Eulerian Smoke Simulation

Brett Lambright (i7263347)
MSc Computer Animation and Visual Effects
Bournemouth University - NCCA

16 August 2013

Contents

1 Introduction **1**

2 Previous Work **2**

3 Technical Background **4**

3.1 Simulating Fluid Flow 4

3.2 Eulerian Fluid Simulation 4

3.3 Navier-Stokes Equations 5

3.3.1 Operators 5

3.3.2 Terms of the Momentum Equation 6

3.4 Breakdown of Momentum Equation to Differential Equations 7

3.4.1 Convection Term to Differential Equations 8

3.4.2 Viscosity Term to Differential Equations 8

3.4.3 Putting It All Together 9

3.5 Discretization 9

3.5.1 Finite Differences 10

3.5.2 First and Second Order Derivative Approximations 11

4 Solution **12**

4.1 Algorithm 12

4.2 Emitting Energy 12

4.3 Dissipation 13

4.4 External Forces 13

4.4.1 Buoyancy Force 13

4.4.2 Noise/Turbulence 14

4.5 Vorticity Confinement 15

4.6 Advecting the Velocity Field 16

4.7 Pressure 17

4.7.1 Calculating Pressure 17

4.7.2 Applying Pressure 18

4.8 Convecting Energy 18

4.9 Boundary Conditions 19

4.10 Obstacles 19

4.11 Visualizing the Smoke 20

4.12 Stability 21

4.13 Design 21

4.14 Problems 22

5 Conclusion **24**

Appendices **27**

A Derivative Approximations **28**

A.1 First Order Derivative Approximations 28

A.2 Second Order Derivative Approximations 28

B Boundary Conditions **30**

B.1 No-Slip Boundary Condition 30

B.2 Free-Slip Boundary Condition 31

B.3 Outflow Boundary Condition 32

C Class Diagrams **33**

List of Figures

3.1 The 3D space divided into evenly spaced voxels. (Fedkiw et al. 2001) 9
3.2 Location of the velocity components (u,v, and w) in a voxel. (Cline et al. 2004) . . 10
3.3 Exchanging energy with neighbours (Stam 2003) 10

4.1 Smoke with a low dissipation value (left) and a high dissipation value (right) . . . 13
4.2 Smoke with no buoyancy force (left) and a high buoyancy force (right) 14
4.3 Smoke with no turbulence force (left) and high turbulence force (right) 15
4.4 Advection of the velocity field. 16
4.5 Subtracting the pressure gradient from the velocity field to get a divergence-free field (Stam 2003). 17
4.6 Temperature being convected through the velocity field. 18
4.7 Setting velocity and temperature conditions at a boundary (Foster et al. 1997) . . 19
4.8 The smoke interacting with obstacles. 20
4.9 The sprites being drawn at each voxel. 21
4.10 The high-level class diagram of the smoke simulator. 22

C.1 The MainWindow class. 33
C.2 The Obstacle class. 33
C.3 The VoxelGrid class. 34
C.4 The SmokeSystem class. 35
C.5 The GLWindow class. 36
C.6 The Emitter class. 37
C.7 The Sprite struct. 37
C.8 The Voxel struct. 37

Abstract

Fluid simulation has been a popular topic over the last few decades now that computers are fast enough to perform the necessary calculations in a reasonable amount of time. This thesis is on the topic of incompressible Eulerian smoke simulation and covers the Navier-Stokes equations, the expanding of the equations into differential equations, and the discretization to simulate the equations on the computer. This thesis also covers the implementation of a basic smoke solver that is comprised of elements and techniques from a variety of sources. After reading this report, one would have enough knowledge to implement their own basic smoke solver that calculates the buoyant, turbulent and rotational motions evident in real smoke. The research topics and methods in this thesis have already been implemented in other smoke solvers however there are areas and aspects that can be improved. These improvements are discussed in this report.

Chapter 1

Introduction

The flow of fluids can be observed in everyday life. Examples of fluid flow include rivers, waterfalls, smoke, lava flow, and the air flow around a plane's wings. Two elemental properties govern the motion of a fluid: viscosity and inertia. Griebel et al (1998) used laminar flow to describe the viscosity and inertial forces. Imagine individual cards stacked on top of each other that are all moving in one direction. If the bottom card abruptly stops, the cards above will keep moving forward due to the inertial force. Friction between the cards will eventually bring all of the cards to a stop and the resulting pattern of the cards will look like a set of stairs. This friction between cards can be thought of as the viscosity of the fluid. Viscosity is a physical property of fluids that generates frictional forces that eventually brings the motion of a fluid to a rest (Griebel et al. 1998). Liquids like molasses have high viscosities and their motion comes to rest far sooner than fluids that have lower viscosities like air and water. For gases the viscosity is so low that it can be disregarded in the simulation. Because of this, gases are known as inviscid fluids.

Compressible fluids can be compressed. This means that "fluid of the same mass does not always occupy the same volume" (Griebel et al. 1998). On the other hand, incompressible fluids cannot be compressed. The mass of incompressible fluids "will always occupy the same volume" (Griebel et al. 1998) and the total density will remain constant. This report will only cover incompressible fluid flow.

This thesis is on incompressible Eulerian smoke simulation. The topics that will be discussed include an overview of Eulerian fluid simulation, the Navier-Stokes equations, a breakdown of the equations to differential equations, and the discretization process. This report also explains all of the steps to build a basic smoke solver like advecting the velocity field, convecting energy, and calculating the pressure. It also covers topics like vorticity confinement, buoyancy, turbulence, stability, boundary conditions, and obstacles. To conclude the report, the design and problems of this smoke solver are discussed.

Chapter 2

Previous Work

The Marker and Cell (MAC) method was developed by Francis Harlow and J. Eddie Welch in 1965. It made use of finite-difference approximations of the Navier-Stokes equations with an explicit time integrator (Harlow and Welch, 1965). The method was originally designed for 2D fluid simulation but can be translated to three dimensions. Particles (markers) are traced through the velocity field to represent the flow of a fluid. Harlow and Welch's work was highly referenced in most of the conducted research on fluid simulation that has taken place over the last few decades and is still referenced in recent reports.

In 1997, Foster and Metaxas wrote a paper called *Modeling the Motion of a Hot, Turbulent Gas*. This was one of the first papers to address the turbulent, buoyant, and rotational motion of gas in three dimensions. Their model accounted for convection, turbulence, buoyancy, vorticity and smoke flowing around obstacles (Foster et al. 1997). Their model solved the Poission pressure equation using a modified version of Harlow and Welch's solution. The drawback to their approach is that it is stable only when the time step is considerably small. The change in time, Δt, needs to be less than $h/|\mathbf{u}|$ where h is the width of the voxel.

In 1998, Griebel et al. wrote a book on numerical simulation called *Numerical Simulation in Fluid Dynamics: a Practical Introduction* (Griebel et al. 1998). The method in the book was based off of Harlow and Welch's marker and cell method and it used an explicit time integrator. It also used the donor-cell discretization along with central differences for extra stability during the advection step. The book covered the mathematical description of flows, the numerical treatment of the Navier-Stokes equations and everything from boundary conditions to turbulent flows and parallelization.

In 1999, Jos Stam wrote a SIGGRAPH paper called "Stable Fluids." It described his method which uses a semi-Lagrangian advection technique and solves the partial differential equations using the "method of characteristics" (Stam 1999) Stam's fluid solver used an implicit integration of the viscosity which resulted in an unconditionally stable solver. This was a big improvement to Foster and Metaxas' solver which required small time steps to be stable. The unconditional stability allowed for larger time steps which resulted in faster simulations.

In 2001, Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen wrote a SIGGRAPH paper called *Visual Simulation of Smoke*. Their model was stable, fast, and properly dealt with numerical dissipation (Fedkiw et al. 2001). This paper introduced vorticity confinement which is a force

that puts energy back into the swirls that was previously lost due to numerical dissipation. The new energy made swirls more evident in the simulation which led to a much more realistic result.

Chapter 3

Technical Background

3.1 Simulating Fluid Flow

Fluid mechanics is governed by the Navier-Stokes equations, ”a precise mathematical model for most fluid flows occurring in nature” (Stam 2003). They are a description of the evolution of a velocity field over time (Stam 2003). The ultimate goal is to compute the Navier-Stokes equations numerically. There are hundreds of books published with different methods of solving these equations (Stam 1999). Fluid solvers should enable the user to achieve fluid-like effects in real-time (Stam 1999). Physical accuracy is not as important when it comes to computer graphics because it can be computationally expensive. What matters most is that the simulations are fast and believable (Stam 2003). The simulation needs to demonstrate the turbulent, buoyant, or rotational motion that develops as a gas interacts with itself and obstacles (Foster et al. 1997).

This implementation is influenced heavily by the marker-and-cell (MAC) method. The MAC method uses finite differences with an explicit time integration. However there are no particles (markers) in this simulation because of the fact that this solver uses the Eulerian fluid simulation technique, a topic that is covered in the next section.

3.2 Eulerian Fluid Simulation

In a Eulerian fluid simulator the 3D space is split up into a grid of evenly spaced cubes called voxels. A notable difference between other fluid simulation methods is that there are no particles traced through the velocity field. Each voxel has a scalar value associated to it that changes depending on the fluid flow out of and into the voxel which is determined by the velocity field.

In the case of smoke, it would be impossible to simulate every particle. If particles are used, ”it would be unclear how interaction between volumes of gas is modeled using forces between particles” (Foster et al. 1997). Also, gas is a continuous medium and needs to mix with its surrounding atmosphere. This cannot be achieved with the use of particles.

Instead of using particles, a scalar density value is used at each voxel. The density value is a

value between zero and one. A value of zero implies that there is no smoke and a value above zero indicates that there is smoke present (Stam 2003). The velocity and sources of energy at each voxel are pushed, or convected, by their neighbours (Foster et al. 1997).

3.3 Navier-Stokes Equations

$$\frac{\partial \mathbf{u}}{\partial t} = -(\nabla \mathbf{u})\mathbf{u} - \frac{1}{\rho}\nabla p + \lambda \nabla^2 \mathbf{u} + \mathbf{F} \tag{3.1}$$

$$\nabla \mathbf{u} = 0 \tag{3.2}$$

The two differential equations above are the Navier-Stokes equations for incompressible fluids. The equations describe the velocity field of a fluid, **u**, over time (Cline et al. 2004). Bold is used to indicate a vector quantity and italic is used to indicate a scalar quantity. λ is the kinematic viscosity of the fluid, ρ is the density and **F** is the external forces acting on the fluid. A term that is frequently used in fluid simulations is the Reynolds number, Re, which is basically the ratio of inertial forces to viscous forces (Griebel et al. 1998). The kinematic viscosity, λ, is equivalent to $\frac{1}{Re}$.

Equation 3.1 is for the conservation of momentum. It accounts for the internal and external forces that act on the fluid (Cline et al. 2004). This equation models the convective, viscous, and rotational motion in the smoke (Foster et al. 1997). Equation 3.1 is broken down into its individual terms in section 3.3.2.

Equation 3.2 is for the conservation of mass. This equation implies that the amount of fluid flowing into any volume must be equal to the fluid flowing out of that volume (Cline et al. 2004). In other words, the divergence of the velocity field must be equal to 0 (Cline et al. 2004).

For the purposes of the discretization which will be discussed in Section 3.5, the velocity vector field **u** needs to be separated into three scalar values for each component of its velocity vector. These three scalar values will be called u,v, and w. u will represent the x direction, v will represent the y direction, and w will represent the z direction.

3.3.1 Operators

There are two operators that are in the Navier-Stokes equations. These include the gradient operator: ∇ and the Laplacian operator: ∇^2. The dot product of the gradient operator and a vector field, $\nabla \cdot$, is called the divergence of a vector field.

3.3.1.1 Gradient of a Scalar Field (∇)

$$\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}) \tag{3.3}$$

The gradient operator tells how a scalar field changes spatially (Cline et al. 2004). It is a vector field that points in the direction of the rate of increase. The two scalar fields in this smoke solver are density and temperature.

3.3.1.2 Divergence of a Vector Field ($\nabla\cdot$)

$$\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \tag{3.4}$$

The divergence of a vector field is the dot product of the gradient operator and a vector field. Its result is a scalar field that is "the net flow out of or into points in the vector field" (Cline et al. 2004). The mass conservation equation (Equation 3.2) basically says that the divergence of the velocity vector field needs to be 0.

3.3.1.3 Laplacian (∇^2)

$$\nabla^2 = \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{3.5}$$

"The Laplacian operator is the dot product of two gradient operators" (Cline et al. 2004). Applying the Laplacian operator to a scalar field basically describes how the values in the scalar field differ from their neighbourhood average (Cline et al. 2004).

3.3.2 Terms of the Momentum Equation

The conservation of momentum equation, Equation 3.1, can be broken down into four terms: convection, viscosity, pressure, and external force.

3.3.2.1 Convection

$$-(\nabla \mathbf{u})\mathbf{u} \tag{3.6}$$

This term is the convection term. The momentum of the fluid must be convected through space along with the fluid itself (Cline et al. 2004).

3.3.2.2 Pressure

$$-\frac{1}{\rho}\nabla p \tag{3.7}$$

This term is the pressure gradient. The pressure is important for ensuring that the divergence of the velocity field is 0. If the velocity field is divergence-free, the mass will be conserved meaning that there will be no mass gained and/or lost during the simulation.

3.3.2.3 Viscosity

$$\lambda\nabla^2\mathbf{u} \tag{3.8}$$

This term can also be thought of as drag. It is the internal friction of the fluid. The fluid moves slower with higher viscosity values. This term can be disregarded in a smoke solver because gas is inviscid.

3.3.2.4 External Force

$$\mathbf{F} \tag{3.9}$$

This term accounts for external forces like gravity, buoyancy, wind, turbulence, and vorticity and is simply added onto the end of the momentum equation.

3.4 Breakdown of Momentum Equation to Differential Equations

The Navier-Stokes equations can be expanded into a series of first and second order differential terms (Foster et al. 1997). This section walks through the steps to break down the momentum equation into the final expanded differential equations.

Recall the Navier-Stokes equation for the conservation of momentum:

$$\frac{\partial\mathbf{u}}{\partial t} = -(\nabla\mathbf{u})\mathbf{u} + \lambda\nabla^2\mathbf{u} - \frac{1}{\rho}\nabla p + \mathbf{F} \tag{3.1}$$

Equation 3.1 can be split into an equation for each of the three velocity components of **u**: u, v, and w.

$$\frac{\partial u}{\partial t} = -(\nabla \mathbf{u})u + \lambda \nabla^2 u - \frac{1}{\rho}\nabla p + F_u \tag{3.10a}$$

$$\frac{\partial v}{\partial t} = -(\nabla \mathbf{u})v + \lambda \nabla^2 v - \frac{1}{\rho}\nabla p + F_v \tag{3.10b}$$

$$\frac{\partial w}{\partial t} = -(\nabla \mathbf{u})w + \lambda \nabla^2 w - \frac{1}{\rho}\nabla p + F_w \tag{3.10c}$$

3.4.1 Convection Term to Differential Equations

The first step is to expand the convection term into three differential equations.

Convection term = $-(\nabla \mathbf{u})\mathbf{u}$

Recall the divergence of a vector field differential equation:

$$\nabla \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \tag{3.4}$$

Using the divergence equation, we can determine the differential equations of the convection term for u, v, and w.

$$-(\nabla \mathbf{u})u = -\frac{\partial uu}{\partial x} - \frac{\partial uv}{\partial y} - \frac{\partial uw}{\partial z} \tag{3.11a}$$

$$-(\nabla \mathbf{u})v = -\frac{\partial vu}{\partial x} - \frac{\partial vv}{\partial y} - \frac{\partial vw}{\partial z} \tag{3.11b}$$

$$-(\nabla \mathbf{u})w = -\frac{\partial wu}{\partial x} - \frac{\partial wv}{\partial y} - \frac{\partial ww}{\partial z} \tag{3.11c}$$

3.4.2 Viscosity Term to Differential Equations

The next step is to expand the viscosity term into differential equations.

Viscosity term = $\lambda \nabla^2 \mathbf{u}$

Recall the Laplacian differential equation:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{3.5}$$

We can use the Laplacian equation to break down the viscosity term into differential equations for u, v, and w.

$$\lambda\nabla^2 u = \lambda(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}) \tag{3.12a}$$

$$\lambda\nabla^2 v = \lambda(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}) \tag{3.12b}$$

$$\lambda\nabla^2 w = \lambda(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}) \tag{3.12c}$$

3.4.3 Putting It All Together

Now that the partial derivative equations of the convection and viscosity terms have been determined, they can be substituted into the momentum equation to get the final differential equations needed before the discretization step.

$$\frac{\partial u}{\partial t} = -\frac{\partial uu}{\partial x} - \frac{\partial uv}{\partial y} - \frac{\partial uw}{\partial z} + \lambda(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}) - \frac{1}{\rho}\nabla p + F_u \tag{3.13a}$$

$$\frac{\partial v}{\partial t} = -\frac{\partial vu}{\partial x} - \frac{\partial vv}{\partial y} - \frac{\partial vw}{\partial z} + \lambda(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}) - \frac{1}{\rho}\nabla p + F_v \tag{3.13b}$$

$$\frac{\partial w}{\partial t} = -\frac{\partial wu}{\partial x} - \frac{\partial wv}{\partial y} - \frac{\partial ww}{\partial z} + \lambda(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}) - \frac{1}{\rho}\nabla p + F_w \tag{3.13c}$$

3.5 Discretization

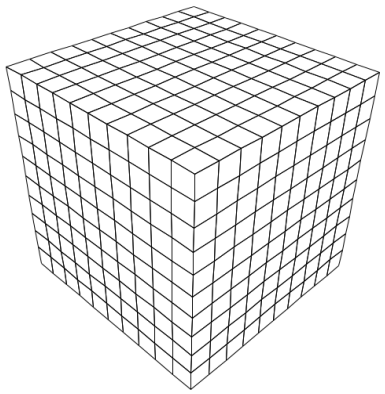

Figure 3.1: The 3D space divided into evenly spaced voxels. (Fedkiw et al. 2001)

The 3D space needs to be discretized into a grid of cube-shaped voxels like in Figure 3.1. h is the width of each voxel and i, j, and k are the x, y, and z indices of the voxel in the 3D grid of voxels. Figure 3.2 shows the location of the velocity components u,v, and w in a voxel. They are located on the minimal faces of the voxel. The density, temperature, and pressure values are stored at the center of the voxel.

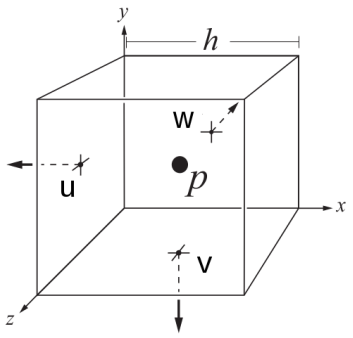

Figure 3.2: Location of the velocity components (u,v, and w) in a voxel. (Cline et al. 2004)

Now that the Navier-Stokes momentum equation are expanded into three differential equations (Equation 3.13), there needs to be a finite approximation scheme for the discretization. Each partial derivative translates to a simple algebraic equation involving the adding and subtracting of the values of neighbouring voxels.

3.5.1 Finite Differences

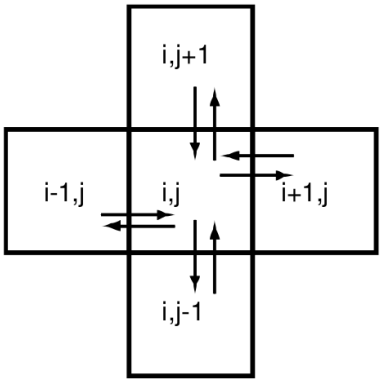

Figure 3.3: Exchanging energy with neighbours (Stam 2003)

The finite difference approximation scheme must preserve the turbulent and swirliness of gaseous motion (Foster et al. 1997).

There are three methods of finite differences: backward, forward, and central.

3.5.1.1 Backward Difference

For the backward difference, the previous cell is subtracted from from the current cell.

$$\partial u/\partial x = \frac{u(i,j,k) - u(i-1,j,k)}{h} \tag{3.14}$$

3.5.1.2 Forward Difference

For the forward difference, the current cell is subtracted from the next cell.

$$\partial u/\partial x = \frac{u(i+1,j,k) - u(i,j,k)}{h} \tag{3.15}$$

3.5.1.3 Central Difference

For the central difference, the previous cell is subtracted from the next cell.

$$\partial u/\partial x = \frac{u(i+1,j,k) - u(i-1,j,k)}{2h} \tag{3.16}$$

3.5.2 First and Second Order Derivative Approximations

Each partial derivative in the momentum differential equations (Equation 3.13) can be mapped to a finite difference approximation using forward, backward, or central differences. The first order derivative approximations using forward differences are listed in Appendix A, Section A.1 and the second order derivative approximations are listed in Appendix A, Section A.2.

Chapter 4

Solution

This smoke solver uses a finite difference approximation scheme with an explicit time integration method (RK4). The advection function is based on the Griebel et al. (1998) solver and uses central differences with the donor-cell discretization for stability purposes. To solve for the pressure, it uses the Geiss-Seidel successive relaxation technique also used by Griebel et al. (1998). To calculate the buoyancy force, this solver uses the equation provided by Cline et al. (2004) and for the turbulence force, the libnoise[1] library by Jason Bevins provided the noise calculations. The vorticity confinement force equation was provided by Fedkiw et al. (2001).

4.1 Algorithm

The following steps are carried out at each time step.

1. Emit energy.
2. Apply external forces (buoyancy, turbulence, gravity, etc.).
3. Calculate and apply the vorticity confinement force.
4. Advect the velocity field.
5. Calculate the pressure using a relaxation technique.
6. Apply the pressure.
7. Force the velocity of the voxels inside the obstacle (if any) to 0.
8. Convect and dissipate the energy.

4.2 Emitting Energy

For smoke, there are two sources of energy that need to be emitted and convected in the simulation: density and temperature. In this implementation, the user is able to change the position and size of the emitter. The user can also change the rate that the density and temperature are being emitted. The emitter is associated to a cube of voxels based on the emitter's position and size. Interpolation with the sine and power functions work to emit the energy in a semi-spherical shape inside the cube of voxels.

[1] http://libnoise.sourceforge.net/

4.3 Dissipation

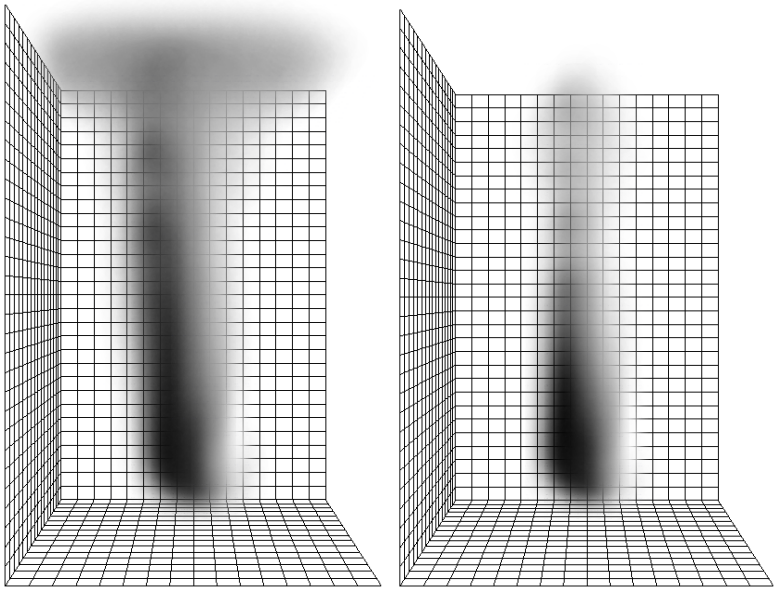

Figure 4.1: Smoke with a low dissipation value (left) and a high dissipation value (right)

Dissipation is necessary for smoke because of the fact that smoke evaporates and mixes with the atmosphere over time. Also the temperature of the smoke gets cooler over time. Dissipation is easy to implement because it is basically the subtraction of a uniform value at each time step from the scalar density value at each voxel. The following equation is used for the dissipation of the smoke's density:

$$\rho_{n+1} = max(0, \rho_n - k_{dissipation} \cdot \Delta t) \tag{4.1}$$

(Cline et al. 2004)

ρ is the density of the smoke and $k_{dissipation}$ is a user-defined constant controlling the amount of dissipation. Figure 4.1 shows smoke with a low dissipation value and a high dissipation value.

4.4 External Forces

4.4.1 Buoyancy Force

Smoke tends to rise and fall depending on its density and temperature. If the smoke is hotter and/or less dense than the surrounding air, it rises. If the smoke is colder and/or more dense than the surrounding atmosphere, it falls. Equation 4.2 was used to calculate the buoyancy force:

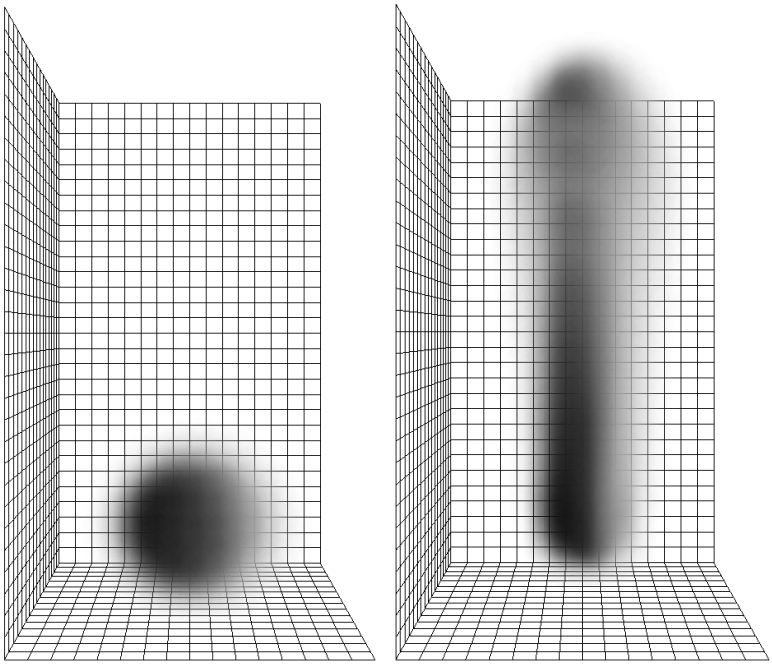

Figure 4.2: Smoke with no buoyancy force (left) and a high buoyancy force (right)

$$\mathbf{F}_{buoyancy} = (k_{rise} \cdot (t - t_{atmosphere}) + (-1 \cdot k_{fall}) \cdot \rho) \cdot \frac{\textbf{gravity}}{||\textbf{gravity}||} \tag{4.2}$$

(Cline et al. 2004)

t is the temperature of the smoke, $t_{atmosphere}$ is the temperature of the atmosphere, ρ is the density of the smoke, and k_{rise} and k_{fall} are user-defined constants for the rising and falling of the smoke. The gravity force vector does not affect the x and z components of the buoyancy force so the buoyancy force is applied either straight up or straight down. Figure 4.2 shows how the buoyancy force affects the smoke.

4.4.2 Noise/Turbulence

After emitting energy and applying a buoyancy force, the smoke uniformly rises up in the air. There needs to be some way to add a little variation to the way the smoke rises and falls to make the simulated smoke more smoke-like. The best way is to use noise. An excellent library to use is libnoise[2] by Jason Bevins. This smoke solver uses libnoise's perlin noise and turbulence feature. The noise is used in conjunction with the buoyancy force to add some variance to the motion of the smoke. Because of the use of the libnoise library, the user can change the frequency, octave count, persistence, and power of the noise. The user can also change the speed of the noise which is basically how fast the noise evolves over time.

[2] http://libnoise.sourceforge.net/

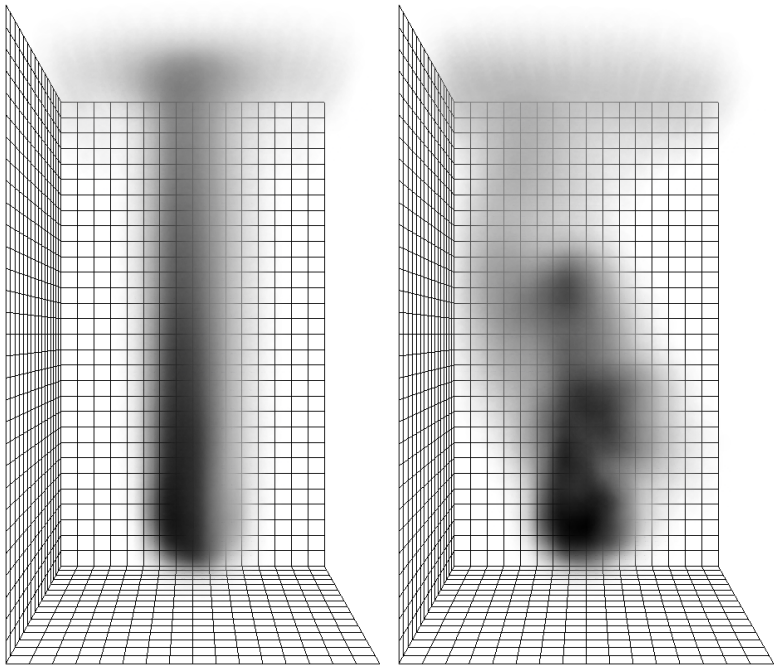

Figure 4.3: Smoke with no turbulence force (left) and high turbulence force (right)

4.5 Vorticity Confinement

Numerical simulation results in a loss of energy (Fedkiw et al. 2001). The swirls that are evident in real smoke are not as evident in smoke that is numerically simulated because of this "numerical dissipation" (Fedkiw et al. 2001). To maintain the swirliness, there is a principle called vorticity confinement. Vorticity confinement results in a force that adds energy back into the simulation to make up for the loss of energy (Fedkiw et al. 2001).

To calculate this force, there are a series of steps that need to be taken. The first step is to calculate the curl. The curl is the cross product of the gradient operator and the velocity field ($\nabla \times \mathbf{u}$) and is calculated using Equation 4.3.

$$\nabla \times \mathbf{u} = (\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}) \tag{4.3}$$

(Cline et al. 2004)

The next step is to create a scalar field of the magnitude of the curl, n, and use that value to calculate the gradient direction, **N** (Cline et al. 2004).

$$\mathbf{N} = \frac{\nabla n}{||\nabla n||} \tag{4.4}$$

(Fedkiw et al. 2001)

The gradient direction, **N**, is a "normalized vorticity location vector field that points from lower

vorticity concentrations to higher vorticity concentrations" (Fedkiw et al. 2001). The last step is to calculate the vorticity force which is a force in the direction of the swirl using Equation 4.5.

$$\mathbf{F}_{vorticity} = k_{vorticity} \cdot (\mathbf{N} \times (\nabla \times \mathbf{u})) \tag{4.5}$$

(Cline et al. 2004)

$k_{vorticity}$ is a user-specified constant to control the magnitude of the vorticity confinement force. After the force is applied and the velocity field is updated, swirls are more evident in the simulation.

4.6 Advecting the Velocity Field

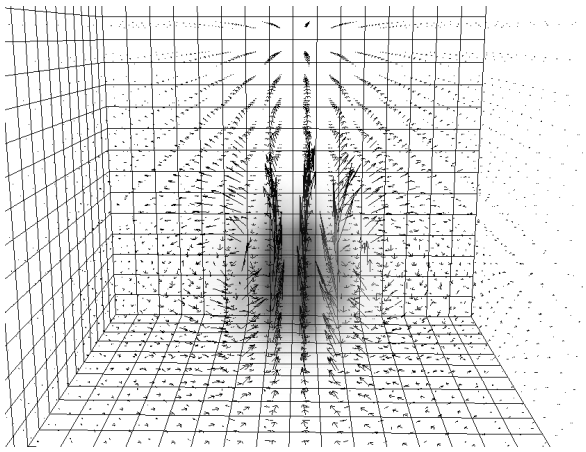

Figure 4.4: Advection of the velocity field.

The velocity field needs to be advected to satisfy the conservation of momentum equation (Equation 3.1). The goal is to use finite differences to solve Equation 3.13 without the pressure and external force terms. The first order and second order finite difference approximations are in Appendix A. After substituting the finite difference approximations into Equation 3.13, the resulting velocity field can be calculated. This result is then stored in the voxel so it can be accessed later to compute the new velocity after the pressure is calculated.

When central differences are used, oscillations occur in convection-diffusion problems (Griebel et al. 1998). A way to avoid stability problems is to use the donor-cell discretization scheme described by Griebel et al. (1998). It involves using values at the spatial interval mid-points to calculate a more accurate result.

4.7 Pressure

The pressure term is important for satisfying the mass conservation equation of the Navier-Stokes equations (Equation 3.2). Its purpose is to maintain a divergence-free velocity field at every time step. If the velocity field is not divergence-free, mass will be lost and/or gained during the simulation. The pressure field needs to be calculated and then applied to the velocity field.

4.7.1 Calculating Pressure

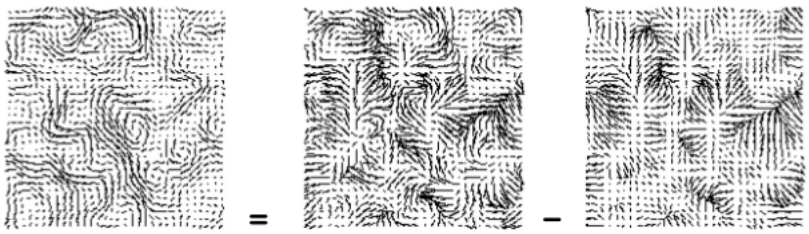

Figure 4.5: Subtracting the pressure gradient from the velocity field to get a divergence-free field (Stam 2003).

To ensure that the divergence of the velocity field is 0 and satisfy the conservation of mass equation (Equation 3.2), the smoke solver needs to have a solution to Poisson's pressure equation. There are several ways to solve the pressure equation however this solver uses a relaxation technique. More specifically, this solver uses the Geiss-Seidel method with successive overrelaxation. This relaxation method is an iterative solution. "Each cell is successively processed once in every cell in such a way that the equation is solved exactly" (Griebel et al. 1998). The following pseudocode is the relaxation method described by Griebel et al. (1998):

```
iter = 0,...,iterMax
        i = 0,...,iMax,
                j = 0,...,jMax,
                        k = 0,...,.kMax,
                                -Calculate pressure using central differences
                                -Calculate residual
        -Calculate residual norm
        -Copy calculated pressures to grid
        -If residual norm is lower than absolute tolerance, break out of loop
```

The residual is calculated for each voxel and the iteration stops when the norm of the residual falls below an absolute tolerance value or if the number of iterations exceeds the max number of iterations (Griebel et al. 1998).

4.7.2 Applying Pressure

Applying the pressure is a matter of subtracting the calculated value of the pressure gradient from the vector field computed in the advection step. This can be visualized in Figure 4.5. After the pressure is subtracted, the velocity field should be at least close to divergence-free.

4.8 Convecting Energy

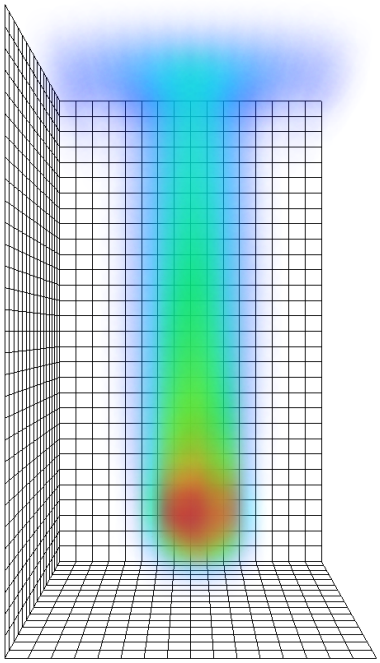

Figure 4.6: Temperature being convected through the velocity field.

The energy emitted into the simulation, density and temperature, need to be convected through the velocity field at each time step. The scalar energy values at each voxel are pushed by the velocities of the neighbouring voxels. The energy is drawn into areas of greater velocity and lower pressure (Foster et al. 2001). Equation 4.6 is the equation for the convection of energy.

$$\frac{\partial E}{\partial t} = -(\nabla E)\mathbf{u} + \lambda \nabla^2 E \tag{4.6}$$

(Foster et al. 2001)

E is the scalar energy value and λ is a constant that controls the magnitude of the diffusion process. As you can see, this equation is similar to the conservation of momentum equation (Equation 3.1) except for the fact that it does not have a pressure and external force term.

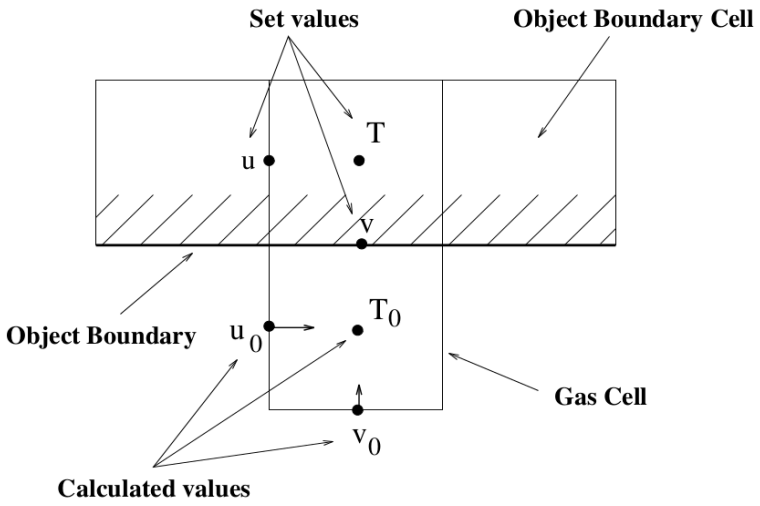

Figure 4.7: Setting velocity and temperature conditions at a boundary (Foster et al. 1997)

4.9 Boundary Conditions

A boundary is an edge/wall of the voxel grid domain. There are four boundaries in a 2D fluid simulator and six boundaries in a 3D fluid simulator. The condition of the boundary strongly influences the flow of the fluid. Below are three boundary conditions that are implemented in this smoke solver.

The *no-slip* boundary condition: all of the fluid next to the boundary is motionless and doesn't flow through the boundary (Griebel et al. 1998). The velocity components of **u** for the no-slip boundary condition can be found in Appendix B, Section B.1.

The *free-slip* boundary condition: the fluid flows freely next to a boundary without any friction and does not flow through the boundary (Griebel et al. 1998). This can be thought of as a "non-adhering ('greased') surface" (Harlow and Welch 1965). The velocity components of **u** for the free-slip boundary condition can be found in Appendix B, Section B.2.

The *outflow* boundary condition: the fluid flows freely out through the boundary. This condition basically acts as if the boundary does not exist. The velocity components of **u** for the outflow boundary condition can be found in Appendix B, Section B.3.

4.10 Obstacles

A great feature to have in a smoke simulator is to have the smoke interact with OBJ meshes. To do this, the voxels inside the mesh and the voxels intersecting with the mesh need to be behave differently than the other voxels. The velocity in each of these solid voxels needs to be set to 0 at the end of every time step before the energy is convected.

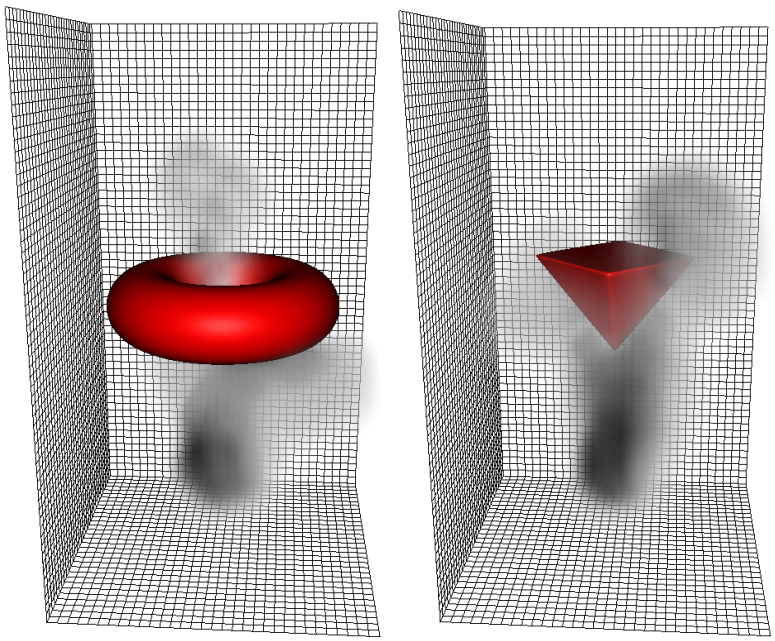

Figure 4.8: The smoke interacting with obstacles.

The difficult part is determining which voxels are inside the mesh. This can be accomplished with signed distance fields. An ideal library to use is Mathieu Sanchez's SDF library (NCCA). Due to some OS incompatibility issues, the library could not be incorporated into this simulation. This smoke simulator determines the solid voxels based on the mesh's vertices. The issue with this method is that it does not account for the voxels inside of the mesh. Also, if the faces of the OBJ are bigger than the width of each voxel, then some voxels that intersect with the mesh will be skipped and the smoke will flow into the obstacle.

4.11 Visualizing the Smoke

A vital step to a smoke simulator is the displaying of the smoke itself. The best way to show the smoke is to use point sprites. The picture used for the sprite is basically a blurry gray dot. One sprite is drawn at the center of each voxel and the sprite's opacity is determined by the voxel's density value. The size of the sprite needs to be bigger than the voxel so the blurry gray dots will blend together. The use of sprites makes it possible to alter the colour of each individual sprite at each voxel with the use of GLSL. So adding a self-shadow and displaying the temperature at each voxel is fairly easy to implement because it can be written in the fragment shader.

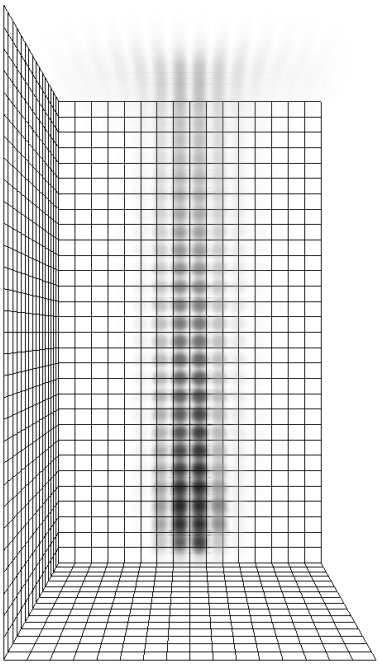

Figure 4.9: The sprites being drawn at each voxel.

4.12 Stability

If the time steps and finite space differences are small, the results will be close to continuous. Large time steps and large spatial differences (voxel widths) lead to instability problems and produce an unrealistic result (Harlow and Welch 1965). Ideally an implicit time integrator would be the best choice for a time integrator because it leads to an unconditionally stable solver. Being that it is somewhat difficult to implement, this solver uses the Runge-Kutta fourth order method (RK4) which is one of the more accurate and stable explicit time integration methods. RK4 basically samples the result at four different intervals during a time step and uses a weighted average to determine the most accurate prediction of the result.

4.13 Design

Figure 4.10 is the high-level class diagram of the smoke simulator. The design of this smoke solver uses primarily aggregation. The SmokeSystem class, (Figure C.4 in Appendix C) has all of the global variables of the smoke simulator like the kinematic viscosity constant, the dissipation constants, the pressure constants, and the buoyancy and turbulence force constants. It also has all of the functions that do the heavy duty calculations like calculating the pressure and advecting the velocity.

The GLWindow class (Figure C.5) contains the methods for drawing the different elements of the smoke simulator: the voxel grid domain, the velocity field arrows, the smoke sprites, and the

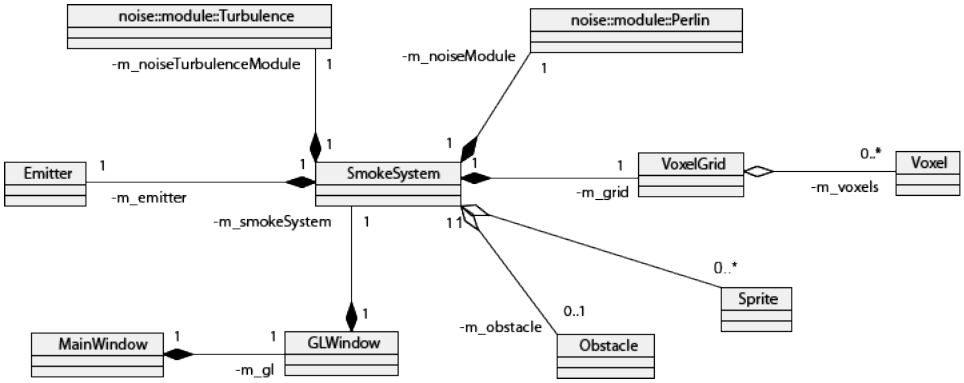

Figure 4.10: The high-level class diagram of the smoke simulator.

obstacle. It has methods for initializing the shaders and lights, moving the emitter, and updating the transform stack. The GLWindow class contains all of the public slots for receiving signals from the UI and all of the generic openGL functions like initializeGL, paintGL, and resizeGL.

The VoxelGrid class (Figure C.3 in Appendix C) contains the attributes and functions of the 3D voxel grid domain. It contains a 1D array of Voxels (Figure C.8) called m_voxels and functions to return the index of the voxel in that array based on a coordinate or three indices. When the class is constructed, the three precision values and the voxel width values are calculated based on the width of the domain in the three dimensions and the base resolution. Its m_boundaryVoxel private member is used as a special way of handling boundary voxels.

The Emitter class (Figure C.6 in Appendix C) contains the attributes and functions associated to the emitter and the Obstacle class (Figure C.2) contains the attributes and functions associated to the obstacle. The Perlin noise and Turbulence classes are part of the libnoise library by Jason Bevins (http://libnoise.sourceforge.net/) and do all of the noise calculations for the turbulence force.

4.14 Problems

An issue with this smoke solver is that it is considerably slow to simulate. This is due to the pressure relaxation method and the fact that it does not take advantage of any parallelization techniques. Also, there are no efficiency methods to minimize the amount of voxels that are iterated.

The speed of the pressure calculation is dependent on the velocity field at that time step. If the velocity field gets too varied and random, the pressure relaxation method sometimes does not reach a divergence-free field in a reasonable amount of iterations. When the max number of iterations is reached, the simulation gets unstable because of a sudden gain or loss in mass due to the fact that the velocity field is no longer divergence-free.

This leads to another issue: instability. This solver is somewhat unstable at large time steps

because of the explicit time integrator. The instability due to the time step and pressure causes the simulation to explode on occasion. It would be more stable if the time integrator was implicit and if a more reliable pressure calculation method was implemented.

Chapter 5

Conclusion

The end product of the research and implementation of this smoke solver was fairly successful. The smoke is believable and the features of the the smoke simulator can be easily controlled by an artist. It has all of the necessary forces and features of a basic smoke simulator.

There are definitely a lot of areas that could be improved especially in terms of efficiency, speed, and stability. The solver could use a better time integration technique. If the solver is more stable, the simulation will run faster because it would be possible to use bigger time steps and coarser grids. For calculating the pressure there are newer and more efficient techniques other than the Geiss-Seidel relaxation method that could help speed up the simulation.

The smoke interacting with obstacles could also be improved. Instead of determining the solid voxels based on the vertices of the OBJ mesh, it would be better to use signed distance fields to account for the voxels inside of the mesh. As an additional feature, the obstacles could have a friction force to account for different materials.

To make the simulation faster, the calculation processes can be parallelized and could take advantage of the GPU. This can be accomplished using OpenCL[1] or CUDA[2]. Caching of the smoke to render it is something that will need to be implemented in the future. There are libraries like Field3D[3] and OpenVDB[4] that can be used to export voxel data and import it into an application like Maya or Houdini.

This smoke simulator may have issues with stability and efficiency but it is successful because it is believable, artist-friendly, and it demonstrates the turbulent, buoyant, and rotational motion evident in real smoke.

[1] http://www.khronos.org/opencl/
[2] http://www.nvidia.com/object/cuda_home_new.html
[3] http://opensource.imageworks.com/?p=field3d
[4] http://www.openvdb.org/

References

Cline, D., Cardon, D., Egbert, P. K., 2004. Fluid Flow for the Rest of Us: Tutorial of the Marker and Cell Method in Computer Graphics. Provo: Brigham Young University. Available from: http://people.sc.fsu.edu/~jburkardt/pdf/fluid_flow_for_the_rest_of_us.pdf [Accessed 8 June 2013]

Deville, M. O., Fischer, P. F., and Mund, E. H., 2002. High-order Methods for Incompressible Fluid Flow. Cambridge, UK: Cambridge University Press. Available from: http://ehis.ebscohost.com/ehost/detail?sid=bb0b0a3d-dd3f-43f9-abff-e2a764ddfe30%40sessionmgr104&vid=2&bk=1&hid=102&bdata=JnNpdGU9ZWhvc3QtbGl2ZQ%3d%3d#db=nlebk&AN=112366 [Accessed 8 June 2013]

Fedkiw, R., Starn, J., and Jensen, H. W., 2001. Visual Simulation of Smoke. Stanford: Stanford University. Available from: http://graphics.ucsd.edu/~henrik/papers/smoke/smoke.pdf [Accessed 6 June 2013].

Foster, N. and Metaxas, Dimitris, 1997. Modeling the Motion of a Hot, Turbulent Gas. Philadelphia: Center for Human Modeling and Simulation University of Pennsylvania. Available from: http://graphics.cs.cmu.edu/nsp/course/15-464/Spring07/papers/fosterGas.pdf [Accessed 12 June 2013]

Griebel, M., Dornseifer, T., and Neunhoeffer, T., 1998. Numerical Simulation in Fluid Dynamics: a Practical Introduction. Philadelphia: the Society for Industrial and Applied Mathematics.

Harlow, F. H., and Welch, J. E., 1965. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. Los Alamos: Los Alamos Scientific Laboratory, University of California. Available from: http://scitation.aip.org/getpdf/servlet/GetPDFServlet?filetype=pdf&id=PFLDAS000008000012002182000001&idtype=cvips&doi=10.1063/1.1761178&prog=normal&bypassSSO=1 [Accessed 1 August 2013]

Stam, J., 1999. Stable Fluids. Alias | wavefront. Available from: http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/ns.pdf [Accessed 12 June 2013]

Stam, J., 2003. Real-time Fluid Dynamics for Games. Toronto: Alias | wavefront. Available from: http://www.dgp.toronto.edu/

people/stam/reality/Research/pdf/GDC03.pdf [Accessed 12 June 2013]

Wrenninge, M. and Zafar, N. B., 2010. Volumetric Methods in Visual Effects. In: SIGGRAPH 2010, Los Angeles, CA, USA. Sony Pictures Imageworks and DreamWorks Animation. Available from: http://magnuswrenninge.com/content/pubs/VolumetricMethodsInVisualEffects2010.pdf [Accessed 20 May 2013]

Xiu, D. and Karniadakis, G. E., 2001. A Semi-Langrangian High-Order Method for Navier-Stokes Equations. Providence: Brown University. Available from: http://www.math.purdue.edu/~dxiu/Papers/XiuK_JCP01.pdf [Accessed 6 June 2013].

Appendices

Appendix A

Derivative Approximations

A.1 First Order Derivative Approximations

$$\partial u/\partial x = \frac{u(i+1,j,k) - u(i,j,k)}{h} \tag{A.1a}$$

$$\partial v/\partial y = \frac{v(i,j+1,k) - v(i,j,k)}{h} \tag{A.1b}$$

$$\partial w/\partial z = \frac{w(i,j,k+1) - w(i,j,k)}{h} \tag{A.1c}$$

$$\partial uu/\partial x = \frac{1}{h} \cdot [u(i+1,j,k) \cdot u(i+1,j,k) - u(i,j,k) \cdot u(i,j,k)] \tag{A.2a}$$

$$\partial uv/\partial y = \frac{1}{h} \cdot [u(i+1,j,k) \cdot v(i,j+1,k) - u(i,j,k) \cdot v(i,j,k)] \tag{A.2b}$$

$$\partial uw/\partial z = \frac{1}{h} \cdot [u(i+1,j,k) \cdot w(i,j,k+1) - u(i,j,k) \cdot w(i,j,k)] \tag{A.2c}$$

$$\partial vu/\partial x = \frac{1}{h} \cdot [v(i,j+1,k) \cdot u(i+1,j,k) - v(i,j,k) \cdot u(i,j,k)] \tag{A.2d}$$

$$\partial vv/\partial y = \frac{1}{h} \cdot [v(i,j+1,k) \cdot v(i,j+1,k) - v(i,j,k) \cdot v(i,j,k)] \tag{A.2e}$$

$$\partial vw/\partial z = \frac{1}{h} \cdot [v(i,j+1,k) \cdot w(i,j,k+1) - v(i,j,k) \cdot w(i,j,k)] \tag{A.2f}$$

$$\partial wu/\partial x = \frac{1}{h} \cdot [w(i,j,k+1) \cdot u(i+1,j,k) - w(i,j,k) \cdot u(i,j,k)] \tag{A.2g}$$

$$\partial wv/\partial y = \frac{1}{h} \cdot [w(i,j,k+1) \cdot v(i,j+1,k) - w(i,j,k) \cdot v(i,j,k)] \tag{A.2h}$$

$$\partial ww/\partial z = \frac{1}{h} \cdot [w(i,j,k+1) \cdot w(i,j,k+1) - w(i,j,k) \cdot w(i,j,k)] \tag{A.2i}$$

A.2 Second Order Derivative Approximations

$$\partial^2 u/\partial x^2 = \frac{1}{h^2} \cdot [u(i+1,j,k) - 2 \cdot u(i,j,k) + u(i-1,j,k)] \tag{A.3a}$$

$$\partial^2 u/\partial y^2 = \frac{1}{h^2} \cdot [u(i,j+1,k) - 2 \cdot u(i,j,k) + u(i,j-1,k)] \tag{A.3b}$$

$$\partial^2 u/\partial z^2 = \frac{1}{h^2} \cdot [u(i,j,k+1) - 2 \cdot u(i,j,k) + u(i,j,k-1)] \tag{A.3c}$$

$$\partial^2 v/\partial x^2 = \frac{1}{h^2} \cdot [v(i+1,j,k) - 2 \cdot v(i,j,k) + v(i-1,j,k)] \tag{A.3d}$$

$$\partial^2 v/\partial y^2 = \frac{1}{h^2} \cdot [v(i,j+1,k) - 2 \cdot v(i,j,k) + v(i,j-1,k)] \tag{A.3e}$$

$$\partial^2 v/\partial z^2 = \frac{1}{h^2} \cdot [v(i,j,k+1) - 2 \cdot v(i,j,k) + v(i,j,k-1)] \tag{A.3f}$$

$$\partial^2 w/\partial x^2 = \frac{1}{h^2} \cdot [w(i+1,j,k) - 2 \cdot w(i,j,k) + w(i-1,j,k)] \tag{A.3g}$$

$$\partial^2 w/\partial y^2 = \frac{1}{h^2} \cdot [w(i,j+1,k) - 2 \cdot w(i,j,k) + w(i,j-1,k)] \tag{A.3h}$$

$$\partial^2 w/\partial z^2 = \frac{1}{h^2} \cdot [w(i,j,k+1) - 2 \cdot w(i,j,k) + w(i,j,k-1)] \tag{A.3i}$$

Appendix B

Boundary Conditions

The following equations are based off of the equations provided by Griebel et al. (1998).

B.1 No-Slip Boundary Condition

Min X Boundary:

$$u_{0,j,k} = 0 \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.1a}$$

$$v_{0,j,k} = -v_{1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.1b}$$

$$w_{0,j,k} = -w_{1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.1c}$$

Min Y Boundary:

$$u_{i,0,k} = -u_{i,1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.1d}$$

$$v_{i,0,k} = 0 \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.1e}$$

$$w_{i,0,k} = -w_{i,1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.1f}$$

Min Z Boundary:

$$u_{i,j,0} = -u_{i,j,1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.1g}$$

$$v_{i,j,0} = -v_{i,j,1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.1h}$$

$$w_{i,j,0} = 0 \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.1i}$$

Max X Boundary:

$$u_{i_{max},j,k} = 0 \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.1j}$$

$$v_{i_{max},j,k} = -v_{i_{max}-1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.1k}$$

$$w_{i_{max},j,k} = -w_{i_{max}-1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.1l}$$

Max Y Boundary:

$$u_{i,j_{max},k} = -u_{i,j_{max}-1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.1m}$$

$$v_{i,j_{max},k} = 0 \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.1n}$$

$$w_{i,j_{max},k} = -w_{i,j_{max}-1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.1o}$$

Max Z Boundary:

$$u_{i,j,k_{max}} = -u_{i,j,k_{max}-1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.1p}$$
$$v_{i,j,k_{max}} = -v_{i,j,k_{max}-1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.1q}$$
$$w_{i,j,k_{max}} = 0 \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.1r}$$

B.2 Free-Slip Boundary Condition

Min X Boundary:

$$u_{0,j,k} = 0 \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.2a}$$
$$v_{0,j,k} = v_{1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.2b}$$
$$w_{0,j,k} = w_{1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.2c}$$

Min Y Boundary:

$$u_{i,0,k} = u_{i,1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.2d}$$
$$v_{i,0,k} = 0 \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.2e}$$
$$w_{i,0,k} = w_{i,1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.2f}$$

Min Z Boundary:

$$u_{i,j,0} = u_{i,j,1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.2g}$$
$$v_{i,j,0} = v_{i,j,1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.2h}$$
$$w_{i,j,0} = 0 \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.2i}$$

Max X Boundary:

$$u_{i_{max},j,k} = 0 \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.2j}$$
$$v_{i_{max},j,k} = v_{i_{max}-1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.2k}$$
$$w_{i_{max},j,k} = w_{i_{max}-1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.2l}$$

Max Y Boundary:

$$u_{i,j_{max},k} = u_{i,j_{max}-1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.2m}$$
$$v_{i,j_{max},k} = 0 \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.2n}$$
$$w_{i,j_{max},k} = w_{i,j_{max}-1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.2o}$$

Max Z Boundary:

$$u_{i,j,k_{max}} = u_{i,j,k_{max}-1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.2p}$$
$$v_{i,j,k_{max}} = v_{i,j,k_{max}-1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.2q}$$
$$w_{i,j,k_{max}} = 0 \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.2r}$$

B.3 Outflow Boundary Condition

Min X Boundary:

$$u_{0,j,k} = u_{1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.3a}$$
$$v_{0,j,k} = v_{1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.3b}$$
$$w_{0,j,k} = w_{1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.3c}$$

Min Y Boundary:

$$u_{i,0,k} = u_{i,1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.3d}$$
$$v_{i,0,k} = v_{i,1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.3e}$$
$$w_{i,0,k} = w_{i,1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.3f}$$

Min Z Boundary:

$$u_{i,j,0} = u_{i,j,1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.3g}$$
$$v_{i,j,0} = v_{i,j,1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.3h}$$
$$w_{i,j,0} = w_{i,j,1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.3i}$$

Max X Boundary:

$$u_{i_{max},j,k} = u_{i_{max}-1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.3j}$$
$$v_{i_{max},j,k} = v_{i_{max}-1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.3k}$$
$$w_{i_{max},j,k} = w_{i_{max}-1,j,k} \qquad j = 1, ..., j_{max} \quad k = 1, ..., k_{max} \tag{B.3l}$$

Max Y Boundary:

$$u_{i,j_{max},k} = u_{i,j_{max}-1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.3m}$$
$$v_{i,j_{max},k} = v_{i,j_{max}-1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.3n}$$
$$w_{i,j_{max},k} = w_{i,j_{max}-1,k} \qquad i = 1, ..., i_{max} \quad k = 1, ..., k_{max} \tag{B.3o}$$

Max Z Boundary:

$$u_{i,j,k_{max}} = u_{i,j,k_{max}-1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.3p}$$
$$v_{i,j,k_{max}} = v_{i,j,k_{max}-1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.3q}$$
$$w_{i,j,k_{max}} = w_{i,j,k_{max}-1} \qquad i = 1, ..., i_{max} \quad j = 1, ..., j_{max} \tag{B.3r}$$

Appendix C

Class Diagrams

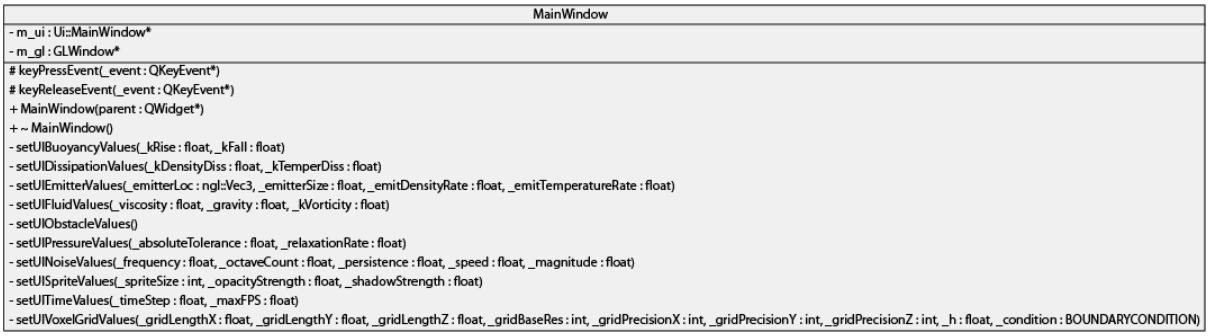

Figure C.1: The MainWindow class.

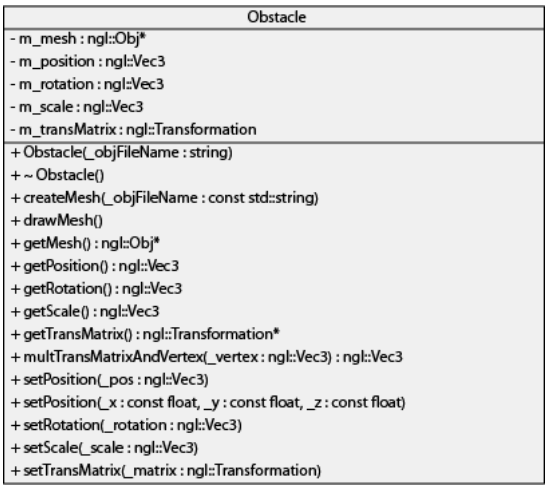

Figure C.2: The Obstacle class.

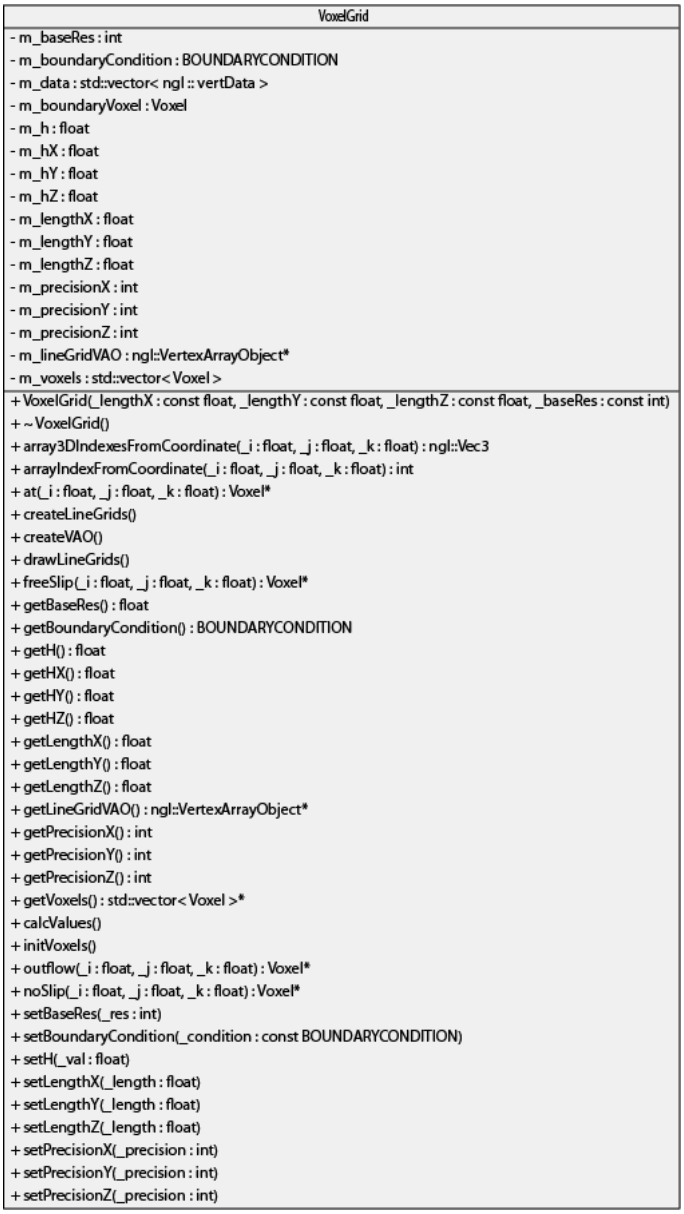

Figure C.3: The VoxelGrid class.

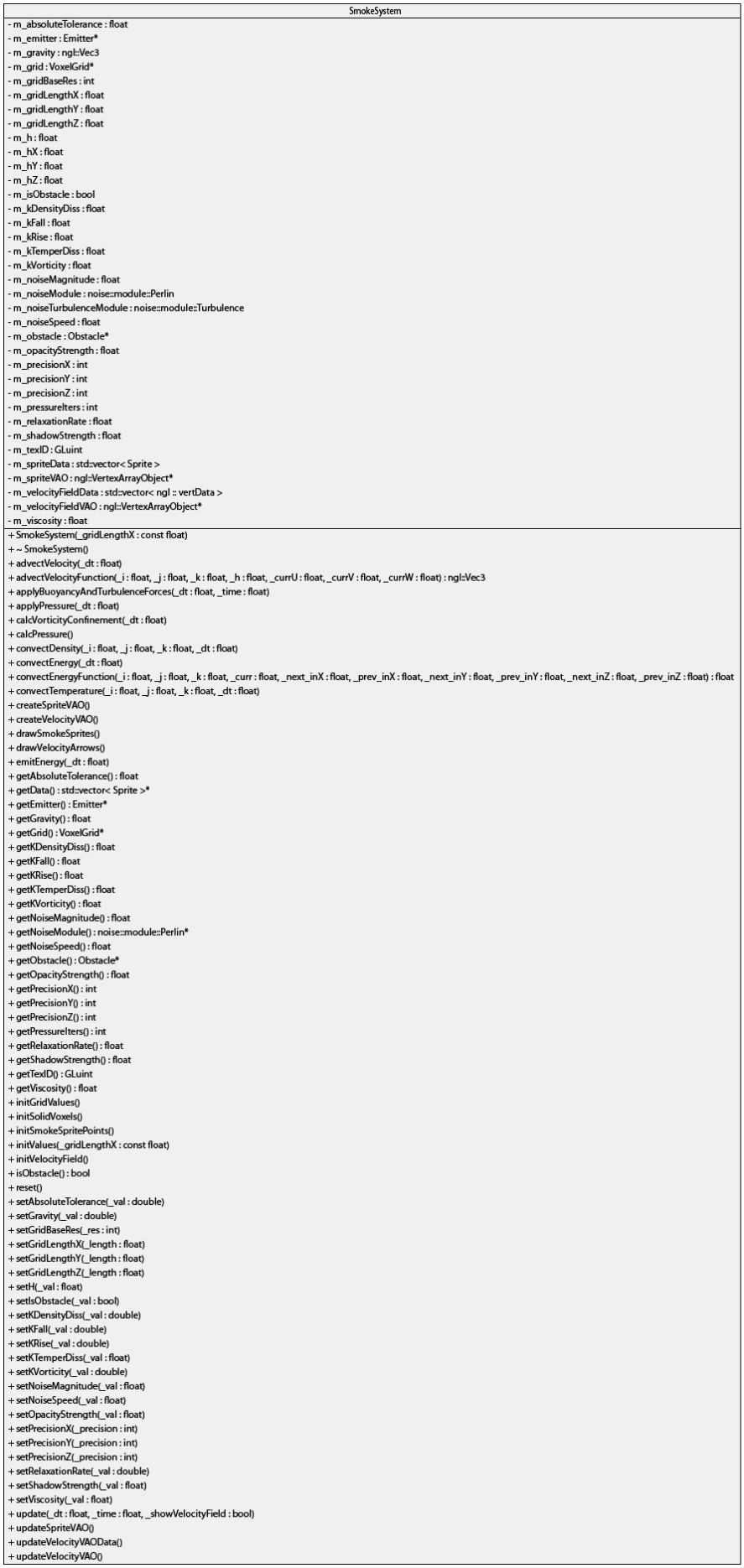

Figure C.4: The SmokeSystem class.

Figure C.5: The GLWindow class.

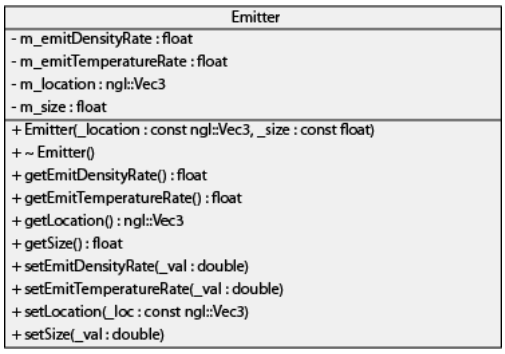

Figure C.6: The Emitter class.

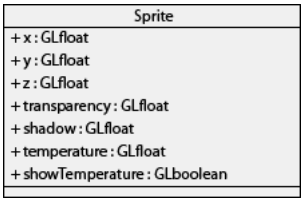

Figure C.7: The Sprite struct.

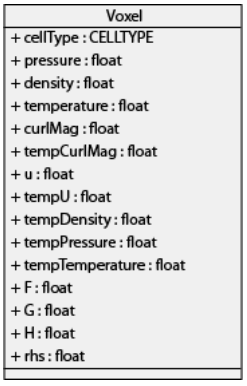

Figure C.8: The Voxel struct.