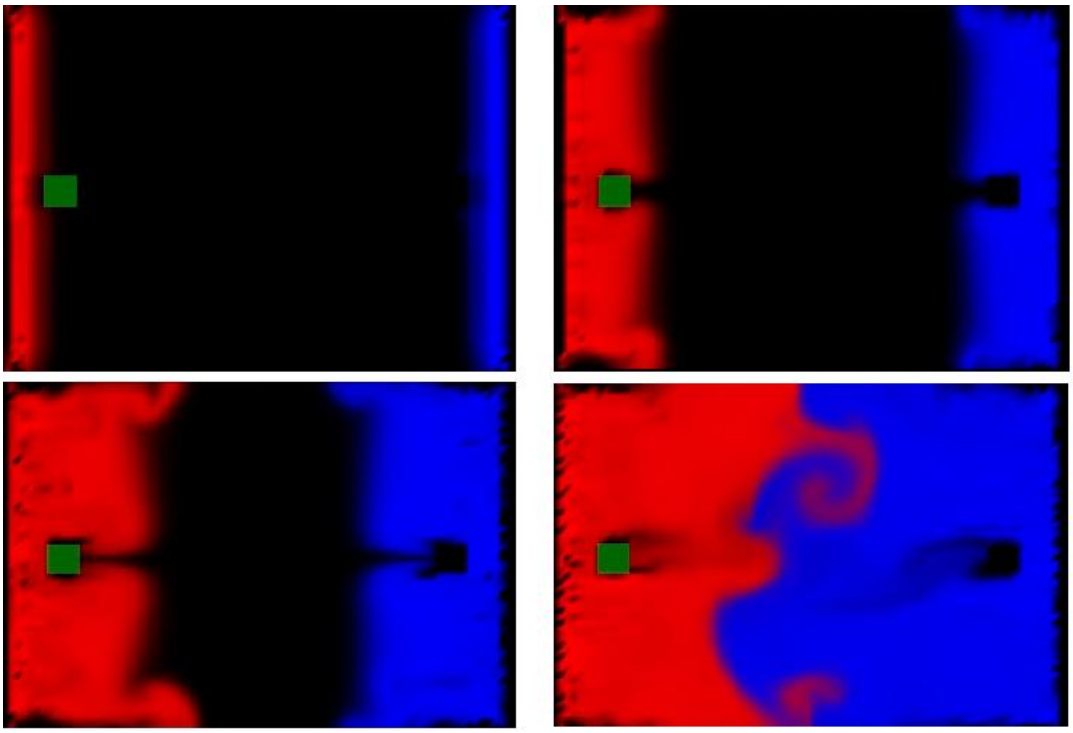

Figure 5: Mixing turbulent flows

3D

We extended the Navier Stokes solver to 3D using the equations given in the theory section above. The main requirement in 3D is to have a good volume renderer. We decided on using blended cubes with transparency because we wanted the simulator to run in real time.

CLOUDS

We used the simulator to animate a scene with clouds. We varied the alpha values of the clouds with height to emulate the sunlight filtering down the clouds and the resulting effect was good enough for a real time simulator. Figure 7 shows the 3D grid used to animate the clouds. The grid was made up of fewer cells in the vertical direction as we do not need much detail in this direction and also it helps to run the simulation faster.

Figure 6: Images of clouds rendered by using blending and transparency

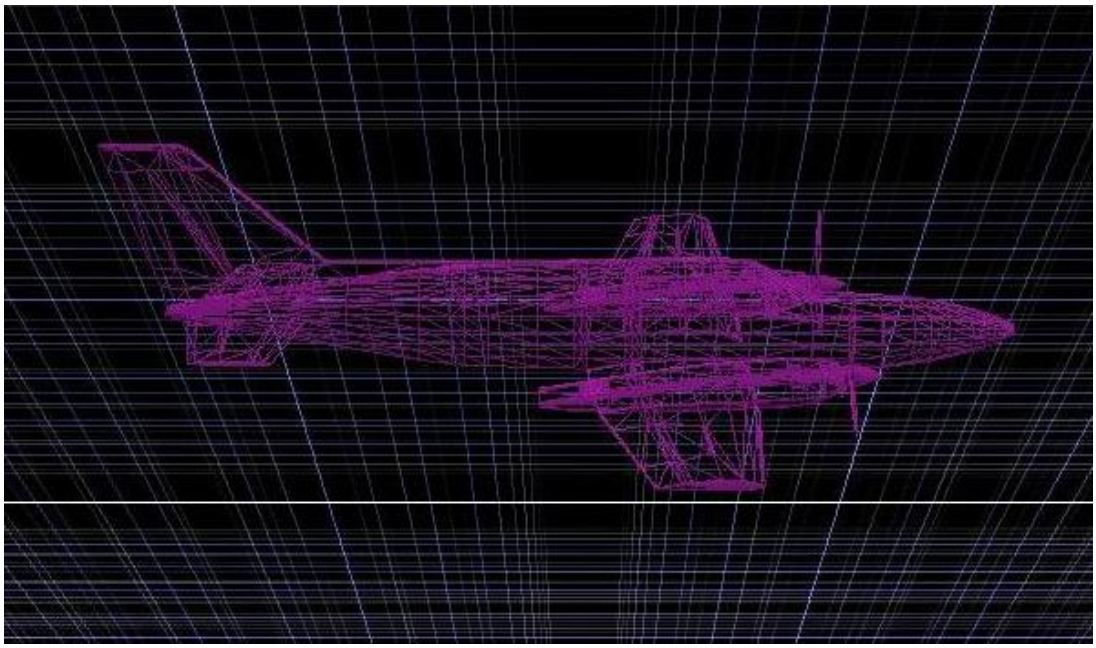

Figure 7: Image of the grid used to simulate clouds

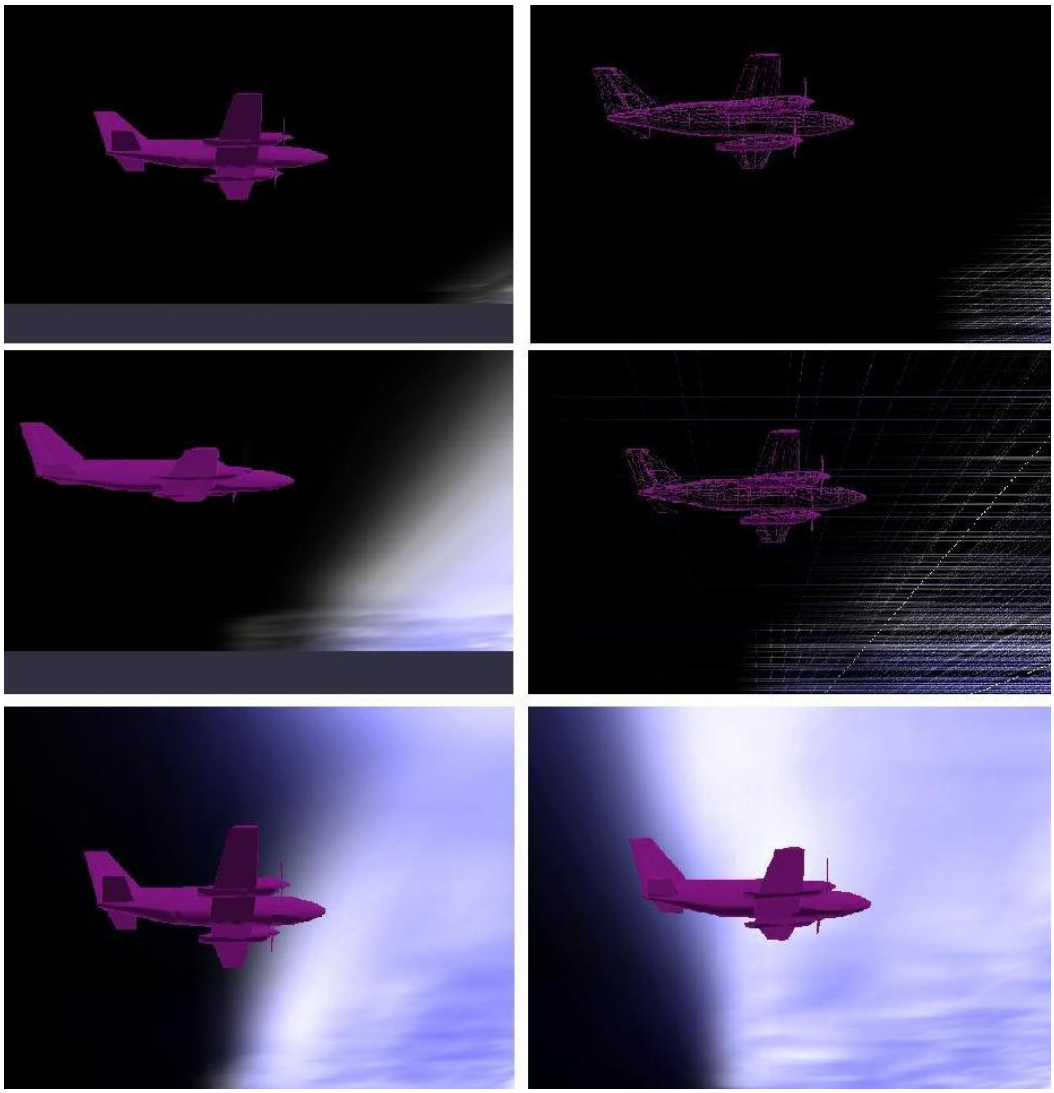

Figure 8: Sequence of images showing the formation of clouds

Figure 9: Fully rendered scene with a plane in the foreground

FIRE

One of the applications of a fluid solver is to render fire as it also exhibits fluid like motion. We used our simulator to render fire in 2D first as shown in Figure 10. We used sources located at the bottom edge with forces pointing up.

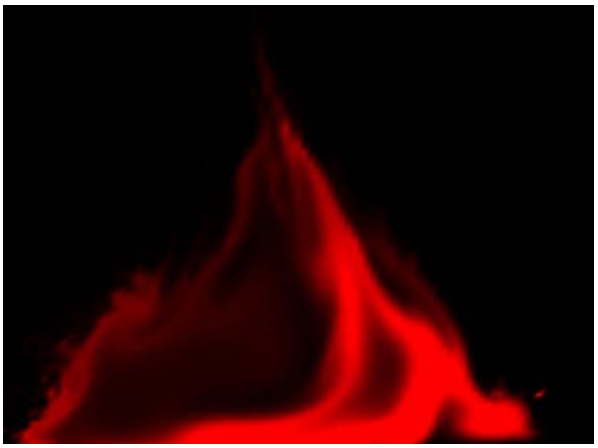

Figure 10: Animation of fire in 2D

When we first started to work in 3D, we ran into a problem of getting a realistic shape of the fire as the forces applied to the sources were constant. To resolve this issue, we added forces in the x and z directions with a random magnitude. This introduced realistic swaying motion to the fire. We also changed the alpha values based on the distance from the source to mimic the changes in emission color based on temperature. The final result is demonstrated in Figure 11. The grid used for the computation is shown in Figure 12 .

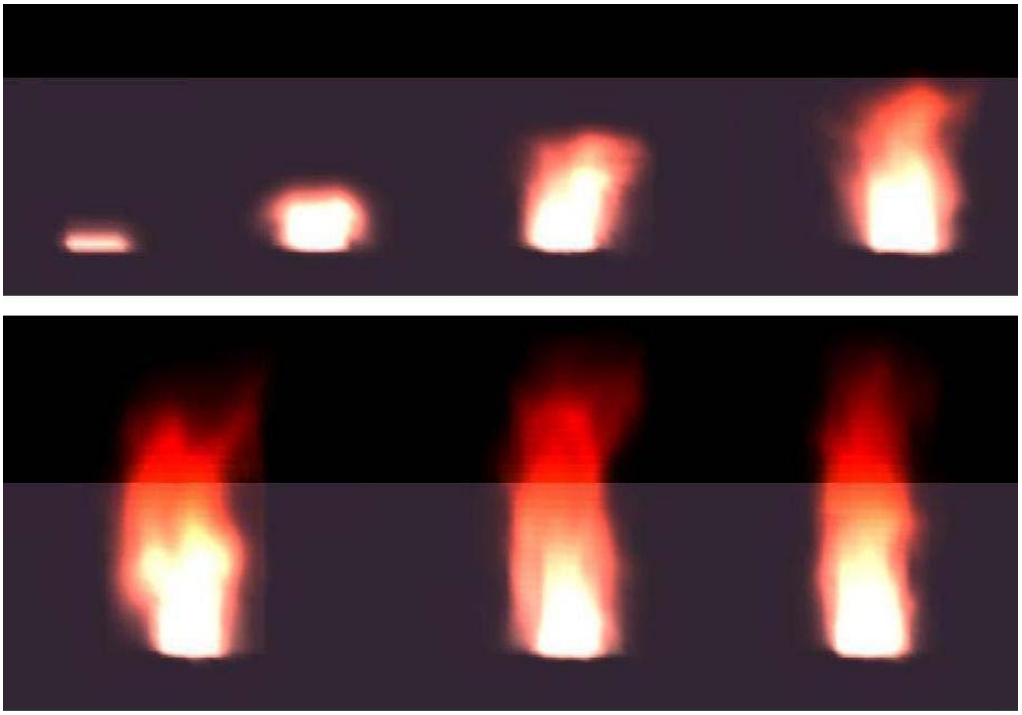

Figure 11: Animation of fire

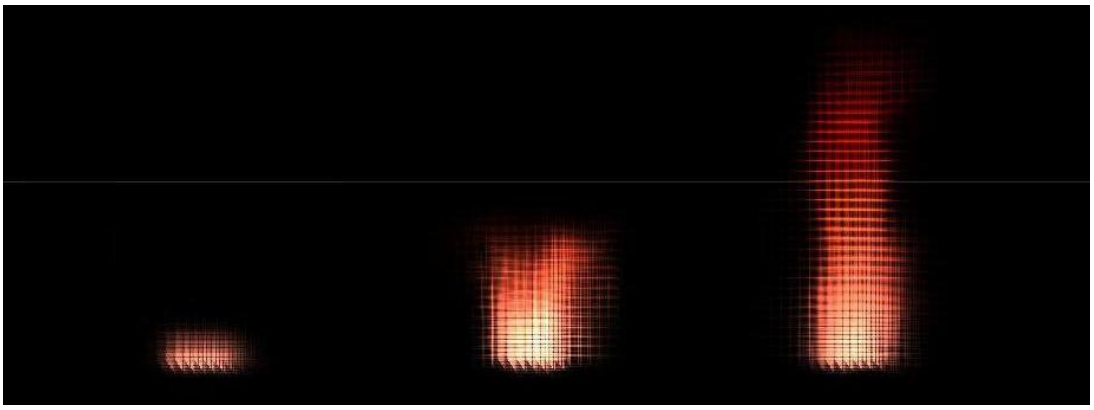

Figure 12: Grid used for the animation of fire

CONCLUSIONS

We have implemented a fluid solver to render physics based animations in real time. We started with a 2D solver and used it to simulate flow past a rectangular object as well as blending of 2 fluids. We extended the solver to 3D which we used it to render clouds and fire. We have achieved our basic objectives as well as some of our secondary goals.

This project helped us in understanding physics based animations and allowed us to get experience in implementing ideas from a technical paper. After analyzing different ways to achieve volume rendering, we used a blended cubes as it was the only method which could run in real time and produce good looking scenes. We learnt about alpha blending and transparency while working on the renderer.

Fluid simulators have a wide variety of applications. We noticed that we can use it to render motions produced by the wind by using the velocity at each cell. In particular we thought it would be a good idea to simulate a flag fluttering in the wind. Another extension to make the animations look better is to use ray marching to render the scenes. However it will make the simulations slower and they cannot be run in real time.

REFERENCES

1. J. Stam, “Stable Fluids,” SIGGRAPH 99 Conference Proceedings, Annual Conference Series, pages 121-128, August 1999.
2. J. Stam, “Real-Time Fluid Dynamics for Games.”
3. R. Fedkiw, J. Stam, and H. W. Jensen, “Visual Simulation of Smoke,” SIGGRAPH 2001 Conference Proceedings, Annual Conference Series, pages 15-22, August 2001.