Abstract

The main goal of this project was to build a system for producing smoke animation with several control features. The system would provide a user interface using which users can control the smoke animation without having to deal with the complication that goes behind the actual fluid simulation. Users should be able to animate smoke to form shapes, follow a specified path or even follow an animation. This would give an animator the freedom of being creative with the system and would allow them to concentrate on the actual movements of the animation. The system would generate smoke animation of any shapes by keeping its natural smoke behaviour.

To build such system, the first step was to implement the basic fluid simulation with added control features. An extensive background research has been conducted on the various ways of solving the fluid equation and implementation. Simulating fluid is considered to be one of the most challenging topic in the field of computer graphics. For this project the fluid simulation needed to be fast enough to generate a real-time display of the smoke animation. The basic fluid simulation is implemented based on Stam's Simple fluid solver and the control parameters that were introduced in 'Target driven smoke animation' has been implemented on top of that. A user interface was built on top of the fluid simulation to make the system usable as an animation tool. The user interface consists of a display screen and input fields to enable user to set target shapes and control parameters for the smoke animation.

After building the system a usability evaluation (based on Monk et. al.'s Cooperative evaluation method) was conducted to identify any usability issues associated with it. Despite the fact that a number of usability issues were identified from the usability evaluation, the participants rated the functionality, quality of the animation and user interface above average. Even though the results from the evaluation shows that the system works as an effective and useful animation tool it can be further improved to take it to a professional level. Compared to the existing animation tools, the features for controlling smoke behaviour and forming shapes with smoke in this system are innovative and can be very useful for animators.

Table of Contents

1 Introduction and Context

Simulation of realistic looking and physics based animation has become very popular in recent days. With recent advancement in physics based simulation of natural phenomena it has been possible to achieve excellent visual effects. In fact every now and then we go to the cinema and get stunned by the amazing new visual effects. The effects in fluid simulation has been particularly outstanding. Examples of such animation can be seen in the motion pictures such as 'Pirates of the Caribbean: At World's end'.

To achieve this stunning animation researchers have been facing many challenges to improve the realisticness and to find a faster and effective way of achieving the results. There has often been compromises between the realisticness and speed. The realism is a very important aspect for this kind of physical simulations, but the most important issue is to give the animator the ability to control the animation without having to know the complex mathematics and computation that goes behind the actual simulation. For animators it is also very important to efficiently control the behaviour of the fluid. Since fluid simulation if very hard to predict, a high burden is placed on an animator who tries to achieve a specific flow animation. The main concern here is to constrain the fluid flow animation to satisfy the animators' need while preserving the physical behaviour of the fluid. There has been many researches in this topic of how to control fluid simulation and achieve interesting animation.

The issue of controlling particles have been examined by many researchers already. Among them Foster and Metaxas were first to introduce a method for fluid controlling. Their method used embedded controllers to control the pressure and velocity fields in [FM97]. This concept is further extended in [FF01] by sampling 3d parametric space curve. In ' Detail-Preserving Fluid Control '(2006) by N.Thurey, R. Keiser, M. Pauly, a fluid control technique is proposed that uses scale dependent force to preserve small scale fluid detail [TKPR06].

Control particles define local force and can be generated automatically from either a physical simulation or a sequence of target shapes. Small scale detail is preserved in a natural way avoiding the artificial viscosity often introduced by force based control method. This method implements the 'Smoothed particle hydrodynamics' [MCG03] to solve Navier Stokes equations. This method has proven to work quite well if we are more interested in realistic looking fluid flow.

In ' Target-driven smoke animation' by Raanan Fattal and Dani Lischinski a new method for efficiently controlling animated smoke is presented [FL04]. Given a sequence of target smoke states, this method generates a smoke simulation in which the smoke is driven towards each of these targets in turn, while exhibiting natural-looking interesting smoke-like behaviour. A driving force and a gathering term is introduced in this technique that are explicitly driven by the instantaneous state of the system at each simulation time step. This method shows good result when the control parameters are carefully controlled. In some cases the fluid simulation might look unrealistic. This method uses the traditional staggered grid system [HW65].

There are researches on various methods of key frame controlling on this type of animations like in ' Key frame control of complex particle systems using ad-joint method ' (2006)by Chris Wojtan, Peter J. Mucha and Greg Turk key frame control has been achieved on particle systems using the ad-joint method [WMT06]. Hong and Kim introduced a technique for deriving potential field form a initial state and target shape in [HK04].

1.1 Aims and Objectives:

The aim of the project is to build a system with a user interface using which a user can make interesting fluid animations without having to deal with the complexity that goes behind the simulation of fluid. User can make the smoke to form any kind of shape or follow an animation and be creative with it. User can make desired effect by adjusting some control parameters, which will have effect on how the smoke is behaving to achieve target shape. The goal is to build the system so that user can have maximum control over the animation with minimum involvement to background knowledge of fluid simulation and can be creative with it.

The objectives to meet this aim can be broken down into the following points:

1. Study the basic theory and equations behind fluid simulation and study the methods that can be applied to control basic fluid flow.

2. Implement the algorithms to create a basic fluid simulation to illustrate the concept.

3. Implement the fluid control methods over the basic simulation to form some target shapes . Observe the effect of the control parameters.

4. Build a user interface around the system through which user can control the parameters and set target shapers.

5. Test the system with different parameters and generate outcomes that reflects the control users can get over the animation.

6. Perform some user testing on the system for evaluation.

1.2 Organization of the dissertation:

This dissertation organized in five main parts.

Firstly, the next chapter introduces the background of this study and gives the overview of the basics of fluid simulation. The equations and the ways to solve the equations involved in producing fluid simulation has been mentioned in detail. Some of the previous works that has been done to control fluid has also been presented in this chapter.

Secondly, chapter three presents that projects in details with the technical and scientific work done to complete the project. It consists of implementing the fluid equations with the control features. All the work that has been done to present the system as an animation tool has been presented here with technical details.

Thirdly, the results produced by experimenting with the project has been presented with images in chapter four.

The fifth chapter consists of the evaluation of the system. The user testing performed to evaluate the user interface of the system and the animation produced has been explained in this chapter. It gives an overview of how effectively the whole system performs as an animation tool.

Finally, in the last chapter, some future work that can be done to extend the system has been mentioned. This might make the system much more effective and interesting.

2 Background and previous work

2.1 Introduction:

To get the background knowledge on this topic, at first I needed to get a good understanding of the basic concepts and equations for fluid simulation. I did a lot of research on the physics behind fluid behaviour, then I needed to understand how the Navier Stokes equation stands on it. Having a good understanding on the Navier Stokes equation, I looked for efficient and effective numerical methods to solve the equations. Among the many techniques exists, I found that many researchers chose to use the "MAC Grid" or staggered grid system [HW65] to solve the equations. Even though this process is quite old but it is still among the most efficient and effective methods to solve the equations in Eulerian view point.

I have come across another method called Smoothed Particle Hydrodynamics (SPH) to solve Navier Stokes [MCG03] . This solves the equations in Lagrangian viewpoint. This process considers every single particles of the object and can become highly time consuming and heavy to compute. However, in certain specification where very small amount of fluid is required, like a splash, or a spill, it performs very well and preserves fine details.

In recent research papers I have come across many ways of controlling fluid simulations. I have mentioned some additional terms introduced with the basic Navier Stokes and how they influence the fluid flow. Some of these terms has direct control over the speed and sharpness which is very useful from my project perspective. With a specified set of target shapes it can control the fluid to form shapes and there are parameters to control the speed and sharpness. In some cases there are trade-offs between achieving a more natural looking fluid and how closely the target shape is matched.

2.2 Basics of fluid simulation:

The physics behind realistic fluid simulation can be generally described by the Naviar Stokes equation. If we consider fluid to have no viscosity then Euler equation can describe physical fluid behaviours in the same way. The basic concept behind Naviar Stokes equation is derived from two natural behaviour of fluid:

- Momentum conservation
- Incompressibility

2.2.1 Momentum conservation

Momentum conservation is actually another form of the very basic Newton's law of force $F=ma$, where F is the force acting on some object of m mass resulting a acceleration. In case of fluids, the momentum conservation in Navier Stokes describes how the particles in fluid accelerated when a force is acting on them.

We can consider fluid as a particle system and each particle can be represented as a blob of m mass, having velocity *u*, then in *t* time acceleration can be represented by,

$$a = \frac{Du}{Dt} \qquad (2.1)$$

following Newtons law we can write,

$$m\frac{Du}{Dt} = F \qquad (2.2)$$

Here *F* is the force acting on the fluid that is causing the acceleration. There can be many sources of this force. Difference of pressure can be one source as the region with high pressure will always be pushing the region of low pressure. This can be derived by integrating the negative gradient of pressure ∇P over the volume *V*, which is

$$-\nabla P * V.$$

Another source of force can be due to the fluid viscosity. This can be considered as the force that makes the particle velocity to be an average of its surrounding particles. The Laplatican $\nabla \cdot \nabla$ operator on the velocity of the particle can define its difference from the surrounding average. Integrating this term over the volume V and taking into account the viscosity coefficient, this force is derived as,

$$V * \mu * \nabla . \nabla u$$

Gravity can easily be considered as another force, which basically acts on everything on earth. This gravitational force can be derived as *mg* (mass X gravity).

Sum of all these forces has to be equal to the acceleration force on fluid to conserve momentum according to Newton's second law. Fluid has to follow the following equation to conserve momentum,

$$m\frac{Du}{Dt} = -\nabla P * V + V * \mu * \nabla . \nabla u + mg \qquad (2.3)$$

When we will consider infinite number of particles, this equation will cause problem because volume of each blob will then become zero. To avoid this situation, we need to remove the *V* from the equation. We can say, *m/V* as *ρ*, the density of fluid. By dividing both side by volume *V* we get,

$$\rho\frac{Du}{Dt} = -\nabla P + \mu * \nabla . \nabla u + \rho g$$

Again dividing both side by *ρ* and rearranging we get,

$$\frac{Du}{Dt} + \frac{1}{\rho}\nabla P = \frac{\mu}{\rho} * \nabla . \nabla u + g \qquad (2.4)$$

In this equation the derivative term *Du/Dt* can be broken into two simplified terms if we apply Eulerian view point of tracing motion.

Eulerian viewpoints on tracing motion makes tracing motion on fluid simulation simpler. It considers a fixed point in space and observes any kind of change in that fixed point as fluid flows by that fixed point. One kind of change can happen at that fixed point for this passing of fluid, at the same time there will be another kind of change happening as the flowing fluid can have different values for those attributes in different position. So as it flows, it contributes to that change of the fixed point. From this concept, the derivative term *Du/Dt* can be broken into two terms, *δu/δt*, which is the change at a fixed position and ∇ *u.dx/dt*, which is the change happening due to position changing.

We can write,

$$\frac{Du}{Dt}=\frac{\delta u}{\delta t}+\nabla u.\frac{dx}{dt} \text{ or, } \frac{\delta u}{\delta t}+\nabla u.u$$

So the previous equation for momentum conservation becomes,

$$\frac{\delta u}{\delta t}=-u.\nabla u-\frac{1}{\rho}\nabla p+\frac{\mu}{\rho}*\nabla.\nabla u+g \qquad (2.5)$$

This is the momentum equation of Naviar Stokes.

2.2.2 Incompressibility

Compressibility is the fluids ability of changing volume. In nature it is almost impossible for fluid to change its volume. Even if is it possible to change the volume applying special equipments, this is not very relevant in terms of fluid animation. Even if we want to consider this negligible amount of volume change, this will make the computation too complex and it is not even required in most of the cases. As we are concentrating on simulating fluid animation, we can consider fluid as incompressible.

If we consider a portion of fluid with volume *V*, and the boundary surface as *δv*, then we can get the change of volume by integrating the normal of its velocity around the boundary surface,

$$\frac{dV}{dt}=\iint_{ab}\vec{u}.\hat{n} \qquad (2.6)$$

For incompressible fluid change of volume would be zero. So we can write,

$$\iint \vec{u}.\hat{n}=0 \text{ , from this, } \iiint \nabla u=0$$

This can only be true if ,

$$\nabla.u=0$$

This is the incompressibility condition for Navier Stokes equation.

This is in fact the main constraint of fluid simulation. At any point of fluid simulation the divergence of velocity ∇u needs to be zero. That is exactly what happens because of the pressure. The pressure needs to be in an amount so that it keeps the divergence of velocity to zero at any point.

We can simplify the Navier Stokes equation a bit more by considering the viscosity term to be zero. Viscosity of a fluid actually means how resistive the fluid is to moving. In here I am not thinking of dealing with fluids that are highly resistive. I am thinking of plain water kind of fluids, where viscosity can be considered as negligible. This is to simplify the Naviar Stokes equation in order to work with it. In fact when viscosity is considered as zero Naviar Stokes equation becomes Euler equation.

So considering viscosity term $\frac{\mu}{\rho} * \nabla . \nabla u$ in Naviar Stokes equation as zero we get a much simpler form Euler equation , which is,

$$\frac{\delta u}{\delta t} = -u . \nabla u - \frac{1}{\rho} \nabla p + g \tag{2.7}$$

$$\nabla . \vec{u} = 0 \tag{2.8}$$

These are the basic equations for fluid simulation. At every point it needs to satisfy these two equation to get a realistic looking fluid simulation which is incompressible and with no viscosity. I will be using this equation to simulate fluid for my project.

2.3 Numerical Simulation

There are many approaches to numerically simulate the fluid equation to make use in graphics. Approach for 'splitting' method known as 'operator splitting [TVF92] is used most of the time as it works well enough for graphics purpose. The concept is like, when there is a sum of many terms in an equation that makes the value of some derivation, we can split the terms and make separate simpler equations to solve it. The separated terms can be simpler to solve, then the values of those terms can be summed up again to get the actual value.

For example for the equation,

$dv/dt = f(v) + g(v)$

we can write separate equation by splitting them,

$v_{temp} = v_n + \Delta t\, f(v)$
$v_{n+1} = v_{temp} + g(vt_{emp})$

or this can be written simply as,

$v_{temp} = V(v_n , \Delta t)$
$v_{n+1} = G(v_{temp}, \Delta t)$

Even if *dv/dt* was complicated to solve, we can have simple methods to solve *V( )* and *G( )*. Following this concept we can split our equation for fluid to get simpler equation that we have well defined method to solve. Then we can add up the solutions we get to solve the main equation. We can have one equation for the advection part, one for the body force and one for the incompressibility part.

$u_a = advect\ (\Delta t, u_n)$
$u_b = u_a + \Delta tg$
$u_{n+1} = project\ (\Delta t, u_b)$

project($\Delta t, u_b$*)* is a method that calculates pressure in such a way that it will always return a velocity *u* which is divergence free. This will satisfy the incompressibility property. If we try to get the value from *advect()*, which returns the advection velocity, using a velocity which is not divergence free then this will cause a problem. For this reason we will need to first calculate the velocity form the *project( )* method, which is always divergence free, then use it in *advect( )* method to get the right velocity. Following this, our basic algorithm for fluid can be stated like the following, Copied from [BF07]

- Find a divergence free velocity to use as an initial velocity *u* 0
- For time step n = 0, 1, 2, 3, 4........
    - Find a suitable time step Δt, which is the time space from t_n to t_{n+1}
    - Set $u_a = advect\ (\Delta t, u_n)$
    - Add $u_b = ua + \Delta tg$
    - Set $u_{n+1} = project\ (\Delta t, u_b)$

While deciding on the time step we need to keep in mind that it should not go pass the duration of current time frame. If we take Δt such that $\Delta t + t_n > t_{frame}$, then we need to set a flag as an alert that we have reached the end of the frame and then set Δt to $t_{frame} - t_n$.

2.3.1 Computational Grid

The 'staggered' grid algorithm introduced by Harlow and Welch in [HW65], is used to solve the fluid equation numerically. In this approach the variables are placed in specific locations in the grid. If we consider a 2D (two dimensional) grid then the pressure p_{ij} will be placed at the centre of the ij cell. The vertical component of the velocity $v_{i,j+1/2}$ is place at the centre of the face between cell *ij* and cell *i, j+1* and the component $v_{i,j-1/2}$ is place at the centre of the face between cell *i, j-1* and cell *i,j*. Similarly for the horizontal component $u_{i+1/2,j}$ will be place at the centre of the face between the cell *i,j* and the cell *i+1, j* and the component $u_{i-1/2,j}$ will be place at the centre of the face between the cell *i,j* and the cell *i-1,j*. For 3D, the placement will be the same, just with another extra component. The pressure will always stay at the centre.

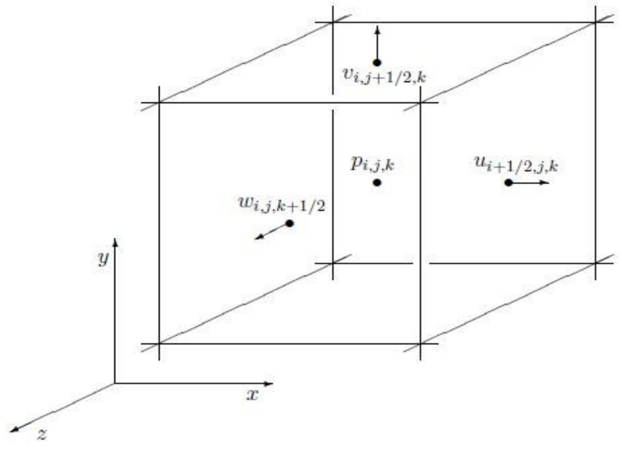


Fig 3.1: Computational grid, staggered grid algorithm. Copied from [HW65]

The reason for using this method is to get the accurate and unbiased value when we calculate the different for pressure and velocity. This method has advantages when we want to calculate pressure but when it comes to calculate velocity we will need to use interpolation of all the components of a velocity. Say for a three dimensional velocity we will need to deal with three components. For the horizontal component of velocity *u i,j* we will need to take the average of two values *u i+1,j ,k* and *u i-1,j ,k* and same for the vertical component. For the vertical component of *u i+1/2, j, k* we will need to average four velocities. The velocities for three dimensional grid will be like the following,

$$u_{\vec{(i,j)}} = \frac{(u_{(i-1/2,j,k)} + u_{(i+1/2,j,k)})}{2}, \frac{(v_{(i,j-1/2,k)} + v_{(i,j+1/2,k)})}{2}, \frac{(w_{(i,j,k-1/2)} + w_{(i,j,k+1/2)})}{2} \qquad (3.1)$$

$$u_{\vec{(i+1/2,j,k)}} = u_{(i+1/2,j,k)}, \frac{(v_{(i,j-1/2,k)} + v_{(i,j+1/2,k)} + v_{(i+1,j-1/2,k)} + v_{(i+1,j+1/2,k)})}{4}, \frac{(w_{(i,j,k-1/2)} + w_{(i,j,k+1/2)} + w_{(i+1,j,k-1/2)} + w_{(i+1,j,k+1/2)})}{4}$$

$$u_{\vec{(i,j+1/2,k)}} = \frac{(u_{(i-1/2,j,k)} + u_{(i+1/2,j,k)} + u_{(i-1/2,j+1,k)} + u_{(i+1/2,j+1,k)})}{4}, v_{(i,j+1/2,k)}, \frac{(w_{(i,j,k-1/2)} + w_{(i,j,k+1/2)} + w_{(i,j+1,k-1/2)} + w_{(i,j+1,k+1/2)})}{4} \qquad (3.2)$$

$$u_{\vec{(i,j,k+1/2)}} = \frac{(u_{(i-1/2,j,k)} + u_{(i+1/2,j,k)} + u_{(i-1/2,j,k+1)} + u_{(i+1/2,j,k+1)})}{4}, \frac{(v_{(i,j-1/2,k)} + v_{(i,j+1/2,k)} + v_{(i,j-1/2,k+1)} + v_{(i,j+1/2,k+1)})}{4}, w_{(i,j,k+1/2)} \qquad (3.3)$$

2.4 Advection algorithm

In the basic algorithm of fluid simulation it is an important step to solve the advection equation. The advection method is denoted as,

$q^{n+1} = advect\ (u, \Delta t, q^n)$

In this case instead of using forward Euler method we will use a method called "semi-Lagrangian" method [Sta99] which is much simpler. The forward Euler method was simple to use but in this case there is a risk of appearing null-space. Even if a very small value of *Δt* is taken this might happen which will cause error in the calculation as some values will appear to be zero. Using semi-Lagrangian method solves this problem.

The concept of semi-Lagrangian method is, when we are looking for a new value at a certain position in the grid, the new value would be the old value of some particle that has ended at that position. It is very simple if we think that all we need to do is to find that old value of a particle that ended at that position in the grid. If the position in the grid we are looking at is X_G and we want to find the new value which is q_G^{n+1} and a particle with old value q_p^n end up at X_G position in the grid in time step Δt, then according to semi-Lagrangian method, the new value, $q_G^{n+1} = q_p^n$

Now we need to find the value for q_p^n.

The particle started at position X_P in the grid and after Δt time interval it reaches at position X_G.
By reversing the particle velocity u to $-u_G$ we can get the starting position of the particle form the following equation,

$X_P = X_G - \Delta t u_G$

It can happen that X_P is not on the grid. In this situation we can use interpolation which will give us an approximate position on the grid. So the semi-Lagrangian formula can be stated as,

$q_G^{n+1} = interpolate\ (q^n, X_G - \Delta t u_G)$

We will need to apply this advection process on velocity, density, and many other properties. Density is always at the centre of a cell in the staggered grid. We can use the value straight away. In case of velocity we will need to find averages and use those values to use in the advection process.

Because of the interpolation we are doing in every advection step, we are taking some average values in place of the original. It is very obvious that we will lose some sharpness, because of some numerical dissipation. It is likely that we will be losing details form our simulation. In some stages it might seem like we are simulating fluid with viscosity even though from the very beginning we started with fluid without viscosity. If we were already dealing with viscosity then this might be a very small problem but fluid with no viscosity might show noticeable dissipation.

2.5 Projection algorithm

The implementation of projection algorithm, *project (Δt, u_b)* is about considering the incompressibility of fluid. It is mentioned earlier that making fluid in compressible means making the velocity divergence free. We will need to find the pressure at a position, then subtract the gradient of pressure from the previous velocity to get the new velocity which needs to satisfy the condition of being divergence free. The boundary condition needs to be considered while computing the velocity.

So the equations we get from this are,

$$u^{(\vec{n+1})} = \vec{u} - \nabla t \frac{1}{\rho} \nabla p \qquad (5.1)$$

For divergence free velocity,

$$\nabla . u^{(\vec{n+1})} = 0 \qquad (5.2)$$

For boundary condition, We are assuming that the solid boundary is exactly in line with the staggered grid cell walls. We can say that no fluid will be flowing through a solid boundary. It can just slip by the solid wall it is facing. This implies that the normal component of the fluid velocity has to be zero. The normal is the normal of the solid face. It is for the case when the solid boundary is not moving. If we consider the solid boundary is moving with velocity u_{solid} and the normal of the solid surface is *n*, then we can say that to satisfy boundary condition, the dot product of the velocity of fluid and the velocity of solid has to be equal. So the equation is,

$$\vec{u}^{(n+1)} \cdot \hat{n} = \vec{u}_{solid} \cdot \hat{n}$$

From the staggered grid system described earlier, we can see that the ∇p to subtract form the velocity component can be obtained by applying the central difference approximation. The equations for the velocity components in a 3D staggered grid are like the followings,

$$u^{n+1}_{i+1/2,j,k} = u_{i+1/2,j,k} - \Delta t \frac{1}{\rho} \frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x} \quad (5.4)$$

$$v^{n+1}_{i,j+1/2,k} = v_{i,j+1/2,k} - \Delta t \frac{1}{\rho} \frac{p_{i,j+1,k} - p_{i,j,k}}{\Delta x} \quad (5.5)$$

$$w^{n+1}_{i,j,k+1/2} = w_{i,j,k+1/2} - \Delta t \frac{1}{\rho} \frac{p_{i,j,k+1} - p_{i,j,k}}{\Delta x} \quad (5.6)$$

If we think of the staggered grid as cells of fluid we can see that the velocity in a fluid cell face is actually derived from the pressure fluid of a cell which is outside of the that cell. That means if we are deriving the velocity of at the fluid surface which has air at its surrounding we can just put zero for the pressure fluid. If it is at solid wall boundary then the out side cell velocity is nothing but the velocity of the solid wall, which is u_{solid}. So by putting the u_{solid} in place of $u^{n+1}_{i+1/2,j,k}$ and rearranging the previous equation we can get the pressure fluid of solid boundary,

$$p_{i+1,j} = p_{i,j} + \frac{\rho \Delta x}{\Delta t}(u_{i+1/2,j} - u_{solid}) \quad (5.7)$$

If there are more than one solid faces surrounding the fluid we will need to calculate pressure component for each of the solid faces.

2.5.1 Discrete Divergence

Here we are going to get the divergence of velocity for the fluid cells only. The divergence of velocity can be written as the following,

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

Again by applying the central difference from the staggered grid we can get an approximation of the divergence of velocity.

$$\nabla \cdot \vec{u}_{i,j,k} \approx \frac{u_{i+1/2,j,k} - u_{i-1/2,j,k}}{\Delta x} + \frac{v_{i,j+1/2,k} - v_{i,j-1/2,k}}{\Delta x} + \frac{w_{i,j,k+1/2} - w_{i,j,k-1/2}}{\Delta x}$$

Now we as our divergence of velocity needs to be zero, our pressure equation in 3D,

$$\frac{u^{n+1}_{i+1/2,j,k} - u^{n+1}_{i-1/2,j,k}}{\Delta x} + \frac{v^{n+1}_{i,j+1/2,k} - v^{n+1}_{i,j-1/2,k}}{\Delta x} + \frac{w^{n+1}_{i,j,k+1/2} - w^{n+1}_{i,j,k-1/2}}{\Delta x} = 0$$

Our main goal for the projection algorithm is to find the pressure that is going to make the velocity divergence free. In the above equation we can simply replace the velocity fields by the once from the previous pressure equations. Making some rearrangements, if we bring the pressure components one side and the velocities to the other side, we get the following equation

$$\frac{\Delta t}{\rho}\left(\frac{6\mathrm{p}_{i,j,k} - p_{i+1,j,k} - p_{i,j+1,k} - p_{i,j,k+1} - p_{i-1,j,k} - p_{i,j-1,k} - p_{i,j,k-1}}{\Delta x^2}\right) = ¿$$
$$¿ - \left(\frac{u_{i+1/2,j,k} - u_{i-1/2,j,k}}{\Delta x} + \frac{vi,j+1/2,k - vi,j-1/2,k}{\Delta x} + \frac{w_{i,j,k+1/2} - w_{i,j,k-1/2}}{\Delta x}\right) \quad (5.8)$$

Now this is our basic equation. We now need to consider where the fluid grid cell is, because it is mentioned before that the velocity we are trying to derive depends on the pressure fields of the outside cell. For the cell *(i, j+1)*, if it is in the air, we can simply put the value for $p_{i,j+1}$ to zero or omit the pressure term for the cell. It can be that the cell *(i, j+1)* is actually in solid boundary. Then for this pressure field we have the formula in equation (5.7) . The effect of applying this on the equation above is, for the air surrounding, the pressure term will be omitted. For solid surrounding the pressure term for air will also disappear. The coefficient for $p_{i,j,k}$ will decrease by one, it is actually the number of fluid cell surrounding it.

2.5.2 Forming the Matrix Vector form

We can now form a matrix vector form the the above pressure equation.

AP = D

Where *A* is the coefficients of the unknown pressure, *P* is a matrix of all the unknown pressure in the left hand side and *D* is a matrix with the divergences on the right hand side in each of the fluid cells in the staggered grid. Each row in A correspond to one equation, so we can address the rows as $A_{i,j,k}$ and the entries in the rows will be the unknown pressure coefficients. As most of them will be zero accept for $p_{i,j,k}$ and its six neighbours. If $p_{i,j,k}$ have n neighbours then the coefficients will be *-n Δt/ρ*, so if we divide both side by *Δt/ρ* we will get the matrix with off-diagonal entries of 0 or -1 and diagonal values of *n*. This becomes a symmetrical matrix. So instead of having the *A* as an actual matrix we can have four values for each cell in the fluid grid. For position *(i, j, k)* the values will be *Adiag(i, j, k), Aplusi(i, j, k), Aplusj(i, j, k), Aplusk(i, j, k)*. we can use the symmetry to refer to the value we need.

2.5.4 Solving the Matrix

There are many ways to solve this type of linear equation using matrix. But for this I will be using the "Modified Incomplete Cholesky Conjugate Gradient, Level zero" or in short MICCG(0) algorithm which works quite well and mostly used in this case as it is fast and quite efficient [BF07].

Conjugate gradient (CG) method is the way of solving equation iteratively. We start with guessing a solution and then test it how far it is form solving the equation. With each iteration the solution gets closer to the actual solution and when we think it has reached a well enough solution we end the iteration.

There are some issues that we need to think of when we use any CG algorithm. We will need a way of knowing how close or how far we are from the actual solution and what should the initial guess for pressure be when we start the iteration. That seems to be a bit tricky since we do not know the actual solution. For each iteration we are going to look in a residual instead of the solution.

$$Ri = D - APi$$

Pi is the pressure at ith iteration. So obviously when *Ri* is equal to zero , we can say that we have reached the solution. What we look for is actually an Ri which is small enough. The smaller we fix our satisfactory *Ri* to be, the accurate our solution will be. We will fix a small value for tol (tolerance), which we will compared with the residual *Ri* . We will compare the maximum absolute value for every iteration step. In fact at every iteration step we will see how much divergence we are getting for the *Pi*. The smaller the divergence the closer we are to the solution, as our solution should get us a divergence free velocity. In fact the smaller we take our tolerance *tol*, the more accurately incompressible our solution will be. But at the same time it will take longer time. We will need to fix the length of our iteration as we do not want it to take much time. It can increase our computational time significantly. This fixed length is a compromise on quality that we need to make to get faster result.

For our initial guess the last solution we got can be a good place to start with if the pressure in the fluid has not changed much. But usually it is unlikely for the pressure to stay unchanged. Pressure will keep changing significantly so it is safe to start with an all zero vector for initial guess.

In this case of solving for the fluid grid, it is possible that it might take a very long time as the larger the grid will get the more computation will be needed to be done. In fact it will take time proportional to the size of the grid. So we will need a way to solve this problem.

Preconditioned Conjugate Gradient

We will perform some preconditioning on the equation to make it reach to a solution quicker. For the equation $AP = D$, it is noticed that if A is close to being identity it is possible to reach solution quicker. So the purpose of preconditioning will be to multiply A with something that will make it close to identity. Solving $MAP = MD$ (multiplying both side of AP=D by M) is the same as solving $AP = D$. So we need to find M such that MA is close to identity. Which means we will need to find a way to precondition A , then apply PCG, Preconditioned Conjugate Gradient algorithm on the equation to solve it.

The algorithm for PCG is stated below. A method for preconditioner , applyPreconditioner ( ) , and a method for multiplying coefficient vector A, applyA( )is used here which will be descried next.

The Preconditioned Conjugate Gradient (PCG) algorithm for solving AP = D : Copied from [BF07]

```
• Set initial guess p = 0 and residual vector r = d (If r = 0 then return p)
• Set auxiliary vector z = applyPreconditioner(r), and search vector s = z
• $\sigma$ = dotproduct(z, r)
• Loop until done (or maximum iterations exceeded):
    • Set auxiliary vector z = applyA(s)
    • $\alpha = \rho$ /dotproduct(z, s)
    • Update p+ = $\alpha$s and r− = $\alpha$z
    • If max $|r| \leq tol$ then return p
    • Set auxiliary vector z = applyPreconditioner(r)
    • $\sigma_{new}$ = dotproduct(z, r)
    • $\beta = \sigma_{new} / \rho$
    • Set search vector s = z + $\beta$s
    • $\sigma = \sigma_{new}$
• Return p (and report iteration limit exceeded)
```

Defining Preconditioner

The main goal of finding the right preconditioner is to make the convergence faster. In this point an inverse matrix for A would be just perfect as the multiplication will give us an identity vector but it is very expensive to compute. There has been many methods for computing a effective and efficient preconditioner. For this project I will be using the Incomplete Cholesky method , which is quite fast and easier to implement.

In Cholesky's method We start row reduction on until we get Upper triangle L Then a is factored as its lower end upper triangular matrix. A = L L-1 . Then solving *AP = D* is same as solving $LL^{-1}P = D$. But the problem here is even though it has a very few non zero element , L might have a lot of non zero to store. This might get very heavy to work with. So to solve this situation, we take the approach of Incomplete Cholesky. The approach is to when we are calculating L, and we tend to put non zero where in place of a zero, we just keep it zero , do not change it. The result of this is to get much lighter L but the consequence is A is not equal to LL^{-1} any more. We get some error here. Even though we do not get the actual factor of A here, it is close enough to solve the equation. This process is called IC(0).

If we split *A* into Lower triangle *F* and Diagonal *D*, ordering our grid in *i, j and k dimension* then we can write,

$A = F + D + F^T$

Then the Incomplete Cholesky's *L* can be written s,

$L = LE^{-1} + E$, here *E* is a diagonal matrix.

Applying some algebra the computation of E can be written as,

$$E_{i,j,k} = \sqrt{A_{(i,j,k),(i,j,k)} - \left(\frac{A_{(i-1,j,k),(i,j,k)}}{E_{(i-1,j,k)}}\right)^2 - \left(\frac{A_{(i,j-1,k),(i,j,k)}}{E_{(i,j-1,k)}}\right)^2 - \left(\frac{A_{(i,j,k-1),(i,j,k)}}{E_{(i,j,k-1)}}\right)^2}$$

Modified Incomplete Cholesky, MIC(0) the one that we will use which is the same s IC(0) but with simple modification. While we re computing L keeping the same non zero pattern we do two extra steps,

1. We keep the off diagonal entries same in A and LL-1
2. We keep the sum of each row in A the same as the sum of each row in LL-1

By doing this there is only a same change in the equation of deriving E, [TKPR06]

….

$$\begin{aligned} E_{i,j,k} = \sqrt{.}\, A_{(i,j,j)(i,j,k)} &- \left(\frac{A_{(i-1,j,k)(i,j,k)}}{E_{i-1,j,k}}\right)^2 - \left(\frac{A_{(i,j-1,k)(i,j,k)}}{E_{(i,j-1,k)}}\right)^2 - \left(\frac{A_{(i,j,k-1)(i,j,k)}}{E_{(i,j,k-1)}}\right)^2 \\ &- A_{(i-1,j,k)(i,j,k)} \frac{\left(A_{(i-1,j,k)(i-1,j+1,k)} + A_{(i-1,j,k)(i-1,j,k+1)}\right)}{E^2_{(i-1,j,k)}} \\ &- A_{(i,j-1,k)(i,j,k)} \frac{\left(A_{(i,j-1,k)(i+1,j-1,k)} + A_{(i,j-1,k)(i,j-1,k+1)}\right)}{E^2_{(i,j-1,k)}} \\ &- A_{(i,j,k-1)(i,j,k)} \frac{\left(A_{(i,j,k-1)(i+1,j,k-1)} + A_{(i,j,k-1)(i,j+1,k-1)}\right)}{E^2_{(i,j,k-1)}} \end{aligned} \quad (5.9)$$

The effect of applying MIC(0) would be much faster convergence without adding any computational complication.

The calculation of the MIC(0) preconditioner in three dimensions: Copied from [TKPR06]

• Set tuning constant = 0.97
• For i=1 to nx, j=1 to ny, k=1 to nz:
• If cell (i, j, k) is fluid:
• Set

$$\begin{aligned} e = Adiag_{i,j,k} &- (Aplusi_{i-1,j,k} * precon_{i-1,j,k})^2 \\ &- (Aplusj_{i,j-1,k} * precon_{i,j-1,k})^2 - (Aplusk_{i,j,k-1} * precon_{i,j,k-1})^2 \\ &- Aplusi_{i-1,j,k} * (Aplusj_{i-1,j,k} + Aplusk_{i-1,j,k}) * precon^2_{i-1,j,k} \\ &+ Aplusj_{i,j-1,k} * (Aplusi_{i,j-1,k} + Aplusk_{i,j-1,k}) * precon^2_{i,j-1,k} \\ &+ Aplusk_{i,j,k-1} * (Aplusi_{i,j,k-1} + Aplusj_{i,j,k-1}) * precon^2_{i,j,k-1} \end{aligned}$$

$precon_{i,j,k} = \dfrac{1}{\sqrt{e + 10^{-30}}}$ (small number to guard against accidental divide-by-zero)

Applying the MIC(0) preconditioner in three dimensions [TKPR06]

• (First solve Lq = r)
• For i=1 to nx, j=1 to ny, k=1 to nz:
• If cell (i, j, k) is fluid:
• Set

$$t = r_{i,j,k} - Aplusi_{i-1,j,k} * precon_{i-1,j,k} * q_{i-1,j,k} - Aplusj_{i,j-1,k} * precon_{i,j-1,k} * q_{i,j-1,k} - Aplusk_{i,j,k-1} * precon_{i,j,k-1} * q_{i,j,k-1}$$

• $q_{i,j,k}$ = t * preconi,j,k
• (Next solve LT z = q)
• For i=nx down to 1, j=ny down to 1, k=nz down to 1:
• If cell (i, j, k) is fluid:
• Set

$$t = q_{i,j,k} - Aplusi_{i,j,k} * precon_{i,j,k} * z_{i+1,j,k} - Aplusj_{i,j,k} * precon_{i,j,k} * z_{i,j+1,k} - Aplusk_{i,j,k} * precon_{i,j,k} * z_{i,j,k+1}$$

• $z_{i,j,k}$ = t * $precon_{i,j,k}$

2.5.5 Projection algorithm

Now as we have all the subroutines that we need to do out projection step in our actual fluid simulation. When we have the matrix for of our pressure equation and set the A matrix properly, We will perform MIC(0) preconditioning. Then all we need to do is to converge to the solution using PCG, and we get the new velocity by using the pressure gradient from the solution of PCG and updating *u*.

The algorithm is stated below Copied from [BF07]

• Calculate the divergence d (the right-hand side) with modifications at solid wall boundaries.
• Set the entries of A (stored in Adiag etc.).
• Construct the MIC(0) preconditioner.
• Solve Ap = d with MICCG(0), i.e. the PCG algorithm with MIC(0) as preconditioner.
• Compute the new velocities u_{n+1} according to the pressure gradient update to u.

2.6 Smoothed Particle hydrodynamics

Smoothed particle hydrodynamic [MCG03] is another method that can be used for numerically solve Navier Stokes equations to simulate fluid. The main advantage of this method over the grid based method is that it requires much less computation and it can be used in real time. The concept is to consider fluid as a collection of particles which are at a specified distance from each other. This distance is called smoothing length. A method is defined to smooth up the properties of the particle over this distance. This is done with a smoothing kernel *W(r)*. This defines a scaler weighting function for the surrounding area around a particles position *xj* with *W(| x-xj |)*. The kernel function can be defined as the following,

$$W_{poly6}(r) = \frac{315}{64\pi d^9}(h^2 - d^2)^3 : 0 \leqslant r \leqslant d \quad 0 : otherwise \tag{6.1}$$

From the particles position *xj* and mass *mj*, a smooth density can be derived,

$$\rho(x) = \sum_j m_j W(|x - x_j|)$$

By integrating this particle density we can obtain the total mass of the system

$$\int \rho(x)\,dx = \sum_j \left(m_j \int W(|x-x_j|)\,dx\right) = \int_j m_j$$

From the individual density we can get the smoothed value *As* of and attribute *Aj* of a particle,

$$A_s(x) = \sum_j m_j \frac{A_j}{\rho_j} W(|x-x_j|) \qquad (6.2)$$

we can get the gradient of this attribute by just putting the gradient of the kernel

There are some advantages on solving Navier Stokes using the approach. The first is that as we are describing the system as collection of particles and the total mass of the system is computed by the individual mass it does not need to worry about mass conservation. The equation for mass conservation can be easily omitted. When it comes to the momentum conservation equation, the term $v.\nabla v$ can be omitted, as we are considering particles are moving with fluid. Then velocity derivative of fluid is same as the derivative of the velocity of particles. The drawback with this system is that when we want to simulate large quantity of fluid, to get the resolution like a simulation done by traditional grid system, we need to use huge amount of particles, which makes the system inefficient. It is not possible to achieve the same level of accuracy in this approach. But in situation where we need to deal with a small amount of fluid this process works very well.

So far I have described the basics of fluid simulation solving the Navier Stokes equation. My main goal is to control the basic flow of fluid through a user interface. There are many techniques that I have come across so far. Some of these techniques added new terms to the basic Navier Stokes and then solved it using the traditional grid based system to simulate the effect [FL04]. In [TKPR06] described forces to control fluid based on the Smoothed Particle hydrodynamics (SPH). For my project I am thinking of experimenting on both these techniques depending on the kind of effect I will be generating as both seems to work quite well in different circumstances, or give animator the option depending on the kind of control he is trying to achieve.

2.7 Controlling fluid flow

For producing various animations, it is very important to be able to control the basic fluid animation and for the animator to be able to control it without any need to get deeper into how the animation is being produced. Following the algorithms discussed so far we can create a fluid animation acting naturally following the rules of physics. If we need to produce much interesting looking fluid animations like making shapes with it or making it follow a specific path and at the same time maintain the natural behaviour of fluid we will need to have some control over the animation.

For my project I will need to use some way of controlling fluid flow which the user will be able to use through a user interface. There have been many researches on this topic as it is a very popular topic these days. In the work of Raanan Fattal and Dani Lischinski in 'Target driven smoke animation' [FL04], they have introduced an interesting and very efficient way of controlling smoke flow. Given sequence of target smoke states, their method can generate smoke simulation in which the smoke is driven towards all these specified target in terms, while maintaining the natural looking smoke like behaviour. They have introduced two additional term [FL04] to the Euler equation of fluid simulation

1. A driving force term which is added as an external force in the momentum equation. This force drives the fluid from current density to a specified density.
2. A gathering term in the advection equation , which works as a counteract to the diffusion of smoke. This gives fluid the ability to form a specific target.

These two terms are defined by the instantaneous states that the animation will try to achieve. This is comparatively simple addition to the computation of the basic fluid simulation. The fluid will have an initial density *ρ(x,0)* at the beginning and there are some sequence of target densities that the fluid will try to achieve those densities while maintaining the fluid like behaviour. If *ρi(x,ti)* is the density at *ti* time then during the time interval *ti-1 to ti*, fluid will be driven towards the *ρi* density. In this paper some parameters are introduced that the animation can manipulate to achieve some control over the animation.

The modification made in this paper on the basic fluid momentum equation is that in case of an external force a new term $F(\rho, \rho^t)$ is introduce. $F(\rho, \rho^t)$ represents the force that acts as an external force to drive the fluid form density *ρ* to target density *ρt.* There needs to be a momentum attenuation term to balance the momentum. The equation now becomes,

$$u_t = -u.\nabla u - \nabla p + v_f F(\rho, \rho^t) - v_d u$$

The gathering term added to the advection equation is $G(\rho, \rho^t)$. This term will prevent the smoke from diffusion too much and match the target density much closely.

$$\rho_t = -u.\nabla \rho + v_g G(\rho, \rho^t)$$

v_f and v_d are non-negative parameters which controls the external force and momentum attenuation. v_g has effect on the gathering term. These will be discussed later.

After defining these two terms, driving force and gathering term , these equations can be solved using the splitting method or the method of fractional steps. Splitting the equations into simpler smaller equation it can be solved more easily and efficiently. The methods is described in previous sections. This two equations are split into the following equation [FL04] to solve using this method,

1. Applying driving forces : $u_t = v_f F(\rho, \rho^t)$
2. Attenuate momentum : $u_t = - v_d u$
3. Advect momentum : $u_t = - u . \nabla u$
4. Project : solve for the pressure field p given by the poission equation $\nabla^2 p = \nabla . u$ then subtract ∇p from u
5. Advect smoke : $\rho_t = -u . \nabla p$
6. Gather smoke : $\rho_t = v_g G(\rho, \rho^t)$

2.7.1 Defining of Driving Force

The purpose of this force is to make the smoke advect from current density to target density. The force has to decrease as it reaches the target density to avoid fluctuations. There will be undesirable fluctuation even if the current density becomes same as the target density. By decreasing the force, the momentum attenuation will bring the system to zero velocity when target density is the same as the current density.

This depends mainly on the gradient of density, in fact in this paper [FL04] the force is derived considering it to be proportional to the gradient of target density. But the problem is in some case it can happen that the density stays constant in some places. Then the gradient of density will be zero. To avoid this the density is used as a slightly blurred value of the actual density so that there will be some gradient even if the actual density gradient is zero.

If the actual density is ρ^t the blurred density is $\tilde{\rho}^t$, which is obtained by applying Gaussian kernel with a small value of σ,

$$g(x)=e^{-x^{Tx}/\sigma^2}$$

σ will have some effect on the smoke animation that will be discussed later.

In this paper [FL04] the force term is defined by,

$$F(\rho,\rho^t)=\tilde{\rho}\frac{\nabla\tilde{\rho}^t}{\tilde{\rho}^t} \qquad (7.1)$$

where, $F(\rho,\rho^t)$ is the force to advect the fluid from ρ density to ρ^t density. $\tilde{\rho}^t$ is the blurred value of $\rho^{t.}$

2.7.2 Defining Gathering Term

In case target density has higher values then initial density then it will be impossible to reach that density because ρ will keep decreasing as a result of numerical dissipation. Even if we avoid numerical dissipation the target value will never be achievable. The advection process will only remap the initial density it is considered to our fluid to be incompressible. Incompressible fluid cannot form a new value for density. The gathering term is introduced by Raanan Fattal [FL04] to solve this situation. It is assumed that the mass of smoke at the initial state is the same as the target state. So if in any state there is a difference between the initial density and the target density or a gradient of density then there will be an error density. In this case a diffusion is applied to smooth this error density as time proceeds. In order to achieve this diffusion the gathering term should be,

$$G(\rho,\rho^t)=\nabla^2(\rho-\rho^t)$$

Considering the fact that the diffusion should occur only when there is a density value, meaning there exists some smoke to avoid negative value and the gathering term is only expected to work only when the target density is approaching, the gartering term is derived to as the following equation,

$$G(\rho,\rho_t)=\nabla.[\rho\tilde{\rho}_t\nabla(\rho-\rho_t)] \qquad (7.2)$$

2.7.3 Effects of Control Parameters

In the definition of the driving force and the gathering term there are some non-negative parameters that allows user to control the effect of there two terms. The parameters introduced here are σ in the Gaussian kernel used for blurring the actual density, $v_f$coefficient of the driving force, v_d

coefficient of the momentum attenuation, v_g coefficient of the gathering term. There effects are described below from,

Effect of σ
The value of this parameter affect the smoothness of the driving force. With a small value of σ smoke will advect towards the target more smoothly. The more the value of σ increases the less finer detail will be preserved in the target density.

Effect of vf
This parameter will have an effect on the speed of the smoke to achieve the target density. With a very low value of v_f the driving force will be very weak, and the smoke will progress to the target density very slowly. For a higher value of v_f, smoke will advect to the target density quicker. It allows animation to speed up or slow down the animation.

Effect of vd
This will control the effect of momentum attenuation. With a higher value of v_d will cause a more controlled viscous like flow.

Effect of v_g
This parameter determines in which rate the smoke will form the target density. With a higher value of v_g will cause less "stray" smoke and the animation will appear to be less natural. With this parameter animation will have control over the gathering term and the effect on how precisely the target needs to be matched or how natural the animation should look.

Even though these control parameter lets animator to manipulate fluid flow but sometimes it can be quite unpredictable. The forming of shape in this approach is enforced by a gathering term which actually reduces the realisticness of actual fluid. Animator has to be aware of the fact that using less influence of gathering term with actually create more realisticness. But at the same time it is unavoidable to get some diffusion. To control this diffusion without gathering term it might be quite hard to form the specified shape. Considering the fact that all the animators need to have some skills on how to control some object and make the desired effect I thought this concept [FL04] to be quite suitable for my project and relatively simple to implement.

2.8 Control Particles

Besides force based ways of controlling fluids, there is another way of controlling fluid simulation using control particles. Control particles will directly influence the fluid velocity which can be generated using a sequence of target shapes. That means the velocities of a naturally flowing fluid will be forced to match perfectly with the velocity of the control particles. As the velocity fields is forces to be completely aligned with the control particles, there will be some unexpected viscosity terms, that will cause smoothing of many finer details.

In the paper "Detail-preserving Fluid Control" by N. Thürey, R. Keiser, M. Pauly and U. Rüde, [TKPR06] a new technique is introduces to control fluid simulation using control particles. They have introduced scale dependent force control to preserve small scale fluid detail. Instead of influencing the velocities through out the whole fluid by the control particles, they applied the control only over the area with the coarse scale component. Using this technique the have avoided the unwanted appearance of viscosity term and maintained the natural behaviours of fluid. The fluid velocity is decomposed into a high and low frequency component and then the control is applied only on the low frequency part. A low pass filter is for smoothing the velocity and then control forces are applied with respect to the smoothed velocity. The control particles are generated using a function which is described in [FM97, FF01]. These control particles can be displaced for every single frame of animation according to the target mesh shape. In [JSW05] a function introduced interpolates a set of values, mean value coordinates, defined at vertices of a mesh. This can be used here to displace the particles for every animation steps. In [JSW05] the steps of mean value interpolation for a triangle mesh is stated as follows,

1. Compute the mean vector m using the equation, $m=\sum_i \frac{1}{2}\theta_i n_i$, θ_i is the length of i th edge, n_i is the inward normal to the edge
2. Compute the weights wi using the equation, $w_i=\frac{n_i.m}{n_{i.}(p_i-v)}$, pi is ith vertex, v is the specified point.
3. Update the denominator and numerator of $\hat{f}[v]$ by adding $\sum_i w_i$ and $\sum_i w_i f_i$ in equation,

$$\hat{f}[v]=\frac{\sum_k \sum_i w_i^k f_i^k}{\sum_k \sum_i w_i^k}$$

Another effective way of generating control particles can be from a previously generated fluid simulation or from any target shape animation sequence. Since this paper demonstrates their fluid control techniques using both Lagrangian methods (SPH) and grid based method (LMB) , they have generated control particles form a subset of fluid particles for SPH or by tracing massless particle velocity in for LMB. They have mentioned two types of forces to control fluid simulation.

1. Attraction Force
2. Velocity Force

2.8.1 Attraction Force

This force is used to make fluid concentrate towards the control particles. The purpose is to make fluid form a desired shape as specified through the control particles. But too much of this pressure make fluid losing some of its natural fluid like behaviours. So this force is scaled down when the control particles already have enough fluid around them to act on. The attraction force *fa* acting on a fluid element *e* is stated as,

$$f_a(e)=w_a\sum_i \alpha_i \frac{p_i-x_e}{\|p_i-x_e\|} W(d_{i,e},h) \qquad (8.1)$$

w_a is a non negative constant here which defines the strength of the attraction force. Animator can control the attraction force by changing this as a parameter. High value of this parameter will cause a stronger attraction force which might lead to an unrealistic looking fluid, or give a much sharper formation of the desired shape. Animator can control these parameters to get the desired effect.

In the definition of the attraction force, *αi* is the scale factor of the attraction force. $d_{i,e} = || p_i - x_e ||$ which specified the distance between p_i and the centre x_e of the fluid element *e*

$$\alpha_i = 1 - min(1, \sum_e V_e W(d_{i,e}, h))$$

W is the normalised spline kernel with support *h* [MCG03]

$$W(d, h) = \begin{matrix} \frac{315}{64}\pi h^9 (h^2 - d^2)^3 : d < h \\ 0 : d \geqslant h \end{matrix}$$

2.8.2 Velocity Force

This force is applied to control the velocity of the fluid. The control particles will determine the target velocity that needs to be achieved. The velocity force applied on the fluid element *e* will make the fluid flow towards the target velocity specified by the control particles. The velocity force is almost similar to the attraction force.

$$f_v(e) = w_v \sum_i (v_i - v(e)) W(d_{i,e}, h)$$

In this equation w_v is a constant that defines the influence of the velocity force on the animation. Higher value of this constant will make the animation achieve the desired velocity of control particles faster as the force will be stronger. This can act as a parameter that the animator can control to get the desired effect.

2.8.3 Total Force

The total force per volume acting on the fluid simulation is the sum of the attraction force and the velocity force. It is stated as,

$$f(e) = f_c(e) + f_f(e)$$

f(e) is the total force applied on per volume fluid unit e. This force is used to get new velocity field which is used in the next SPH step.

From the velocity force equation above, it can be easily noticed that the force is derived form the overall velocity of the fluid. This will cause an averaging on the velocity and we will get an unwanted viscosity term. To avoid this, the technique used here is to separate out the small detail portion of the velocity from the overall velocity then apply the velocity force on the less detailed portion. The less detailed portion will follow the control particles as desired and at the same time the finer detailed portion will be preserved as the velocity force is not working on that. The equation for obtaining the smoothed velocity $\tilde{v_f}$ is stated below. This will replace the *v(e)* in the definition of the velocity force to get the desired effect.

$$\tilde{v_f}(e) = \frac{\sum_i \tilde{v}_i W(d_{i,e}, h)}{\sum_i W(d_{i,e}, h)}$$

In this equation, $\tilde{v}_i$ is the filtered velocity of the control particles, which is derived from the current fluid velocities $\acute{v}_f$

$$\tilde{v}_i = \frac{\sum_e \acute{v}_f(e) W(d_{i,e}, h)}{\sum_e W(d_{i,e}, h)}$$

Controlling fluid in this approach clearly gives a more realistic looking fluid simulation. Filtering the velocity field into hight pass and low pass components effectively solves the problem of smoothing out the finer details, and controls the fluid flow preserving all the nice details like splash and spills. But the only problem is that it can be quite difficult to control the flow accurately. The main control parameters that determines the effect of the forces w_a , w_v can be a bit complicated and unpredictable to achieve desired effect. These two parameters need to be balanced according to the target shape and dynamics. If the velocity control parameter is stronger with respect to attraction force, this might cause fluid to move pass somewhere unexpected without forming the shape. Animator will need to keep experimenting on w_a and w_v and get the right value to achieve the right effect. It is almost in every case that animator needs to gain some experience on using a tool, so this is not totally unexpected from an animator's point of view.

2.9 Usability Evaluation

In order to develop a successful software, it is necessary to make sure that it is usable by the targeted audience groups. For instance, if the purpose of a software is to develop new design concepts, a designer should be able to learn and use the tool to fulfil their goals of developing new design concepts as quickly and easily as possible.

The purpose of conducting a usability evaluation is to assess the accessibility of the system's functionality, user experience of using the system and to identify any problems with the system [DFAB04]. This allows the software development team to understand the usability of the system and take actions or modify based on the results of the evaluation.

There can be two types of evaluation techniques:

- Expert Usability Analysis
- Involving end users

The first type of usability evaluation requires independent usability experts or designers going through a system to identify any usability issues associated with it. Example of such techniques are heuristic evaluation [NR90] and Cognitive walk through [WRLP94]. These techniques can be used at the early stages of the design process to identify any usability issues associated with the system.

The second type of usability analysis involves end users from the targeted audience groups. Example of these techniques are ethnographic studies, experimental design and cooperative evaluation [MWHD93].

The cooperative evaluation requires the end user and the designer of the system, where both party works collaboratively to identify usability issues during a session. This method uses a think aloud protocol where users are prompted think out loud even if it seems obvious. The benefits of this approach are:

- as this technique is less constrained, it is easy to learn and apply;
- users are prompted to provide their honest opinions and criticise the system;

- exploration capability of the designer by asking 'why' or 'how' to identify user expectation and negative aspects of the user interaction during the usability testing session.

Although the expert usability analysis techniques can be cheap, quick and dirty, due to resource constraints, it would be difficult to implement such methods. However, a number of students of the University of Bristol, who are interested in developing animation would be the target audience group of this system. Due to the availability of the targeted audience group and additional benefits of the cooperative evaluation, it was decided that after developing the first version of the animation tool, a cooperative evaluation will be conducted. This would allow to surface any usability issues and improve the system based on user feedback.

2.10 Summary:

As interesting as fluid simulation looks and sounds, there is a massive amount of theory and computation that works behind a realistic looking fluid animation. I have looked into various sources starting form Wikipedia to many research papers to get a flavour of it. For the background knowledge on this topic, I have got a good understanding of the basic concepts and equations for fluid simulation. I went through the physics behind fluid behaviour, then I understood how the "Navier Stokes" equations represent fluid flow according to the laws of physics. Having a good understanding on Navier Stokes, I looked for efficient and effective numerical methods to solve the equations. I have mentioned some additional terms introduced with the basic Navier Stokes and how they influence the fluid flow in [FL04]. Some of these terms has direct control over the speed and sharpness which is very useful from my project perspective. With a specified set of target shapes it can control the fluid to form shapes and there are parameters to control the speed and sharpness. Even though these control parameter gives animator to manipulate fluid flow but sometimes it can be quite unpredictable. The forming of shape in this approach is enforced by a gathering term which actually reduces the realisticness of actual fluid. In some cases there are trade-offs between achieving a more natural looking fluid and how closely the target shape is matched. Animators will need to keep experimenting on the control parameters and set the right value to achieve the right effect. It is almost in every case that animators need to gain some experience on using a tool, so this is not totally unexpected from an animator's view. For this project my goal is to make an effective user interface to make use of this control parameters.

3 Project Design

3.1 Introduction

The project is designed in four sections – Implementing the solver, Generating display for the smoke simulation, Processing inputs for the user interface and at the end designing and implementing the features for the user interface. The fluid solver integrated with the display and GUI performs as a complete animation tool that a user can use to produce smoke animation.

The system is built using C++ programming language. Building the user interface takes advantage of the object oriented feature of C++. Posix thread is used for the solver to run on threads and work simultaneously with the other components of the system and generate a real-time smoke display on the screen.

The solver works in four stages – Applying forces, Advect smoke, Mass conservation and at the end applying gathering term. Generally simulating fluid is considered as one of the most complicated problems in engineering. For this project the implementation of the basic solver is largely based on the 'A Simple fluid solver based on FFT' by Jos Stam. This paper presents a relatively simple approach of implementing the basic fluid simulation. Considering the fluid to be wrapping around in space as a continuous flow has simplified the implementation to some extend. Performing the mass conservation in the Fourier domain reduced some aspects of complication in the implementation and also produces the results fast enough to display a real-time animation.

Some additional terms needed to be implemented on top of the basic fluid solver for controlling and forming shapes with smoke. The additional two terms introduced in the 'Target-Driven Smoke animation' by Rannan Fattal and Dani Lischinski were very helpful to achieve this. The 'Driving force' term is implemented in the 'Applying force' step of the solver and at the end the 'Gathering term' is derived and added with the density. The advection step works mostly as described in Stam's fluid solver.

Before the solver threads start running, the system needs to be initialized for the following: Fourier transform, Gaussian kernel (used to derive smoothed version of density to implement driving force and gathering term), velocity grid, density grid (used for finite differences) and all the parameters used to control the smoke animations with a default value. As the solver works the velocity and the current density grids are updated every time step. The updated density field are displayed on the screen using OpenGL at every time step.

FLTK (Fast Light Tool kit) is used to design and built a user interface integrated with the system. The system is designed to work with three types of inputs – Text input, Image input and Animation input. The inputs need to be processed and formed as target densities when they are selected. The control parameters that are implemented in the system are – driving force, momentum force, gathering force, viscosity and smoke amount. The control parameters are implemented using sliders, which will update the parameters in real time and the effect is visible on the screen as the values are changed. The user interface is designed considering the important usability aspects. The completed system provides a user with various usable control features and settings to produce interesting looking smoke animation.

3.2 Fluid Solver

The fluid solver implemented for this project is based on Stam's 'A Simple Fluid Solver' . The main fluid solver is depending on the velocity field that is directed by some external forces. The forces are derived from the 'Target-Driven Smoke Animation' by Raanan Fattal and Dani Lischinski. The force terms introduced in this paper are derived from a target density and current density fields mentioned in the background section. The velocity and the density fields are defined on a 2d grid system considering the velocities are working on the centre of each grid. This grid system can be implemented in 3d as well but for this project only the 2d system is considered for simplicity. The velocity is continuous across the edge of the grid, so it can be tiled on the screen and used as a texture map. Figure 3.1 shows the continuous periodic velocity on grid. It would look like a continuous flow of fluid on the screen.

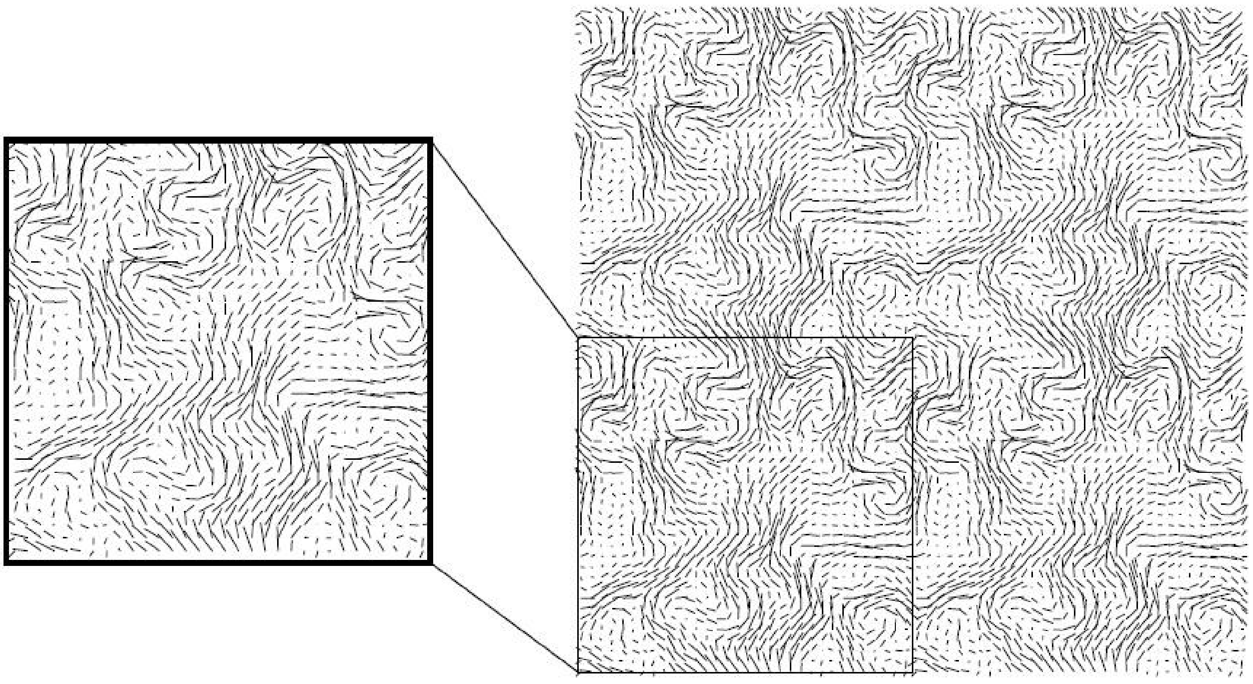


Figure 3.1: Periodic fluid velocity on grid, this image is taken from Stam's Simple fluid solver [Sta01]

The velocity and density fields are updated on every time step. The update of velocity and current density can be described on the following steps,

- Apply Driving forces
- Advect Fluid
- Mass Conservation
- Gather Smoke

For this project the fluid solver generated smoke display in real time. The grid is sliced into sections and each section is updated in the four steps mention above simultaneously. POSIX Thread is used for this purpose. Each thread solving the grid in sections at the same time makes the whole grid updated at once in much less time.

Solver thread:

```
• While (!stop)
    ○ Apply Driving forces
    ○ Advect Fluid
    ○ Execute Fourier transform
    ○ Mass Conservation
    ○ Back to spatial domain
    ○ Gather Smoke
    ○ Update velocity
    ○ Update density
    ○ If Close Window, stop
```

3.2.1 Initialization:

The default value for all the control parameters, grid size, thread numbers and time step dt are set at the beginning of the initialization. The grid size is required to be set with a value which is a factor of 2 for simplicity of computation.

Initialization of density arrays:

The solver requires a set of density array to store the calculated densities. Basically there are two density arrays for storing – Target Density and Current Density. Later in this chapter it is mentioned that Fourier transform and Gaussian blur are performed on the density fields. To store these values another two sets of arrays are needed for both Target and Current densities. All these arrays are allocated memory space for the specified grid size and initialization. To generate the first frame for the display the target density is assigned to a circular shape positioned at the centre. The initial smoke amount is considered as the area of the circle. Any point on the grid that falls inside the circle sets the value for the target density with '1' and for the points that are outside with '0'.

- Circle area = smoke amount
- Circle radius ^2= smoke amount / π

- For Each Point on the Grid:
  - IF point inside circle
  - Target_Density for the point = 1
- Else
  - Target_Density for the point = 0

The target density array is then copied to the current density array to display the first frame and the starting point for the solver to work on.

Initialization of velocity arrays:

The solver updates the current velocity for each time step. To store the velocity three arrays are required – current velocity, Fourier transform of velocity and a temporary velocity. Temporary velocity stores the velocity updates for the in-between steps of the solver and as all the steps are completed the current velocity is updates. For the initialization, memory space is allocated for all the velocity arrays for the specified grid size and set the values to '0' to start with. Memory allocation uses fftw_malloc() so that is allocated enough memory for holding complex data when Fourier transform is done. Both x and y dimension of the velocity are stored in a single array. For this reason the size allocated for the velocity field is twice the size of the grid. The x values are stored in even positions and the y values are stored in odd positions. The computation for both the values can be done looping through the single velocity array.

Initialization for Fourier transform:

Before making any use of the FFTW3 library for Fourier transform it needs to be initialized. To initialize the Fourier transform thread the following two functions need to be called with the number of threads specified.

```
fftwe_init_threads();
fftwe_plan_with_nthreads(threadno);
```

These two functions need to be called only once at the very beginning. This will initialize the thread required to perform the Fourier transform. Before executing Fourier transform a Fourier plan is needed to be formed using fftw_plan functions. The input array and the output array needs to be sent as parameters. fftw_plan contains all the necessary information to perform the Fourier transform. The Fourier plans are generated in the initialization step so that only executing the fftw_plan will perform the Fourier transform in the solver much faster. This helps our real time smoke display. Because the only mass conservation step in the solver is done in the Fourier domain, the arrays need to be back into spatial domain. For this an fftw_plan for the reverse from Fourier is also needed to be constructed. Executing this plan after the mass conservation term will get the velocity back into spacial domain. The functions used to construct the plans are,

fftw_plan fftw_plan_dft_r2c_2d(double *in, fftw_complex *out) – spatial to Fourier
fftw_plan fftw_plan_dft_c2r_2d(fftw_complex *in, double *out) – Fourier to spatial

Initialization for Gaussian kernel:

It is required to construct a Gaussian array to preform the Gaussian blur on the density fields when needed in the solver. In the paper [FL04] the driving force and the gathering force are derived using the Gaussian blurred version of the density field. The reason is to avoid the gradient of the target density to become zero in some region.

The Gaussian kernel in 2d has the form,

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{\frac{-x^2+y^2}{2\sigma^2}}$$

This gives a circular symmetric fall off of the filter which is suitable for the density field for this project. Too much fall off might cause unnecessary smoothing in the direction of the gradients. A small value for sigma is used in this case.

At first the Gaussian array is allocated memory for the same size as the grid and all values set to zero. For a suitable value for sigma the coefficient $\frac{1}{2\pi\sigma^2}$ and the denominator $\frac{1}{2\sigma^2}$ is calculated. The steps to compute the Gaussian array for a grid size x_size * y_size are the followings,

- Coefficient = 1.0 / (2.0 * PI * sigma * sigma)
- denominator = 1.0 / (2.0 * sigma * sigma)
- Gaussian_size = max (x_size, y_size)
- For (j = -Gaussian_size; j < Gaussian_size; j++) :
  - Gaussian_index_y = j & (Y_SIZE – 1)
  - For (i = -Gaussian_size; i < Gaussian_size; i++)
    - Gaussian_index_x = i & (X_SIZE – 1)
    - Gaussian_array[Gaussian_index_x, Gaussian_index_y] +=coefficient*exp(-((sqr(i+sqr(j))*denominator))

The Gaussian array works as the Gaussian kernel to smooth out the density fields. This multiplication is done in the Fourier domain. For this the Gaussian_array is transformed into Fourier domain. This is done as part of the initialization before the solver starts. As the Gaussian kernel is constructed in the initialization step all that is required to get the smoothed version of the density is,

Smooth Density:

- Execute plan Fourier transform for density field
- Multiply the density with Gaussian kernel
- Execute plan for reverse Fourier transform for density

POSIX Thread Initialization:

Before the solver starts four POSIX threads are initialized. The number of threads can be varied if larger grid is used. The y size of the grid is divided by the number of threads to determine the size of the sections each thread will be working on. Then each thread is started with two parameters - the starting of the section the thread will be dealing with and the other one is the size of the section. Each thread then performs the following four steps of the solver in the assigned grid sections and updates the velocity and current density.

3.2.2 Applying Force:

The external force of the basic fluid equation is replace by the driving force term and a momentum attenuation term. The driving force is defined by the following equation in the previous chapter 2.7.1

$$F(\rho, \rho^t) = \tilde{\rho} \frac{\nabla \tilde{\rho}^t}{\tilde{\rho}^t}$$

$\nabla \tilde{\rho}^t$ here is the gradient of the Gaussian blurred version of the target density. All the density fields used in this section are the Gaussian blurred version of the actual density so that the gradient never becomes zero. This term is solved using the finite differences in the computational grid. Using finite differences the gradient term breaks into the following equations.

$$\frac{\delta \tilde{\rho}^t}{\delta x} = \frac{\tilde{\rho}^t(x+1, y) - \tilde{\rho}^t(x-1, y)}{2}$$

$$\frac{\delta \tilde{\rho}^t}{\delta y} = \frac{\tilde{\rho}^t(x, y+1) - \tilde{\rho}^t(x, y-1)}{2}$$

These two terms are calculated from the density values of the the surrounding blocks in the grid. In this case the density array storing the Gaussian blurred version of the target density is used. For each position of the grid in the scan line the blurred current density is divided by the blurred target density then multiplied with the vf (parameter that controls the effect of driving force) and time step dt. This value is then multiplied with the x and y component of the $\nabla \tilde{\rho}^t$ term described above and stored in a temporary array. The steps are the followings,

- Allocate memory for a driving force array for twice the size of the x size of the grid
- Fill the driving force array with zero
- For each grid cell on the scan line:
    - Get x component for the target density gradient:
        - Subtract the target density of the left grid cell from the target density of the right grid cell and divide by 2
    - Get y component for the target density gradient:
        - Subtract the target density of the lower grid cell from the target density of the upper grid cell and divide by 2
    - fill even position on the driving force array with,

vf *dt*xcomponent_ofDensityGradient * current_density/ target density

- fill odd position on the t driving force array with,

vf *dt*y-component_ofDensityGradient * current_density/ target density

At this stage we have the driving force array containing the x and y components of the driving force, described as $\tilde{\rho}\frac{\nabla\tilde{\rho}^t}{\tilde{\rho}^t}$ in the even and odd positions.

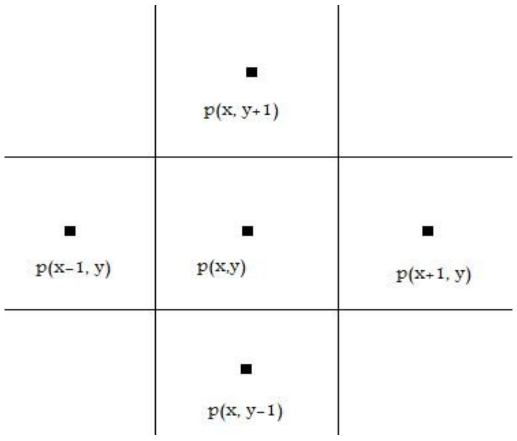


Figure 3.2 : Density of the computational grid

Now as in the background it is mentioned that a momentum attenuation term needs to be subtracted to balance the applied force. This term is defined as 'vd*velocity' in 2.7.1 section. To do this all the x component of the velocity is multiplied with vd(parameter that controls the effect of momentum attenuation) and then subtracted from the x components of the driving force stored in the driving force array, then update the x component of the velocity with it. The updated velocity is stored in a copy of the velocity array. The same process it applied for the y component. The x and y components are stored in the even and odd positions in the array same way as the temporary array storing driving force. The steps to achieve this is a loop through the driving force array and the velocity grid cell for twice of the x-size of the grid,

- While (i< 2*x-size)
- velocity_copy_cell = velocity_cell [i] - vd*velocity_cell [i] + driving_force [i]

At this stage we end up with a temporary copy of the velocity cell updated with the added driving force and the implemented momentum attenuation for the scan line.
As the thread loops through all the the scan lines, the section of the grids velocity that the thread is dealing with is updated with all the forces applied in a temporary copy of the grid.

3.2.3 Advect Smoke:

After applying driving force and momentum attenuation the advection step is applied to update the smoke velocity. The advection step relate to the non-linear property of the fluid causing the unpredictable and chaotic behaviour of the fluid flow. It is the process of moving the fluid's velocity from one place to another by the movements of the fluid particles. In this project the implementation of the advection step uses the 'Self Advection' section of Stam's fluid solver[Sta01]. In this paper the 'semi-lagrangian' method has been used for the implementation. This method is explained in the background part (section 2.4 Advection algorithm) in detail. The implementation in this paper works like this, for each cell of the velocity grid the midpoint is traced backwards through the velocity field over the time step dt. The point will end up somewhere else in the grid. The velocity is then interpolated linearly at that point from the neighbouring cells and transfer this velocity back to the departure cell. The linear interpolation is simple to implement here because from the beginning the grid is considered to be continuous for the solver. So the points that are interpolated outside the grid simply re-enters the grid from the opposite side.

For the solver in this project it is required to apply the process both on the velocity grid and the density grid. After applying all the forces on the velocity grid the update is stored on the copy of the grid. This copy is used here for the linear interpolation and the update is stored in the original velocity grid. The solver thread scans each line of the section of the grid and apply the advection process in the following steps,

- Define 'pos' as starting position of the grid section
- For each point on the scanline:
  - i0 = X_Position_on_Scanline - dt*X_GRIDSIZE*Velocity_on_copy_cell[pos]
    j0 = Y_Position_on_Scanline - dt*Y_GRIDSIZE*Velocity_on_copy_cell[pos+1]
    i0 = (i0 + X_GRIDSIZE) % X_GRIDSIZE , i1 = (i0 +1) % X_GRIDSIZE
    j0 = (j0 + Y_GRIDSIZE) % Y_GRIDSIZE , j1 = (j0 +1) % Y_GRIDSIZE
  - Update velocity on cell[pos] interpolating on copy grid for (i0,j0) , (i0, j1), (i1, j0), (i1, j1) for even positions
  - Update velocity on cell[pos+1] interpolating on copy grid for (i0,j0) , (i0, j1), (i1, j0), (i1, j1) for odd positions
  - Increament pos by 2

In all the velocity arrays the x components are stored in the even positions and the y components are stored in the corresponding odd positions. So in the above steps [pos] and [pos+1] holds the x and y components of velocity and updated by the even and odd positions from the copy grid.

A copy grid for the current density is made and the same steps are applied to update the current density fields of the solver using linear interpolation on the copy density grid. At the end of this step the velocity and the density field is advected to the next time step.

3.2.4 Mass Conservation:

To maintain the natural fluid behaviour mass conservation needs to be applied after the adding forces and advection step. Fourier transform is used in the mass conservation step according to Stam's fluid solver[Sta01], because it makes the operations very simple to implement. To conserve mass the equation $\nabla \cdot v = 0$ requires to work with differential equations which can become very complicated. In this case after applying Fourier transform on the velocity field transforms the differential equations into a simple step of multiplication.

The velocity field is actually the sum of a mass conserving field and the gradient field. In figure 3.3 it can be noticed that the mass conserving field has a swirly pattern. For the fluid to conserve mass at every stage this type of behaviour is expected. The gradient field shows a pattern where the fluid is either completely inflowing or completely out flowing. After performing Fourier transform it can be noticed in figure 3.4 that these two fields are perpendicular with each other. If fact the mass conserving field is perpendicular to the wave number. So by projecting the velocity on a line perpendicular to wave number is actually forcing the velocity to be mass conserving.

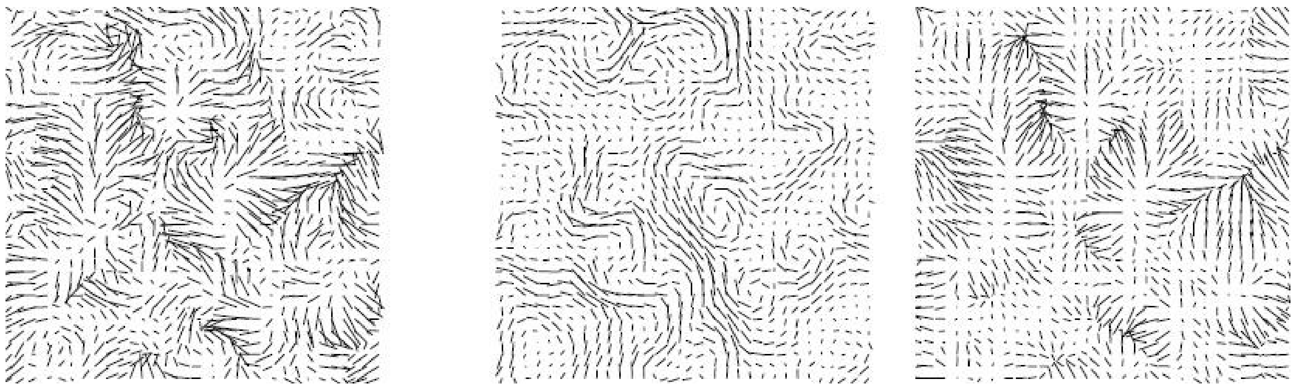


Figure : 3.3 Velocity field is the sum of a mass conserving field and a gradient field, This image is taken from Stam's Simple fluid solver [Sta01]

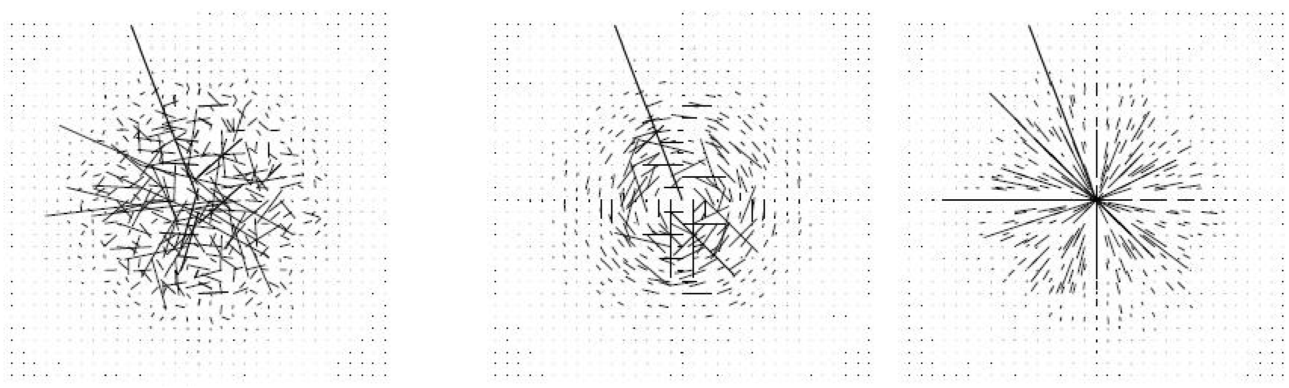


Figure : 3.4 The corresponding decomposition of fig 3.3 after Fourier transform. This image is taken from Stam's Simple fluid solver [Sta01]

It is observed that the velocity vectors are of smaller value when the number of wave is high, since the velocity field is relatively smooth. The effect of viscosity of a fluid is that it smooths out the velocity. After transforming the velocity into Fourier domain it is multiplied by a filter, which is derived from the wave number and the viscosity of the fluid. The decay of the filter depends on these values. In the implementation of Stam's solver the filter is derived as

f = exp(k^2 * Visc * dt) , here dt is the time step, k is wave number, visc is viscosity of fluid

The steps for mass conservation are the followings,

- Execute Fourier transform on velocity field
- Filter $f = \exp(r * Visc * dt)$
- For every position on velocity grid
  - wave number $r = x^2 + y^2$
  - Project velocity on a line perpendicular to r
  - Multiply projection with f
  - Update velocity
- Execute reverse Fourier transformation velocity field
- Normalize velocity field

After performing the mass conservation the velocity field is transformed back into spatial domain to perform the other steps. The fftw3 library is used to transform the velocity into Fourier domain and then again back into spacial domain.

After the mass conservation the velocity field is normalized by dividing each velocity in the grid by the total value.

3.2.5 Applying Gathering Term:

At the end of the solver the gathering term is applied on the density field. This step works on the updated density field in the advection step. Adding this term helps the current density to form the target shape by reducing the amount of stray smoke. This term is only effective in the situation where the current density is approaching near to the target density. The gathering term is defined in section 2.7.2. It is derived as the following equation,

$$G(\rho, \rho^t) = \nabla . [\rho \tilde{\rho}^t \nabla(\rho - \rho^t)]$$

This term is added to the current density multiplied by vg, the constant that controls the gathering effect of the smoke. This term is solved using finite differences on the density grid. The term $\nabla(\rho - \rho^t)$ breaks into,

$$\frac{\delta(\rho - \rho^t)}{\delta x} = \frac{\rho_{(x+1,y)} - \rho^t_{(x+1,y)} - \rho_{(x-1,y)} + \rho^t_{(x-1,y)}}{2}$$

$$\frac{\delta(\rho - \rho^t)}{\delta y} = \frac{\rho_{(x,y+1)} - \rho^t_{(x,y+1)} - \rho_{(x,y-1)} + \rho^t_{(x,y-1)}}{2}$$

The gathering term is formed by multiplying these two term by $\rho \tilde{\rho}^t$ and taking the divergence of the terms

$$\nabla.[\rho\tilde{\rho}^t\nabla(\rho-\rho^t)]=\frac{\delta}{\delta x}(\frac{\delta(\rho-\rho^t)}{\delta x}*\rho\tilde{\rho}^t)+\frac{\delta}{\delta y}(\frac{\delta(\rho-\rho^t)}{\delta y}*\rho\tilde{\rho}^t)$$

The x component in the equation can be broken into the following term using finite differences

$$\frac{((\frac{\rho_{(x+2,y)}-\rho^t_{(x+2,y)}-\rho_{(x,y)}+\rho^t_{(x,y)}}{2})*\rho_{(x+1,y)}\tilde{\rho}^t_{(x+1,y)}-(\frac{\rho_{(x,y)}-\rho^t_{(x,y)}-\rho_{(x-2,y)}+\rho^t_{(x-2,y)}}{2})*\rho_{(x-1,y)}\tilde{\rho}^t_{(x-1,y)})}{2}$$

The x component in the equation can be broken into the following term using finite differences

$$\frac{((\frac{\rho_{(x,y+2)}-\rho^t_{(x,y+2)}-\rho_{(x,y)}+\rho^t_{(x,y)}}{2})*\rho_{(x,y+1)}\tilde{\rho}^t_{(x,y+1)}-(\frac{\rho_{(x,y)}-\rho^t_{(x,y)}-\rho_{(x,y-2)}+\rho^t_{(x,y-2)}}{2})*\rho_{(x,y-1)}\tilde{\rho}^t_{(x,y-1)})}{2}$$

The gathering term is now the sum of these two terms that can now be solved using the values from the density grid and the smoothed version of the target density. For each grid cell the gathering term is derived from the density of the surrounding current density and target density grid cell. The solver thread will calculate the gathering term from the section of the density grid, multiply it by vg and add it with the current density of the section. The process can be explained with the following steps,

- For each point in the density grid:
  - Compute gathering term for the current position
  - Add gather term with current density
- Smooth current density using Gaussian blur

3.3 Generating Display

As the solver works, it keeps updating the velocity and the current density arrays. It starts working on the initial density and updates it at every time step and gradually reaches the target density. To display the smoke it is required to generate a visual representation of the current densities. OpenGL (Open Graphics Library) is used to achieve this. OpenGL has got the useful functions and commands through which geometric objects can be specified in two or three dimensional space. It has got the feature to specify how the object is going to be rendered in the frame buffer. While rendering images with OpenGL texture or lighting can be enabled or disabled as required which is very useful for our smoke display. OpenGL alone does not come with any function to create user interface or any sort of user interaction. But there are many utility libraries that are built to work with OpenGL to provide the features that are not available. There are libraries like SDL and GLUT providing window facilities which came very useful for this project. For building user interface there is GLUI which is simple to use, but comes with a very few features. FLTK is another library to build user interface on top of OpenGL. For this project a good use of OpenGL and the associate utility libraries have been made for displaying smoke and building the user interface.

When OpenGL is initialized, a function glGenTextures (GLsizei , GLuint *) is called to create a Glunit texture name. Binding this texture name with a target makes the texture usable for rendering. The function glTexImage2D( ) will use the density array as the pointer to the texture image and load it as a texture into the target. After OpenGL is initialized as texture enabled, each time a frame is rendered, glTexImage2D( ) loads the current density array into a texture target, which is GL_TEXTURE_2D. Binding the target with a Glunit makes the texture ready to be used. For each frame a rectangle is drawn and the texture is placed on the rectangle.

- For each frame
  - Reset current matrix
  - Enable texture
  - Load the current density into texture target
  - Bind texture
  - Set colour
  - Draw rectangle
  - Disable texture

While drawing the rectangle a texture coordinate is specified for each vertex. It is set in such a way that the rectangle covers the whole screen and the texture appears aligned to the screen without being repeated. Drawing the rectangle with the updated texture for every frame works as a nice display of the current density array generated by the solver in every time step.

3.4 Processing Inputs

As mentioned earlier the solver works based on a target density and velocity. When the target density changes, the velocity is generated from that new target density and the solver keeps updating the velocities as it proceeds. To Implement the feature that user can specify a target it is required to convert from a text or an image to a density array. The SDL library is used for this purpose. The idea is to load the text and image into a SDL surface and then convert each pixel on the surface into an array of values indicating the target shape. Before any data is read from the SDL_Surface it needs to be locked and then unlocked after the data has been processed. The steps that are taken to generate a target density array from a specified input are the followings,

Generate Target Density:

- If input is a text
  - Load font
  - Draw text on SDL_Surface using the font

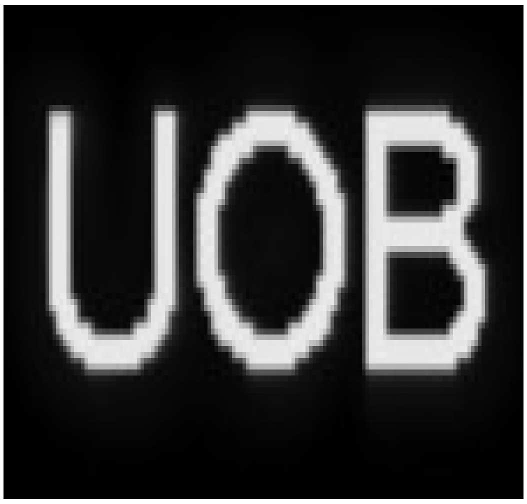
- If input is image
  - Draw image on SDL_Surface

- Convert the specified target coordinates on the display screen into array index
- Lock SDL_Surface
- Calculate the size of the target in terms of non zero pixel values on the SDL_Surface
- Point to the specified index of the Target_Density array
- For every pixel on the SDL_Surface
  - If pixel value is non zero
    - Entry in the Target_Density array = 1 * smoke_amount / target _size
  - Else
    - Entry in the Target_Density array =0
- Unlock SDL_Surface

All the non zero entries in the target density array specifies the target shape for the solver. Each time a target text or image is specified by the user the target density array is updated.

Text Input:

The SDL_ttf library is used for the text input. Calling the function 'TTF_OpenFont(file, size)' will load the font and font size specified by the user. The function 'TTF_RenderUTF8_Solid(font, text)' will then take the text input from the user and render the text with the font on an SDL_Surface. Then every pixel in the surface is checked. If the value is '1' the target density array gets a non zero value, and zero otherwise. While loading the text from the SDL_Surface to the array, it is placed on a position specified by the user.

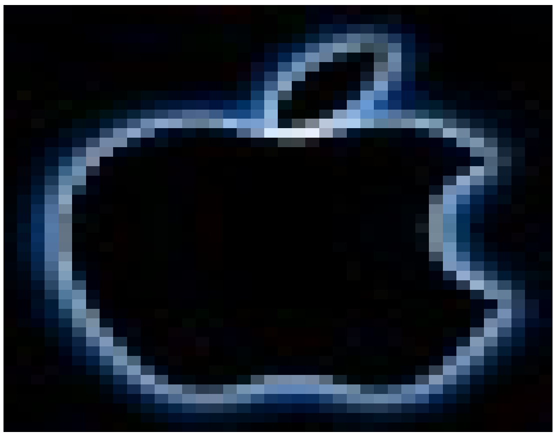


```
0000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
0011100000000000111000000000111111100000000011111111111000000
0011100000000000111000000011111111111000000011111111111110000
0011100000000000111000001111111111111100000011111111111111000
0011100000000000111000011111000000111110000011100000001111100
0011100000000000111000011100000000111000000011100000000011100
0011100000000000111000111000000000001110000011100000000011100
0011100000000000111000111000000000000111000011100000000011100
0011100000000000111001110000000000000011100011100000000011100
0011100000000000111001110000000000000011100011100000001111000
0011100000000000111001110000000000000011100011111111111110000
0011100000000000111001110000000000000011100011111111111110000
0011100000000000111001110000000000000011100011111111111111000
0011100000000000111001110000000000000011100011100000000111100
0011100000000000111001110000000000000011100011100000000011110
0011100000000000111001111000000000000111100011100000000001110
0011100000000000111000111000000000000111000011100000000001110
0011110000000000111100111100000000001111000011100000000001110
0001110000000001110000001111000000001111000011100000000011110
0001111100000011111000011111100000011111100011100000000011100
0000111111111111110000001111111111111110000011111111111111100
0000011111111111000000000011111111110000000011111111111111000
0000000111111100000000000000111111000000000011111111111000000
0000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
```

Figure 3.5 : Target text converted into density array on the right.

Image Input:

The SDL_image library is used for the image input. An image of any shape on a black background is used, Indicating the coloured part of the picture to be the target shape. The function SDL_LoadBMP(char*) loads a bitmap image and returns a SDL_Surface rendering the image in it. It is then loaded on the array in the same way as the text. However, as the image will have three values corresponding Red, Green and Blue for one pixel, the average of these three values are considered to identify the target shape. If the value is anything other than zero that is considered as part of the target shape.



```
00000000000000000000000000000000111111000000000000000
00000000000000000000000000011111111111111000000000000
00000000000000000000000011111111111111111110000000000
00000000000000000000001111111100000111111110000000000
00000000000000001110001111111100000011111111000000000
00000000011111111111111111100000011111111111111110000
00000111111111111111111111111111111111111111111110000
00011111111111000000000111111111110000000011111111000
11111111111100000000000000000000000000000001111110000
11111111100000000000000000000000000000000011111111000
11111000000000000000000000000000000000000011111111110000
11111000000000000000000000000000000000000111111110000
11111000000000000000000000000000000000000111111110000
11111000000000000000000000000000000000000011111111110000
11111111000000000000000000000000000000000001111111111
11111111000000000000000000000000000000000000011111111
11111111000000000000000000000000000000000000011111111
00011111111000000000000000000000000000000000011111111
00011111111111000000000000000000000000000001111111100
00000111111110000000000000000000000000001111111111100
00000000111111111111111111111111111111111111111100000
00000000000111111111111111111111111111111111110000000
00000000000001111111111100000011111111110000000000000
```

Figure 3.6 : Target image converted into density array on the right.

Sequence of Image Input:

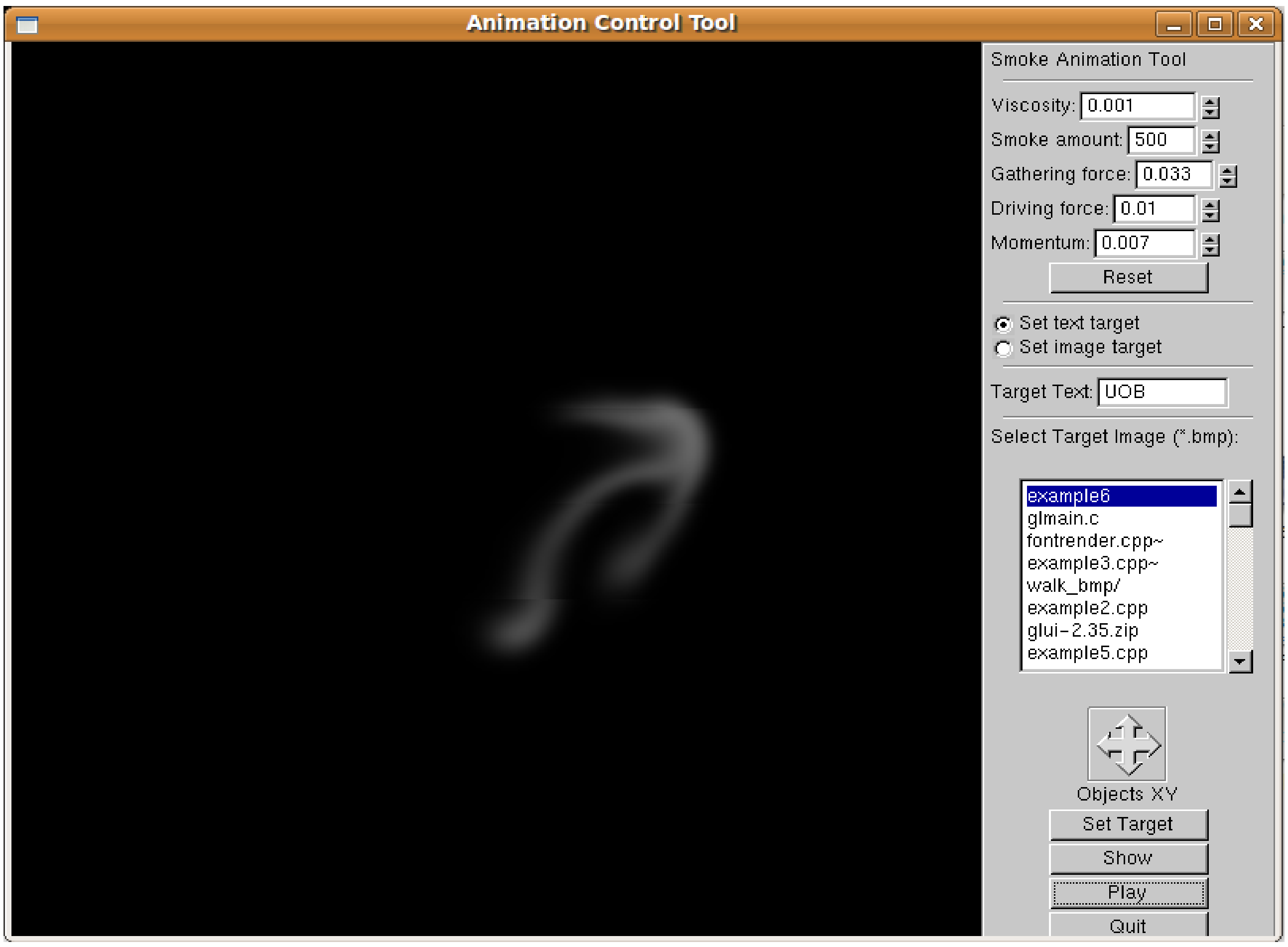
Loading sequence of images uses the same process for loading an image. The sequence of target images needs to be on a black background. A thread is created which will load all the images specified by the user. Then it will set the target density by the images one after another with a delay specified by the user.   The thread will keep updating the target densities and the smoke will be proceeding towards each target during the time interval specified by the user.

3.5 Building the User Interface

As OpenGL does not come with a feature to build a user interface, it was necessary to look into many utility kits that can be used to build a user interface on top of an OpenGL program. Few utility kits of this type were available. But unfortunately many of them did not have a proper documentation. The OpenGL Utility Toolkit (GLUT) is a well known user interface library for OpenGL applications. It has the simple feature for handling a window, mouse movements and keyboard inputs. But as for this project it was required to build a user interface which will have features like buttons, sliders through which user can interact with the system visually, using only GLUT does not seem to cover all the features that is needed. Specially where building a usable user interface is an important aspect of this project.

3.5.1 Using GLUI for simple features

GLUI is a GLUT-based C++ user interface library which has got the features like buttons, list-boxes, check-boxes, spinners, radio buttons etc. that were not available in GLUT . It works on top of GLUT and handles the system-dependent issues, such as window and mouse management depending on it. GLUI comes with the basic input features with a very simple implementation. One single line code can create a button for the user interface. Looking at the documentation it was considered to be a good starting point for the user interface. It had the 'live variable' feature which would update the values automatically whenever any change is made in the input field. That makes it very simple to work with the control parameters in this system. As it had all the features like buttons, spinners for number input, text input required for the project, it was considered to start building the user interface with GLUI.



Viscosity: 0.001
Smoke amount: 500
Gathering force: 0.033
Driving force: 0.01
Momentum: 0.007

[✓ Set text target]
Set image target

Target Text: UOB

Select Target Image (*.bmp):

example6
glmain.c
fontrender.cpp~
example3.cpp~
walk_bmp/
example2.cpp
glui-2.35.zip
example5.cpp

Figure 3.7: User interface built using GLUI

At this stage the OpenGL display function was already showing the smoke animation with a default value assigned to the control parameters. To attach a user interface with it,

- A GLUT window was created, which is the main window with the function glutCreateWindow()
- The OpenGL function for displaying smoke was attached with the glutDisplayFunc ()
- The focus was set to the main window in the glutIdleFunc ()
- A GLUI sub window was created with the GLUT main window with the function GLUI_Master.create_glui_subwindow ()
- The user interface input components were added to the GLUI subwindow.

GLUI_Spinner is used to set the values for vf, vg, vd, smoke amount and GLUI_EditText for text input. A GLUI file browser is added for selecting image. It worked well enough for setting the values for the control parameters. But problems started occurring when extra features were required to be added to make it more intuitive for the user. The GLUI components were very limited in terms of customization features. It was soon realized that it is not possible to meet the standards of a usable interface using GLUI.

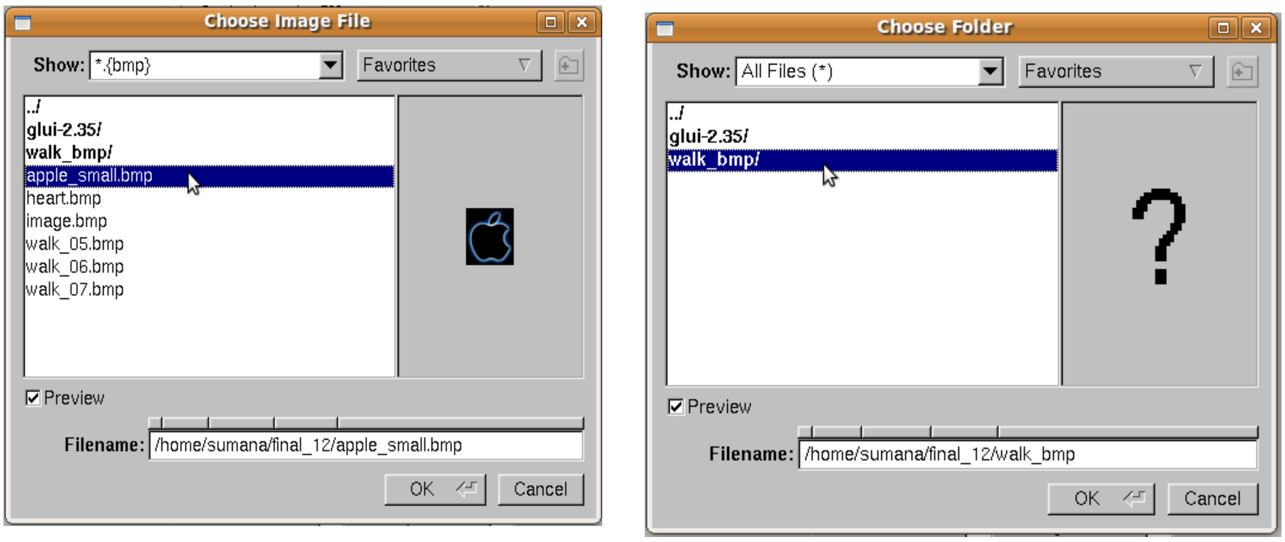
3.5.2 Using FLTK (Fast Light Toolkit) :

FLTK is another tool kit for building user interface in C++ that works with OpenGL. It is not built on GLUT like GLUI but it has got an inbuilt GLUT emulator. Compared to GLUI this comes with a wider range of features and it is quite flexible. The components can be positioned or resized exactly the way it is required. Because of the fact that all the images and widget layout used for the GUI component can be specified directly into the source code it opens up a larger scope of flexibility. On the other hand as every function and parameters that is going to be used for the GUI to work needs to be defined by the developer makes it a bit more complicated compared to GLUI. A very well written documentation for FLTK was found and the features seemed to be suitable for the project. The goal was to improve the performance of the system when it is evaluated.

An Fl_Window was created to hold the smoke display and the input fields on the right hand side. The whole layout is specified by exact pixel position. To show the OpenGL smoke display function into the FLTK GUI Fl_Gl_Window was used. A display class was declared that inherits Fl_Gl_Window with a method that draws the smoke display in OpenGL. Creating an object of the display class will create a Fl_Gl_Window in the specified position on the Fl_Window. For every frame of the animation the display object will be redrawn on the Fl_Gl_Window. Display object has properties for the control parameters that are set with any value specified by the user.

Improved GUI with FLTK:

FLTK allows a much more flexible and customisable file browser for the GLUI. As it is required to import only .bmp images for the image target allowing user to choose other type of file might be confusing. It was possible to specify the selection file type while opening the file browser window with FLTK. For loading the animation the file browser could be set to only point to a directory. This is much more sensible which loading a sequence of images for the animation target. And the browser window had much more user friendly navigation system where user could jump back any parent folders just by clicking one button.



Show: *.{bmp}
Filename: /home/sumana/final_12/apple_small.bmp

Show: All Files (*)
Filename: /home/sumana/final_12/walk_bmp

Figure 3.7: File chooser in the improved GUI built using FLTK

It was possible to display the image in the GUI (as shown in Figure 3.7) when user select an image with the file browser. This makes the user more sure of which file has been selected for target. When a directory is selected, the animation created by the images in the folder was displayed on the GUI, which makes user sure of the sequence that has been selected as target.

It was possible to enable or disable a Fl_Group as required. Which is very important for my GUI as the system had three types of options for user to set target. It would be really confusing for user if it was possible to set all type of targets at once. User would not know which target shape the smoke should form just by looking at the GUI. So disabling all the input fields except the one user wants to use made the GUI more intuitive.

With additional feature in FLTK, user can drag in the screen to place the position indicator anywhere on the display screen. User will be aware of the exact position on the display screen.

3.5.3 Implementing the control Features:

Control Parameters:
The GUI provides control over some parameters that directly effects how the smoke is going to behave while it is achieving the target shape. User can adjust the value of the parameters using the sliders. Fl_Value_Slider() is used to implement this feature. As user slides the value adjuster the current value will appear on the left side so the user knows the exact value the slider is set to. The adjusted values on the parameters works in real time for the smoke display. User can notice the effect of the parameter straight away on the display which makes it easy to figure out which value would work well for a specific requirement. The values are limited to a minimum and a maximum value range to avoid the smoke to behave unnaturally for any setting.

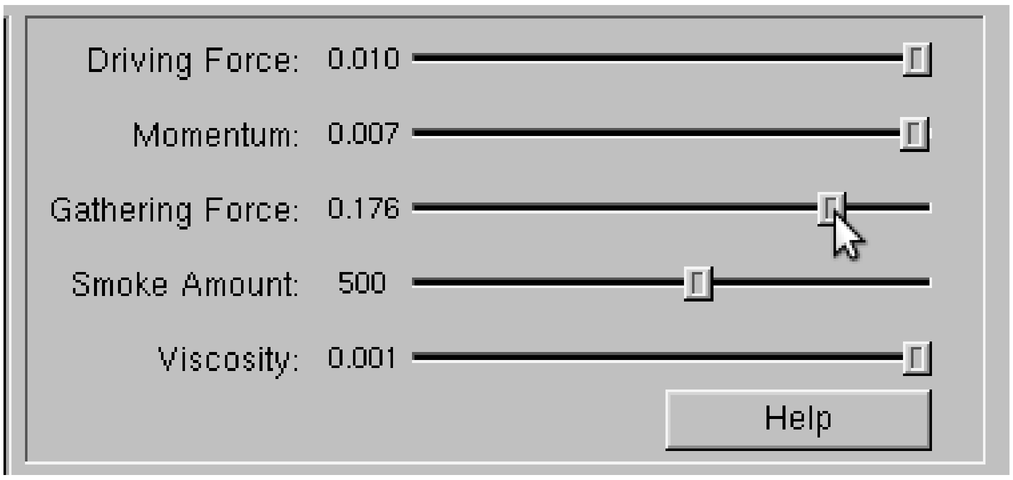

Driving Force: 0.010
Momentum: 0.007
Gathering Force: 0.176
Smoke Amount: 500
Viscosity: 0.001

Figure: 3.8: Control parameter sliders

Driving Force:
The slider for driving force will determine how strongly the driving force will act on the smoke. This slider sets the value for vf in the implementation, which is multiplied with the driving force in the equation. This affects the speed in which the smoke will flow an reach the target position. The slider value range is set to 0.5 to 0.005 to maintain the natural smoke behaviour.

Momentum:
The slider controlling this value acts on the momentum attenuation term in the implementation. The slider sets the value for vd, which is multiplied with the momentum attenuation term while adding

force. The slider adjusting the momentum value range is set to 0.5 to 0.005 to maintain the natural smoke behaviour. Because momentum attenuation is directly related to the driving force, it is some times required to be set to around the same value as vf for realistic smoke behaviour. But different settings might be required for generating a desired effect. The GUI set the value of vd nearly same as the value of vf by default. This gives the user a good starting point to adjust the parameters.

Gathering term:
The slider controlling this value acts on the gathering term added in the advection equation. Adjusting this value updates the value for vg in real time so that effect is viewable straight away. User would adjust the value higher if the target shape needs to be achieved very closely. The slider can be adjusted to a value in the range of 0.1 to 0.01.

Viscosity:
This slider controls the value of the viscosity of the smoke. This adjusts the 'visc' term used in the filter that is multiplied to the velocity in the Fourier domain for the mass conservation step of the fluid solver. User can control the effect of viscosity on the smoke adjusting the slider. The slider range is set to the range of 0.01 to 0.001.

Smoke Amount:
The smoke amount can be adjusted by the user at the starting of the animation. This might be required if the user has specifies a target of a largely varying size. In the input processing step the smoke amount is divided by the size of the target shape. So for a larger target shape the smoke might become very thin as the shape is formed. Using this slider value this can be adjusted depending on the size of the target shape or the font size user has chosen. The slider range is set to 800 to 200.

Help Button :
Clicking on the 'Help button will bring up a text display window. The effects of all the control parameters on the animation is stated here so that user can understand how setting the parameters will change the smoke behaviour. Useful tips are also provided to achieve certain effects.

Target (text / image shape)-

Setting the target sets the target density and the smoke will be animated in a natural smoke like behaviours to form that target shape. The GUI is designed to set three types of target from the user.

- Set text as target
- Set single image as target
- Set sequence of image as target

User gets three radio buttons to select which type of target is desired to be set for the animation. Fl_Group is used to group all three radio buttons so that only one can be selected at any time. All the component related to the three types of target are also grouped in three separate Fl_Groups. Each groups is associated with the corresponding radio button. When one radio button is selected the callback function of the radio button enables the group associated with it and disables other two groups.

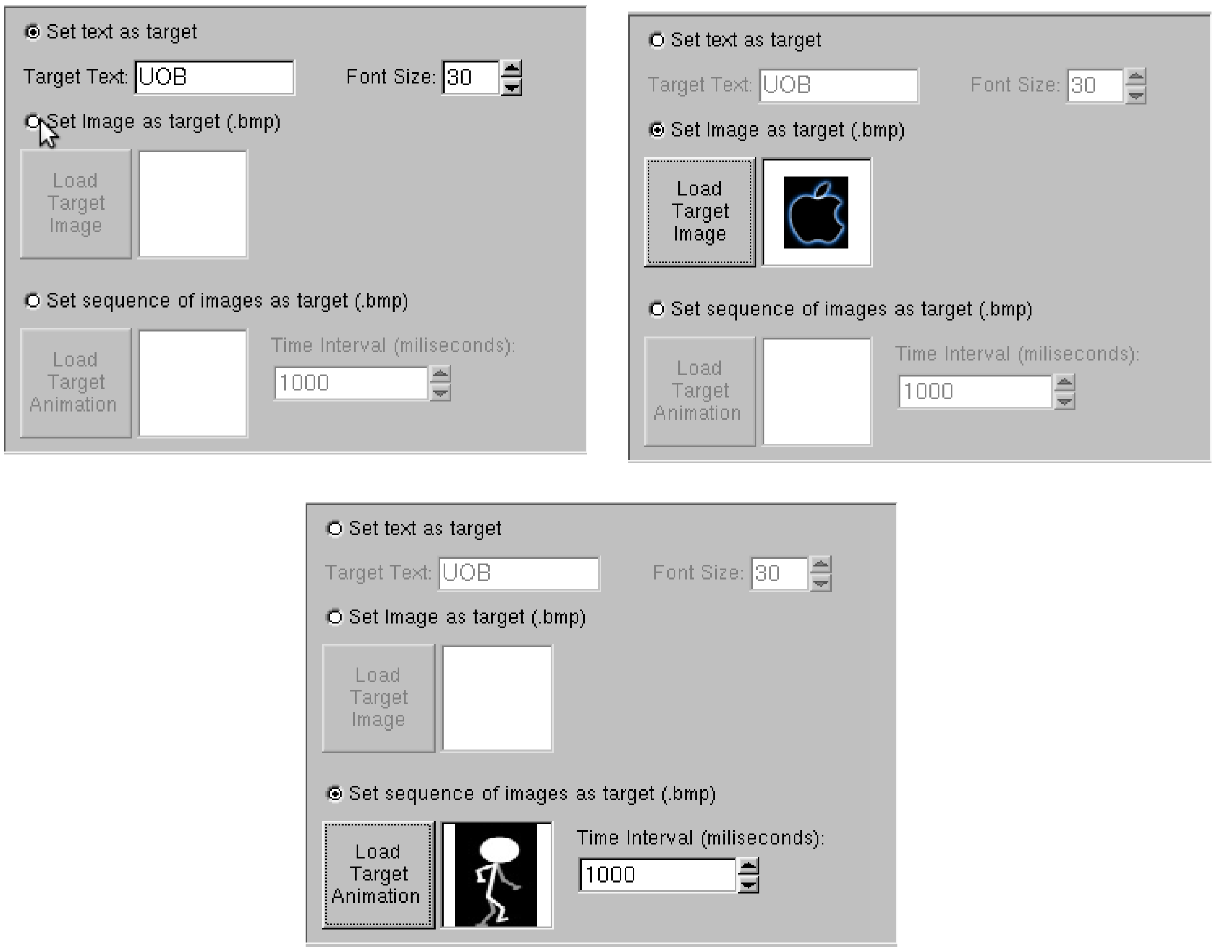

Set text as target
Target Text: UOB
Font Size: 30
Set Image as target (.bmp)
Load Target Image
Set sequence of images as target (.bmp)
Load Target Animation
Time Interval (miliseconds):
1000

Figure 3.9: Input option allowing only one type of input to be chosen at once, disabling the other options accept the selected one.

For the text target input Fl_Input has been used. A suitable font is coded inside the system but it can easily be chosen from the GUI. There is an option for choosing a font size for the text. The text box has been set to a callback function to update the target text with the input text. This call back function will only be called when the 'Show' button is pressed. This stops the callback function to be called every time a change is made in the text input box. The call back function calls for the input processing for text mentioned earlier with the input text and font size.

For image input a button is created using Fl_Button. Right beside the button a Fl_Box is placed to show the selected image preview. The callback function for the button created a file chooser with Fl_File_Chooser(). The file chooser opens a pop-up window with options set to select only one .bmp file. The callback function waits until the file chooser is closed. If user has chosen a file the file chooser will return file count '1' and '0' otherwise. If the file count is more than zero the callback function gets chosen file name form Fl_File_Chooser->value(). Fl_Shared_Image library is used to load an image into the GUI. Calling the function Fl_Shared_Image::get(char*) with the file name returns a Fl_Shared_Image image. This is copied as Fl_Image and resized to fit into the Fl_box for selected image preview.

- Fl_Button Callback:
    - Create Fl_File_Chooser – Single image, file type .bmp

- Show file chooser
- Wait if file Chooser is visible
- If file count is > 0
    - Get file name
    - Load image with Fl_Shared_Image: get()
    - Resize image
    - Set image on Fl_Box for preview
- Else Quit

For the image preview of the images sequence is implemented almost the same way for single image. In this case the file browser is created with option set to allow to select a directory only. When a directory is selected it searches for sequenced image files. If there is any then a POSIX thread is created to preview all the images one by one. The thread sleeps for a short time before loading the next image. The thread exits when it reaches the end of the sequence. Displaying the images sequentially one after another displays the selected animation in the Fl_Box. This indicates the sequence of the images the smoke will be following when play button is pressed.

Positioning Target:

User gets the option to set the target position to any coordinate on the screen. The smoke will reach the specified position to form the shape of the target. User can specify the target position on the screen in two ways

- Dragging the position indicator on the screen with mouse
- Input the x, y coordinate of a position on the screen in the number input field. (0,0) means the top left corner of the screen.

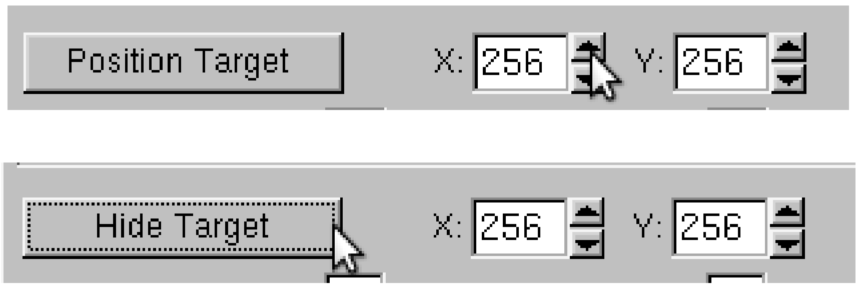
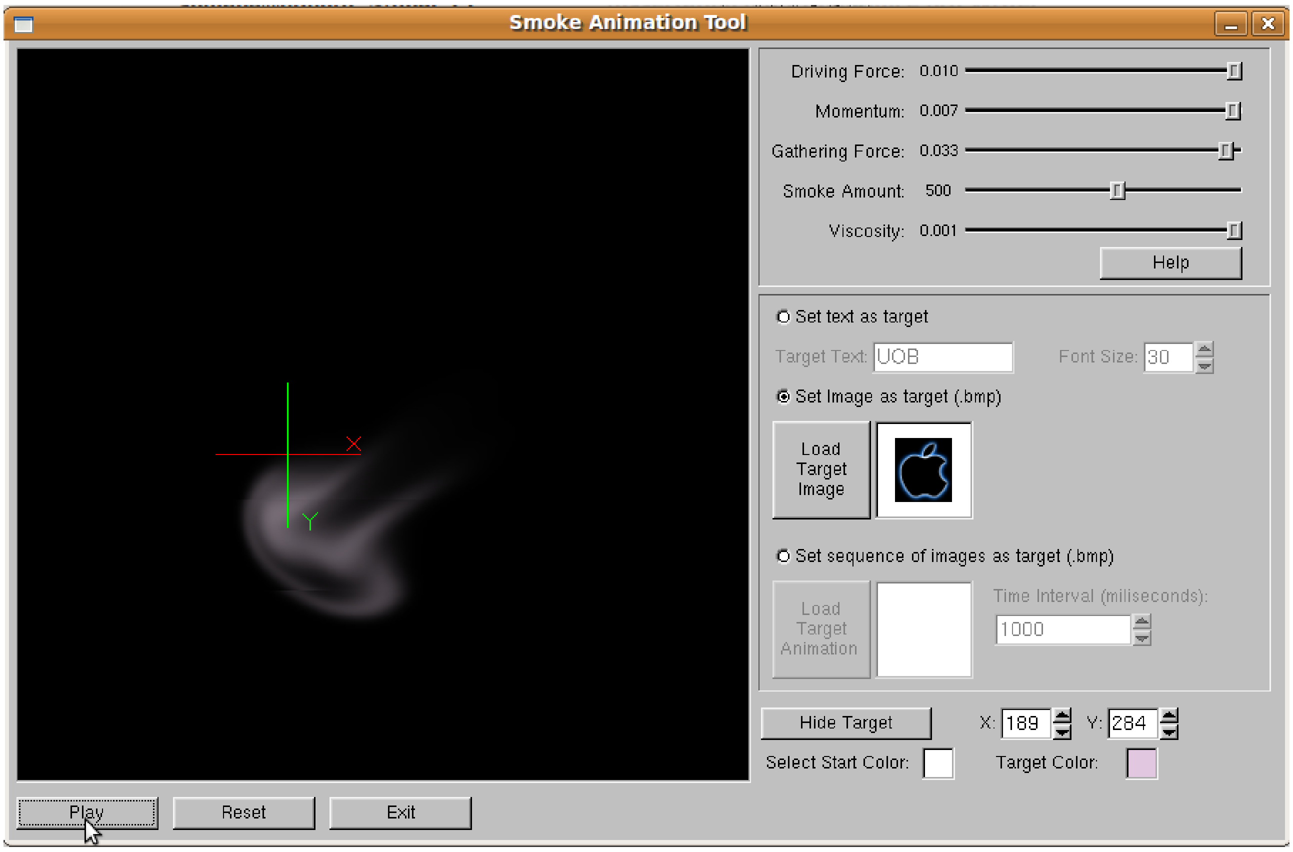


Figure 3.10: Target positioning option on the user interface with toggle button and coordinate input fields

The target positioning component is implemented with a toggle button. Clicking this button with place an indicator on the screen showing where the current position of the target is. It is set to be centre of the screen by default. The callback function for the toggle button will draw the indicator in the current position on the screen and hide it if it was already showing. The mouse event is handled by the handle function of the display class mentioned earlier in 3.3. When the mouse is dragged on the screen the mouse position returned by Fl::event_x() and Fl::event_y(). These positions are the pixel position of the display screen, the top left being (0,0). This value is then converted to the OpenGL coordinate system considering the top-left to be (1,1) and the bottom-right to be (-1, -1).

The current position of the position indicator is then updated by this converted value. The coordinate input fields are then also updated by the mouse position so that user knows the exact pixel position on the display screen the indicator is being placed.



Driving Force: 0.010
Momentum: 0.007
Gathering Force: 0.033
Smoke Amount: 500
Viscosity: 0.001
○ Set text as target
Target Text: UOB Font Size: 30
● Set Image as target (.bmp)
○ Set sequence of images as target (.bmp)
Time Interval (miliseconds): 1000
X: 189 Y: 284
Select Start Color: ⬜ Target Color: 🟪

Figure 3.11: Position indicator on screen. The coordinates on the screen are displaying on the coordinate input fields

When an x, y coordinate is input in the input field the callback function for the two fields are again converted into the OpenGL coordinate system considering the top-left to be (1,1) and the bottom-right to be (-1, -1).

Key framing:

User can produce a key framed smoke animation by setting a sequence of images as target. The time interval between the images are set from an input spinner. The selected image will be set as target after the selected time interval. Depending on the time interval smoke can form the specified key frame exactly or just attempt to form the shape if another target is set before letting it to form one target shape completely. The speed of the animation can be adjusted with the time interval for desired key frame control. A default setting can be made to see how closely the smoke density has achieved the target shape and wait until one key-framed target shape has been completed before moving on to the next key-frame. All these settings allow user to make different kind of key-frame animation, either forming the key-frames exactly or just make the smoke flow through the key-framed shapes. The target specified for each key-frame and the time interval is stored in an array as user selects the inputs. When animation is played, a thread is created which sets the target shapes as target densities one after another with the specified time delay for each key-frame.

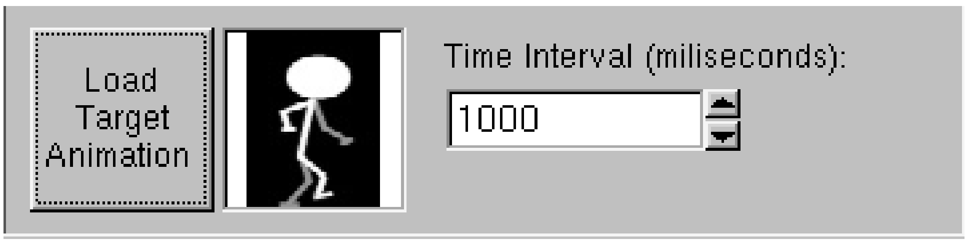


Figure 3.12: Option for loading a sequence of images and specifying time interval

Colour chooser:

User can specify colour for the starting and the target shape of the smoke. The starting colour will gradually change into the target colour when the animation is played. The default starting and target colour is set to white. Two Fl_Box shows the two colours that are selected. Clicking on the colour boxes with open a colour chooser using fl_color_chooser() and will return RGB values for the selected colour. The RGB value is then converted into Fl_Color to set the colour of the corresponding boxes. When the animation starts for each time step a new colour is generated averaging the two colours specified by the colour chooser. The average is weighted towards the staring colour at the start of the animation and gradually weights towards the target colour as the animation proceeds. For each frame the smoke is rendered using the averaged colour. In the animation it seems like the colour is transforming into target colour as it is forming target shape.

- Convert Starting RGB colour into Fl_Color
- Convert Target RGB colour into Fl_Color
- Weight = 1
- For Each Frame:
  - New_Color = Weight * Source_Color + (1- Weight) * Target_Color
  - Render with New_Color
  - If Weight > 0
    - Decrease Weight

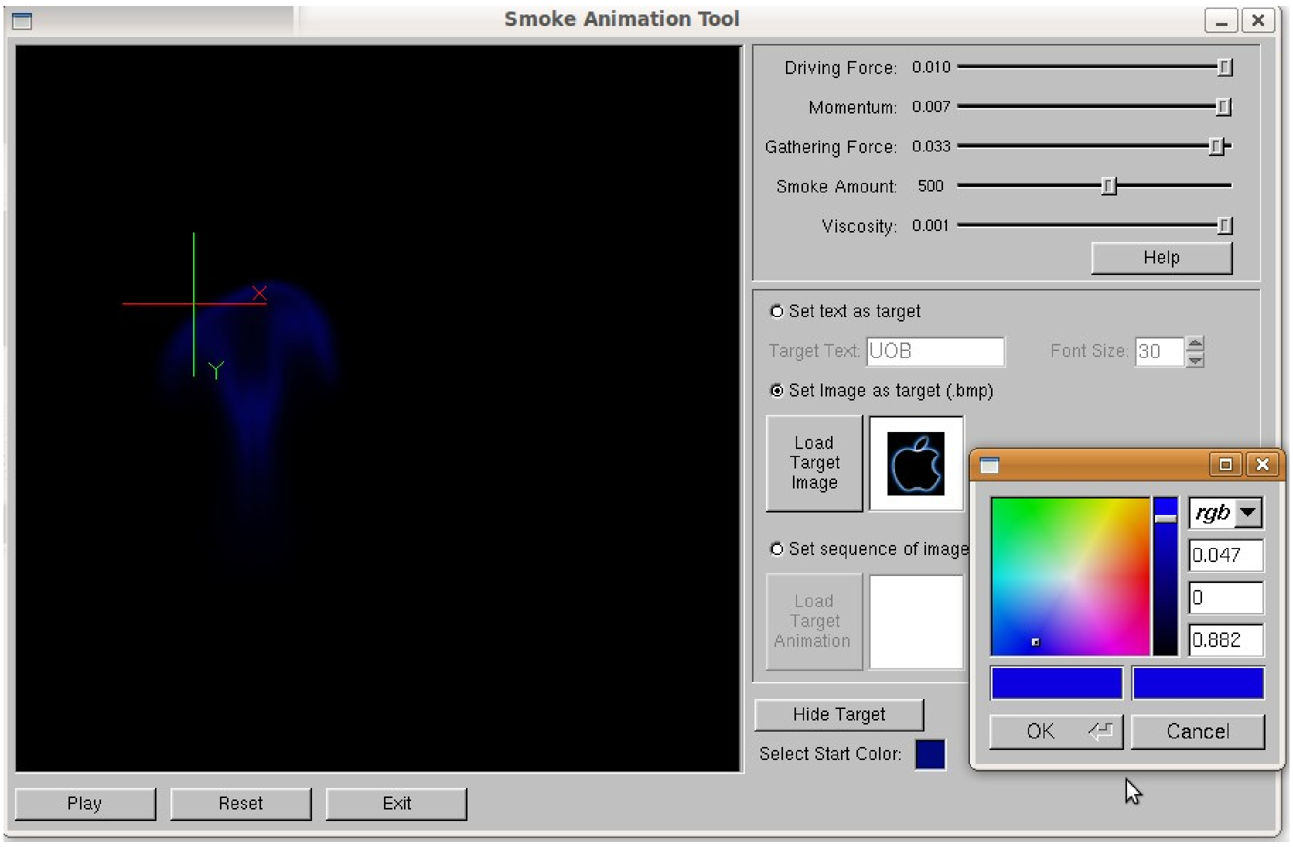


Figure 3.13: Color chooser pop-up window, allowing user to select color for smoke animation.

Play button:
The callback function for the play button takes action depending on which option is selected by the user. It will perform the following steps,

- Play Button CallBack:
  - Set all the values stored into the target density array into '0'
  - If text as target selected
    - Perform input processing for text
  - If Image as target selected
    - Perform input processing for single image
  - If sequence of images as target selected
    - Create a thread for animation
  - Apply Gaussian blur on the target density

Reset Button:
The callback function for the reset button will set all the parameter into their default initial value. It will set all the arrays storing the current density and target density to the initial state and the velocity array to zero. User can undo all the changes made to the settings and the display.

3.6 Summary

This chapter presented the technical aspects of the implementation of the system and the theory that works behind the implementation. The first step was to solve the equations involved in fluid simulation and implement them with additional control capability. The design and implementation choices made for the system are mentioned in terms of suitability for the project. The chapter also reflects the complications that needed to be handled to implement the fluid simulation and the control features. The display and the user interface is built to make it into a usable animation tool. The user interface is implemented with a number features that allow user to make interesting animation with smoke. At the end the system completes with a display screen attached with a user interface, through which user can specify inputs, set parameters then play the animation and be creative with it.

4 Experimental Results

The system has been experimented with many combinations of inputs and parameter settings. The produced animation is presented with snap shots in this chapter.

4.1 Text as Target Shape

User can set a specific text as target for the smoke animation. The snap shots in figure 4.1 is from the animation produced with 'UOB' and 'Hello' which are set as target text. The animation starts with a circle filled with smoke shown in Snap 1. This is the default initial state for starting frame. Snap 2 to Snap 6 shows the smoke animated to form the text 'UOB'. The parameters set for this animations are given below.

> Driving Force : 0.01 , Momentum : 0.007, Gathering force : 0.05, Smoke amount : 500, Viscosity :0.001

It can be seen that the smoke is animated in a smooth away without much disruption. For the animation the gathering term 0.05 was enough to form the text 'UOB' quite neatly. After the formation of the first target next target is set to 'Hello'. Snap 7 to snap12 shows the smoke animated from 'UOB' to 'Hello'. The Gathering force is now set to 0.03. For the lower gathering term, there is slightly more stray smoke visible around the target shape.

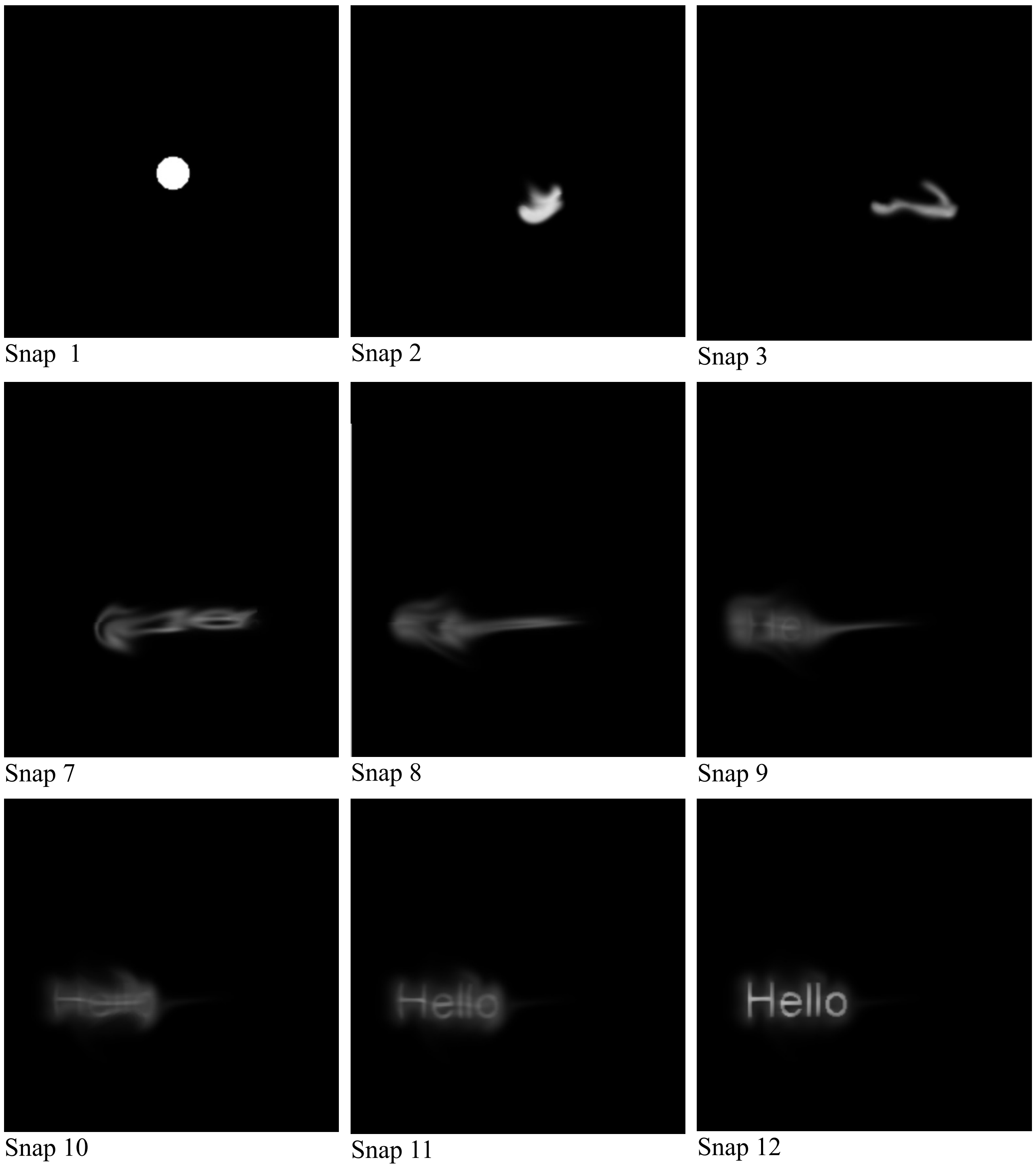


Figure 4.1: Snap shots of smoke animation produced with 'UOB' and 'Hello' set as target text

4.2 Single Image as Target Shape

This section shows the animation with an image set as target shape. A bmp image of the apple logo is set as target image. The snap shots in figure 4.2 and figure 4.3 shows the smoke animation with the same target image. Smoke is behaving differently while forming the same target shape because the parameters are set with different values.

Parameter setting for figure 4.2 is given below:

> Driving Force : 0.05 , Momentum : 0.01, Gathering force : 0.05, Smoke amount : 500, Viscosity :0.003

Parameter setting for figure 4.3 is given below:

> Driving Force : 0.25 , Momentum : 0.15, Gathering force : 0.2, Smoke amount : 500, Viscosity :0.001

The smoke animation presented in figure 4.2 is smooth and with less disruption. The target shape is formed in a more organised manner. In figure 4.3 the same animation shows a much more chaotic behaviour. The smoke flow show swirly movement while achieving the target shape. The higher driving force and relatively smaller momentum attenuation is causing this effect. The viscosity is also set to be lower than the previous animation. The smoke is being influenced by a higher driving force and the lower momentum attenuation is not being able to balance it out. This is causing the smoke to behave more chaotic. For this experiment the driving force and momentum has been set carefully so that the smoke does not go out of control. If the momentum attenuation was set to a lower value the smoke would go too much chaotic causing it not being able to form the target shape. In figure 4.3 the target shape is not as smooth as in figure 4.2. Bringing the momentum a little bit higher at the end could help the situation, if users requirement is a chaotic smoke flow and then forming a neat target shape at the end.

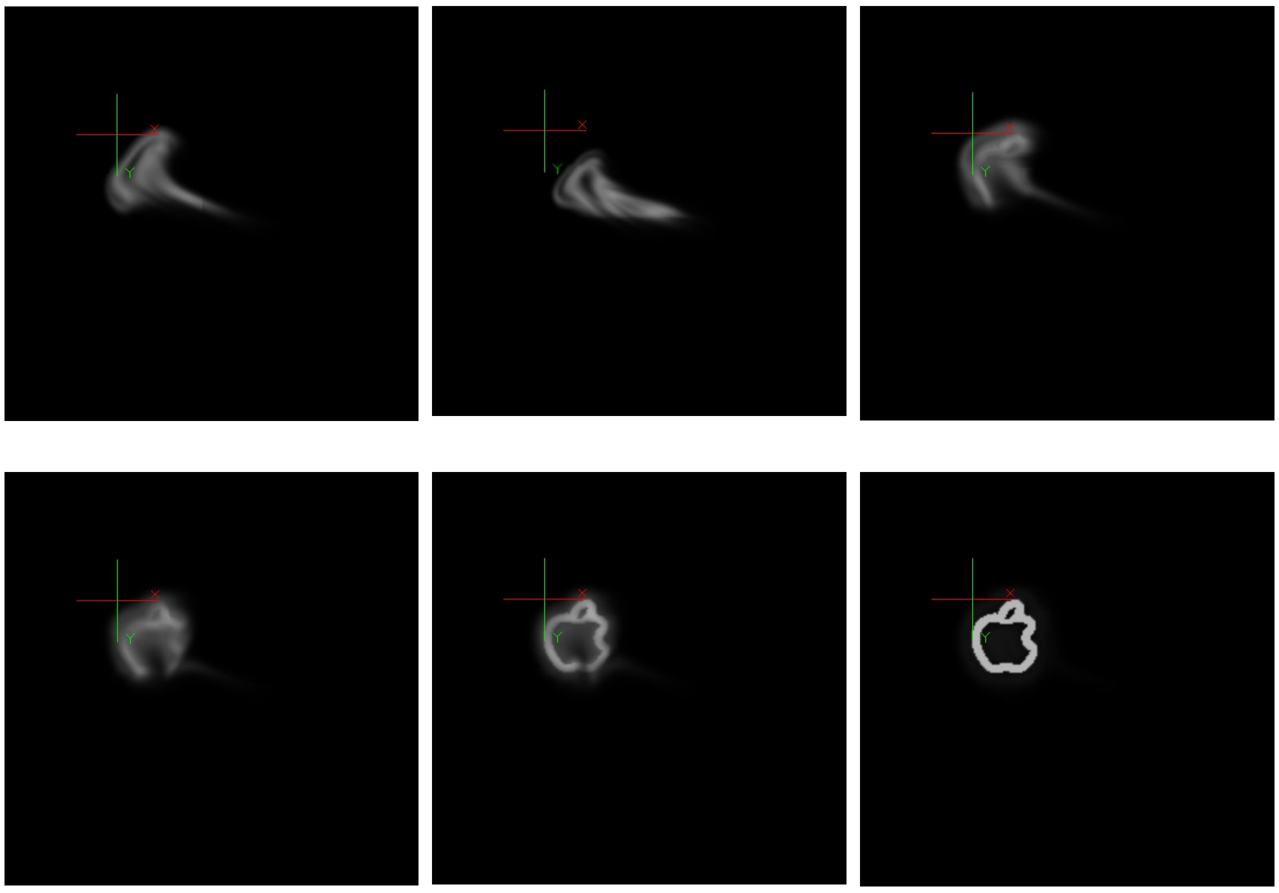


Figure 4.2: Smoke animation showing stable flow compared to figure 4.3

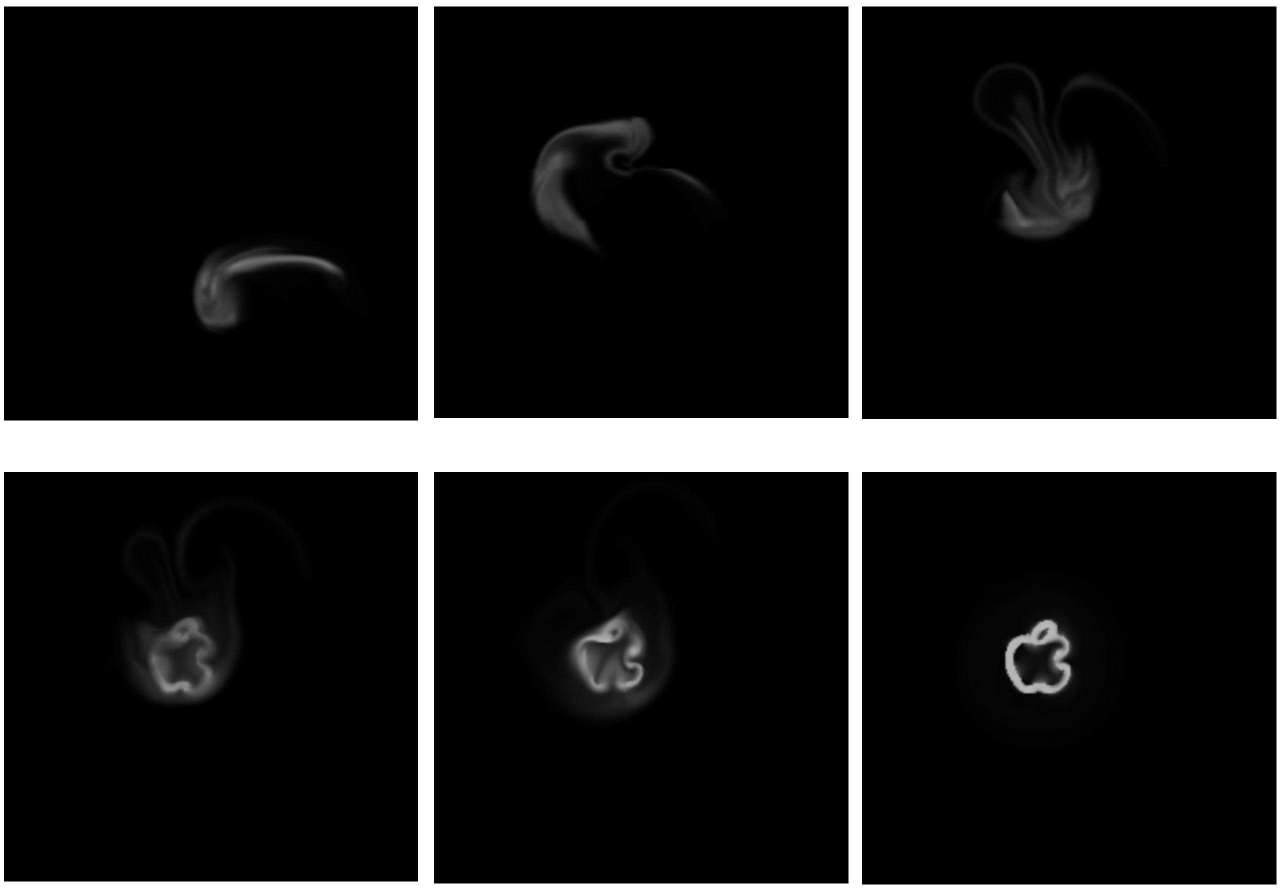


Figure 4.3: Smoke animation showing chaotic behaviour while forming target image shape

4.3 Sequence of Images as Target Shape

This section shows the smoke animation with a sequence of images set as target shape. This illustrates the key-framing feature of the system. Figure 4.4 shows the sequence of images used for this experiment. The sequence of images displays a simple walk cycle animation of a stick-man. Using these images as target shapes generates a walking smoke animation as if smoke is following the stick man. Figure 4.5 show snap shots from the animation generated by the walk cycle.

Parameter setting for Figure 4.5
Driving Force : 0.35 , Momentum : 0.30, Gathering force : 0.25, Smoke amount : 700, Viscosity : 0.003, Time interval: 1000 milliseconds
The resulting smoke animation is slightly disrupted but it is forming the target shapes within 1000 milliseconds. The smoke amount is set to a larger value because the target shapes needed more smoke to be formed properly. Less smoke would cause lighter figures which would have been difficult to notice. The driving force and momentum attenuation is set to equally higher value to adjust the speed of the smoke to form the target shape without being too much chaotic. The resulting animation is a walking smoke, a little bit chaotic but the stick man figure is noticeable.

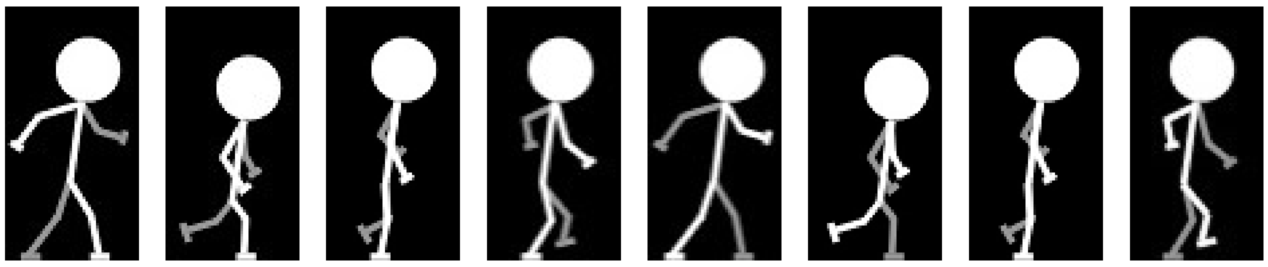


Figure 4.4: Sequence of images used for walk cycle

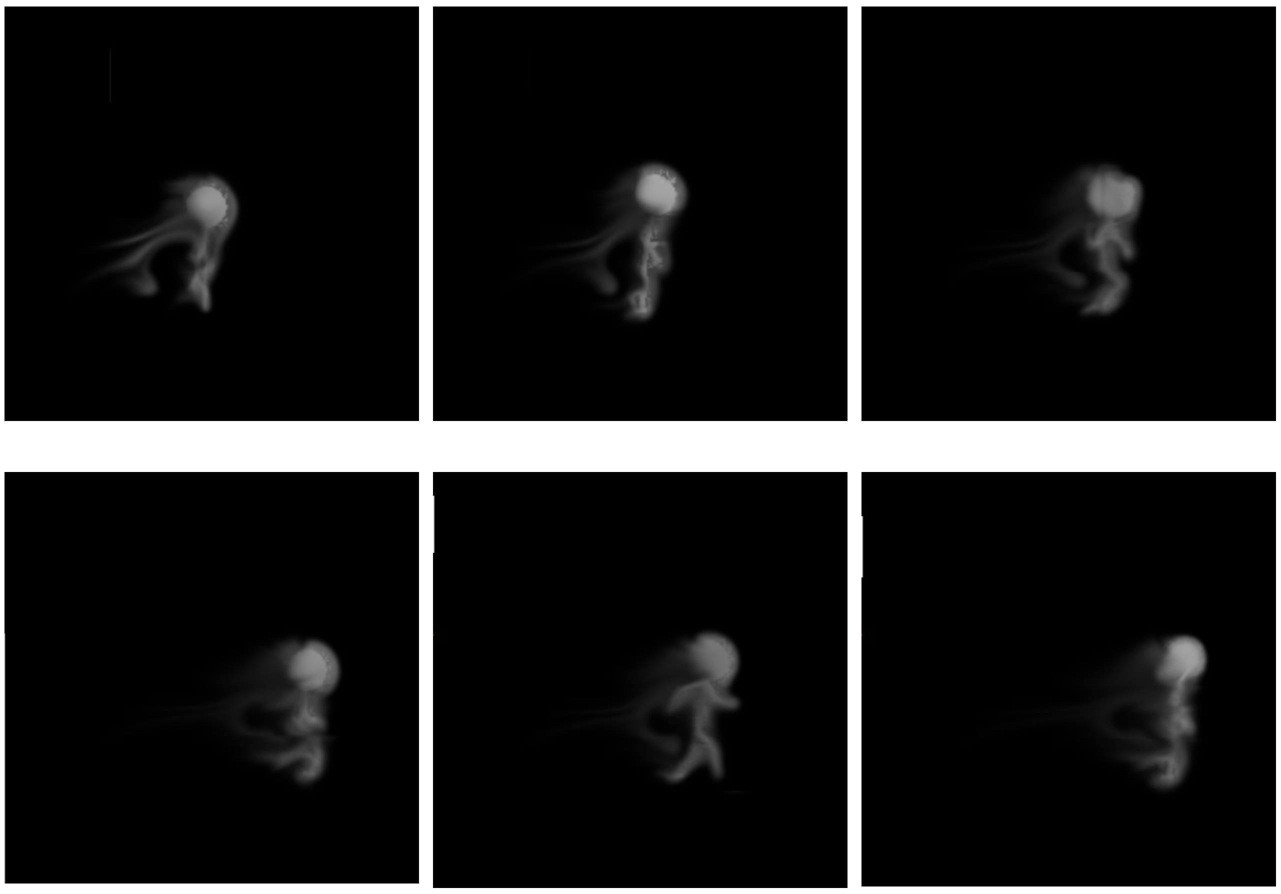
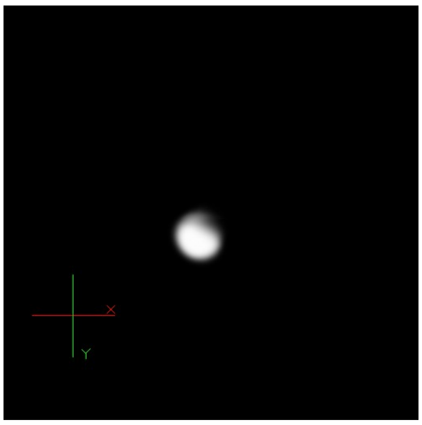


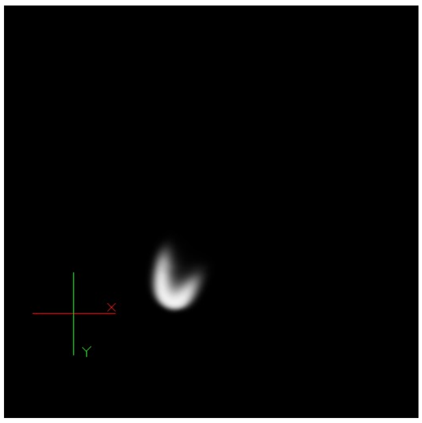
Figure 4.5: Smoke animation following stick-man walk cycle

4.4 Positioning Target

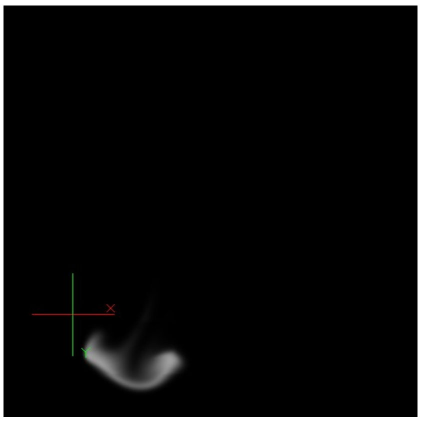
This section illustrates the effect of positioning target in different coordinates on the display screen. The position indicator is shown by a vertical green line and a horizontal red line representing y-axis and x-axis. The centre point indicates the top left point of the target shape. The top left point of the screen is the (0,0) point of the coordinate system of the screen. The snap shots in figure 4.6 has been generated by setting the target as 'UOB' and setting the target position in different coordinates. The resulting animation shows the smoke flowing to the specified coordinates following the position indicator and trying to form the target shape. The position indicator has been set to (100, 400), (155, 218), (334, 140), (360, 400) coordinates on the screen. The starting position is (225, 225). Smoke starts animating from (225, 225) and tries to reach (100, 400) in snap1 to snap4. Then redirected in snap5 as the position is changed to (155, 218). Smoke follows the position indicator form snap5 to snap8 and reaches (334, 140), the target shape starts to appear here. Then from snap10 the driving force is increased and gathering term and viscosity are set to a lower value, and the smoke suddenly drops to (360, 400) coordinate with a swirly flow and gradually form the target shape.



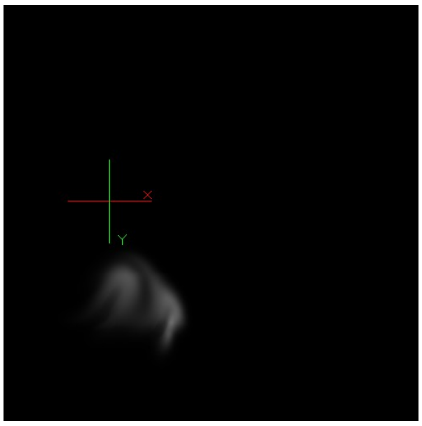
Snap 1



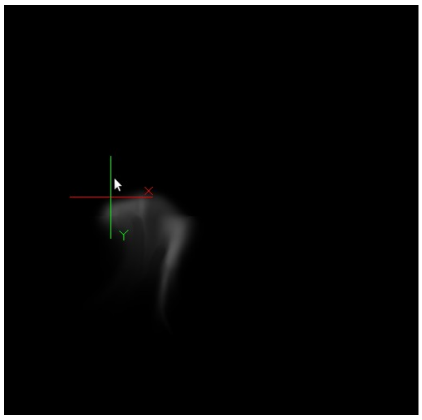
Snap 2



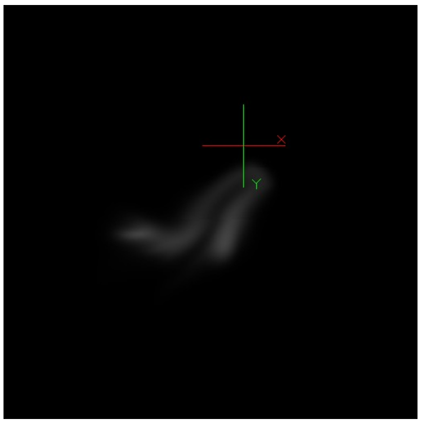
Snap 3



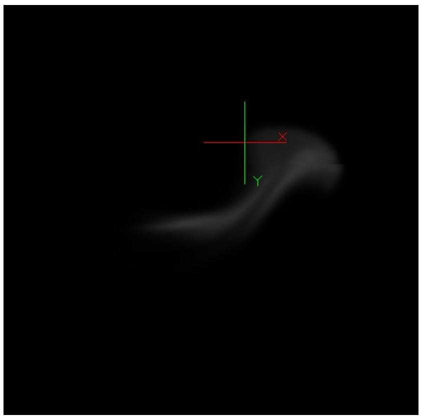
Snap 4



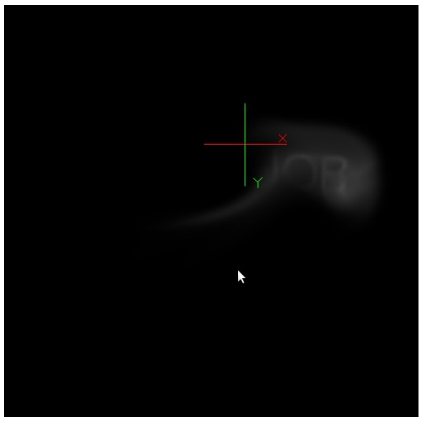
Snap 5



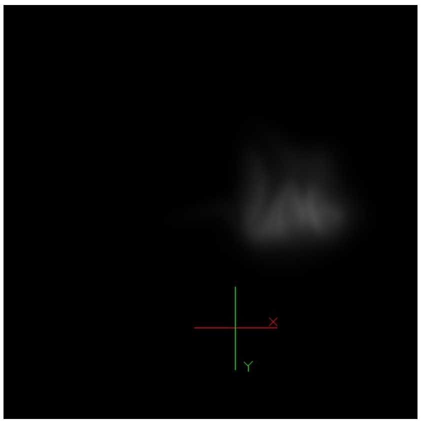
Snap 6



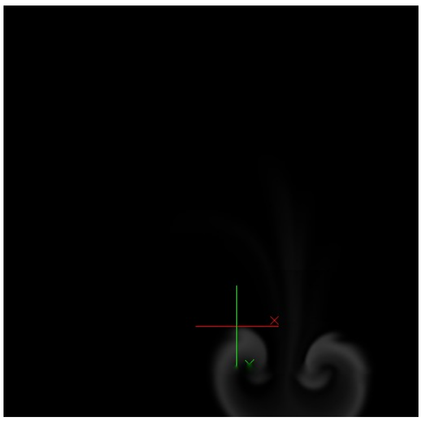
Snap 7



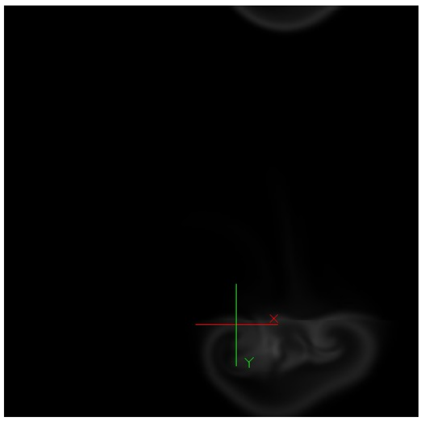
Snap 8



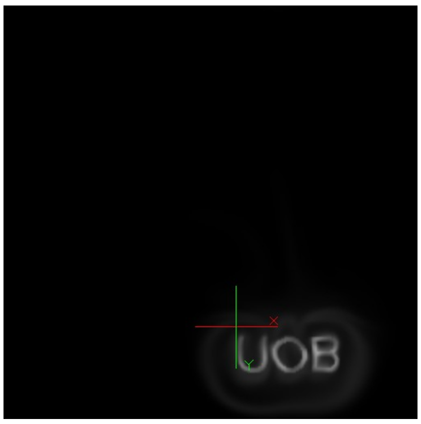
Snap 9



Snap 10



Snap 11

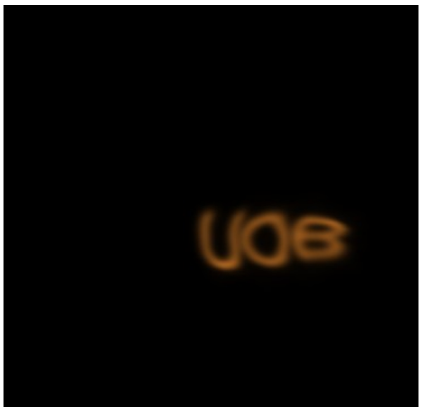


Snap 12

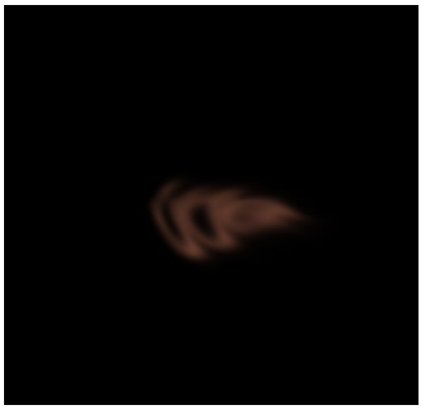
Figure 4.6: Smoke animation following position indicator on screen, from snap1 to snap 12

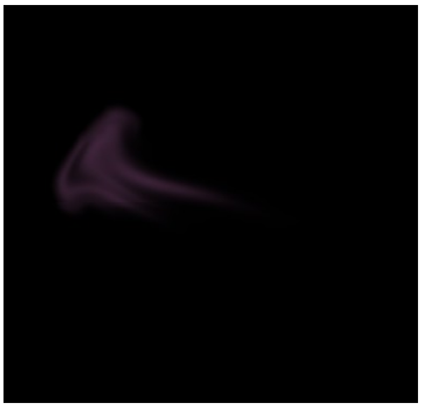
4.5 Selecting colour for target:

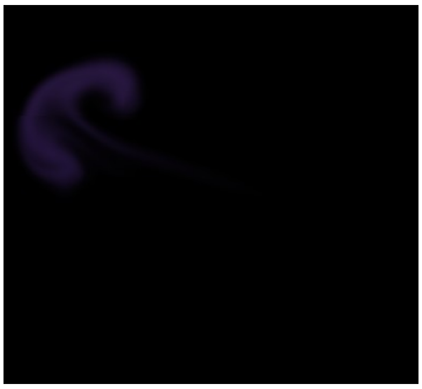
Figure 4.7 shows the smoke animation with setting different colours for starting and target shape. The starting shape is 'UOB' set with yellow colour and the target shape is the apple logo set with blue colour. As the smoke starts forming the target shape the colour of the smoke gradually changes from yellow to blue.

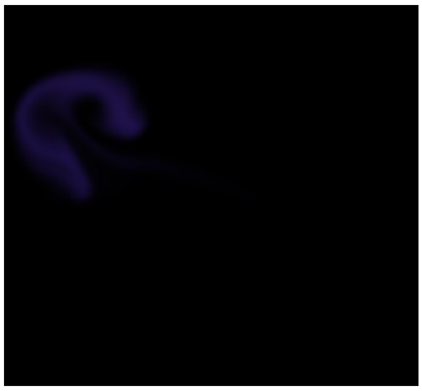


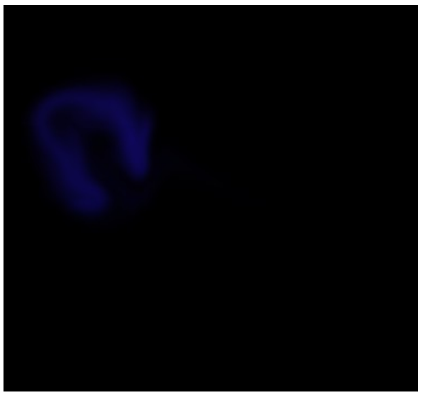
Snap1


Snap2


Snap3


Snap4


Snap5


Snap6

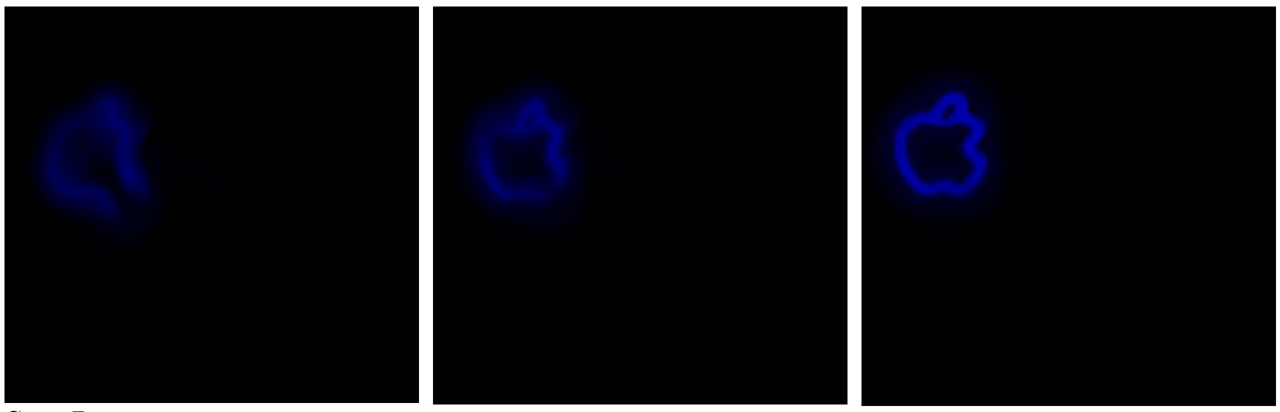


Figure 4.7: Smoke animation with setting different colors for starting and target shape from snap1 to snap9

4.5 Observations

From the experiments and resulting smoke animations the effect of driving force and the momentum attenuation can be observed quite clearly. The momentum attenuation needs to be set carefully close to the driving force term to prevent the smoke to get out of control. Some times chaotic flow of smoke can be required for certain purposes, less viscosity can be enough to achieve that in some situations.

The smoke amount is necessary to be adjusted according to the size of the target shape or the thickness required for the smoke animation. If the smoke amount is set too low the target shape sometimes can be too thin to be visible.

The gathering term helps the smoke to achieve target shape but if the value is set too high then sometimes the target shape starts forming too early even before enough smoke can flow and reach the target position. This might look unrealistic. The gathering needs to be set a value considering that the driving force is set high enough to drive the smoke to target position before it starts gathering the small amount of smoke that reached the target position.

The time interval setting of the key-frame animation needs to be set according to the speed of the smoke flow. If it is too high smoke will reach target shape and wait in that shape for next target shape. This might not be the desired effect in some cases. On the other hand the target shapes might fail to form if the time interval is too low.

Because the system provides user with a lot of flexibility in terms of controlling the smoke behaviour many types of animation is possible to produce. At the same time is it easy to get carried away with the parameters and get undesired effect. It is very common fact with any kind of animation tool. It is required to leans a few tips and practice for a while before getting started with the tool.

5 Evaluation

5.1 Purpose and objective

To create a successful software, it is necessary to test it with the end users. This provides the user acceptance and any usability issues associated with the system. By identifying the usability issues, the developers can resolve these issues and can design a greater user experience in order to use the system.

After developing the first version of the smoke animation tool, it was critical to conduct a usability testing to see how usable the software was to create a smoke animation and manipulate the parameters based on user requirements.

A number of usability testing methodology is available in order to identify usability issues of a system. For example, heuristic evaluation, cognitive walk-through, eye-tracking and so forth. However, due to time and resource limitations, co-operative evaluation (Monk et al., 1993) was chosen. This method allows the design team to test the tool on users.

The purpose of the usability evaluation was to:

- Identify any usability issues associated with it;
- Assess users' satisfaction level on the quality of the animation;

5.2 Methodology

As previously mentioned, the co-operative evaluation allows the design team to test the tool on its users and identify any usability issues associated with it. This method uses the 'think-aloud' protocol, which encourages users to think out loud when performing a set of tasks. During this time, the facilitator asks several questions to verbalise user's thoughts and identify any usability issues associated with the system. For example, 'Why did you click on this button?, How meaningful this label is to you? What would you do next in order to change the location of the animation?' and so forth.

The usability testing session was divided into three parts:

- Introduction to the test session;
- Ask the users to complete three tasks;
- Ask users to rate the tool based on their experience of using it;

5.3 Introduction to the test session

At the beginning of the test session, test participants were provided with a form to sign in, which specifies that their personal details will not be disclosed to anyone. They were notified that anything they would say during the test session would only be used for the research purposes only.

Once they were happy and agreed to take part in the research (signed the consent form), they were thanked and asked to take a sit in front of the computer for a formal introduction to the test session.

During the formal introduction, they were told that the purpose of the test session was to test a tool that can be used to generate smoke animation. They were notified about the following practicalities:

- The time it would take to complete the test session;
- They can leave at any time during the session if they wish to do so;
- The location of the fire exists and what to do in case of a fire alarm;
- The location of the ladies/gents room if required.

In order to introduce with the testing session, the test participants were told that:

- They were going to look at a tool that can generate animation;
- The purpose of the session was to test the tool not them;
- They need to provide their honest opinions and feedbacks about the different aspects of the tool;
- They need to think out loud even if it seems obvious;
- They need to perform some specific tasks, which were already been set.

This helped the test participants to understand the research session. After the introduction to the test session, test participants were introduced to the tool.

Initially, they were asked whether they can understand what this tool was for. They were shown a demo on the tool and a few tips on how to use the system.   They were given a short while to play with the tool and get familiar with it before asking them to perform a set of tasks using this tool.

5.4 Tasks to test the tool

The test participants were asked to perform the following tasks using the tool:

1. Create a smoke animation of your name using this tool;
2. Create a smoke animation of a given image;
3. Create a smoke animation starting with a given text forming into a given image.
4. Create a smoke animation of an animation with a given folder containing a number of sequential images.
5. Create a smoke animation that starts with a chaotic flow and calms down as the smoke forms a shape.

During each of these tasks, users were asked about the different elements of the tool and to change the parameters to control the smoke,. At the end of each tasks they were asked to rate how easy or difficult it was to perform these tasks on a scale of one to five where five was very easy and one was very difficult. To evaluate the quality of the animation they were asked to rate the tool on how realistic the smoke simulation is looking with proper adjustment of the control parameters on a scale of 1 to 5 where 1 is very unrealistic and 5 is very realistic.

The full script with tasks with additional questions is provided in Appendix A.

5.5 Overall experience

After completing all the tasks, users were asked to provide their overall impressions of using this tool. They were asked whether they would use this system again if they require to create a smoke animation. Additionally they were asked whether they would recommend this tool to anyone in order to generate this type of animation.

5.6 Apparatus

The test sessions were run in the computer lab. A laptop was used to run the tool during each of the test sessions. In order to record the the usability issues and user ratings, pen and paper were used.

5.7 Test participants

Ten MSc. students (four of them were male and six of them were female) of the University of Bristol agreed to participate in this research. Their age ranged from 23 to 30. All of them were Advanced Computer Science students and were familiar with computer generated animation.

5.8 Summary of the usability testing

Users were able to complete all the tasks more or less successfully. Some hints and tips were provided if someone struggled to complete any task. The ratings were noted for each tasks during the testing session. The ratings for ease of use and quality of animation were averaged for each task. The average of the rating based on the realisticness of the smoke animation is presented in figure 5.1. The data is based on the improved user interface. The average of the rating based on the ease of use on the scale of one to five is shown in table 5.1. This data is the obtained from evaluation the improved version of the user interface.

Task1	Task2	Task3	Task4	Task5
4.8	4.1	4.5	3.9	4.5

Table 5.1 : The average of the rating for each task based on the ease of use on the scale of one to five, five being very easy and one being very difficult.

As an overall experience 90% of the users expressed their interest in using this tool for making smoke animation in future. They were impressed with the quality of the smoke animation and most of them rated around 4 for most of the tasks. The average of the rating in terms of realisticness came about 4.35 out of 5. Most of the user got real interest in the key framing feature of the system. They were really impressed with the walking smoke animation they could make with this feature. The color changing option added more interest to it.

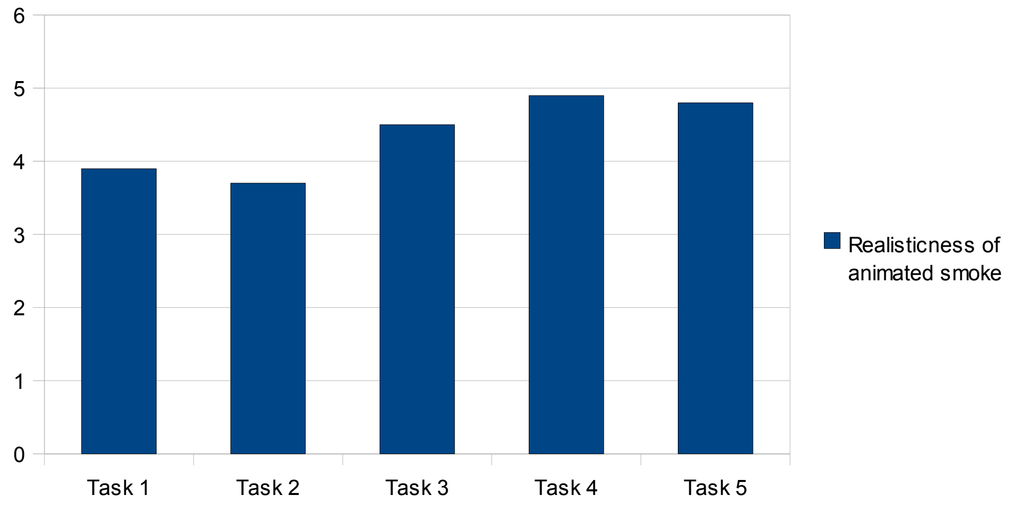


Figure 5.1 : Chart showing the average rating on realisticness of the generated smoke using the tool for each task, on a scale of one to five. Five being very realistic and one being very unrealistic

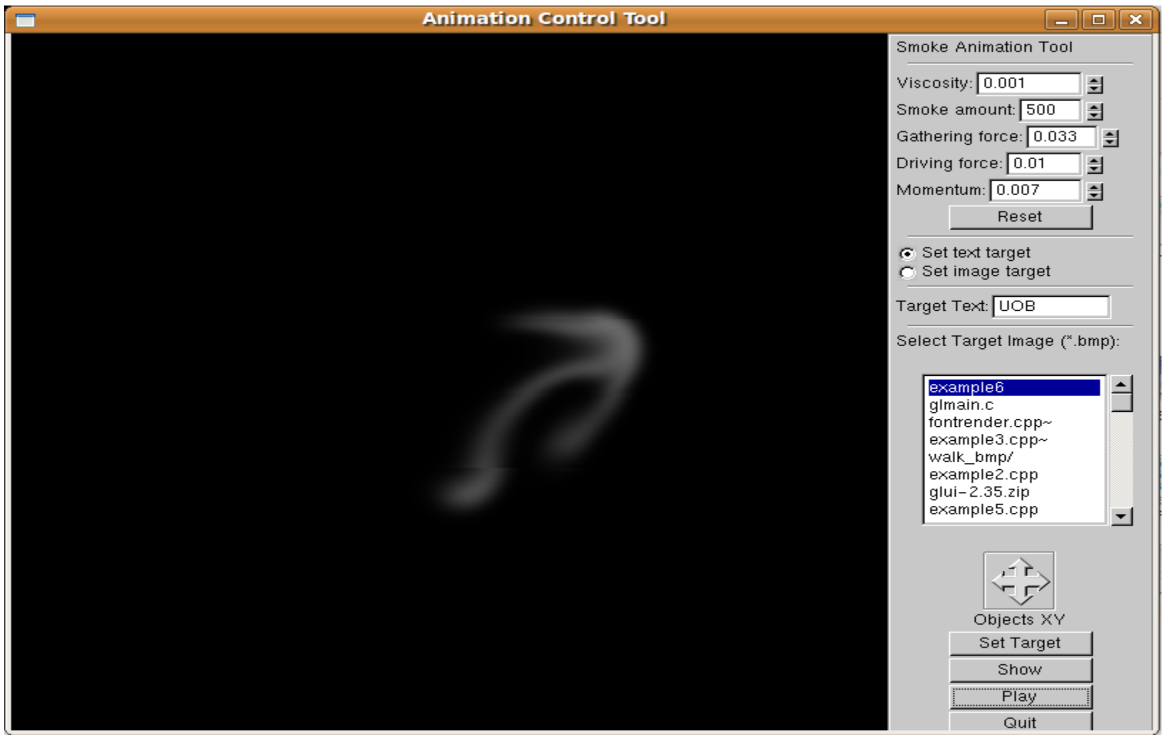
5.9 Usability Issues and actions

Following list shows identified usability issues and actions that needs to be taken:

- Participants were confused with the browser allowing to select any images of any format where only bmp format should be allowed. The file browser needed to be improved for choosing images.
- Some participants were not sure of the effect of the control parameters. The control parameters were not explained. A 'help' option was needed explaining the effect of all the parameters. Renaming the parameters to understandable terms were not enough to explain the effect of changing the parameter values.
- Some participants was confused with the sequence of images in which order they are appearing. The selected sequence of images needed to be displayed in order in the GUI.
- Participants were unable to specify the target position with coordinates on display screen. An input field was necessary for coordinate inputs.
- Some participants could not reset the display of the system to default after playing any animation on the screen. A reset button was necessary for the screen to erase the screen and bring it back to the initial state.
- Participants were able to select all types of input (text, images, animation) at once, easy to loose track of the main goal. Selecting one type of input should disable the options for other types of input, making it impossible for user to make multiple selection at once.
- Specifying target position needed to be made easier like dragging on screen with mouse.

5.10 Improvements based on the usability evaluation:

The older version of the user interface built with GLUI did not have enough features to address the usability issues mentioned above. A new improved version of the user interface was built using FLTK with added new features to solve the problems. The usability evaluation was performed on both user interfaces. The newer version showed a significant amount of improvement in the ratings on how easy or difficult it was to perform the tasks. Figure 5.4 shows a chart comparing the ratings on how easy or difficult it was to perform the tasks on a scale of one to five where five was very easy and one was very difficult. Even though the implementation of the smoke simulation was the same for the improved version of the user interface larger number of user could produce better quality animation for the ease of use. So a slight improvement in the realisticness of the animation is noticed in the chart presented in figure 5.5.



Viscosity: 0.001
Smoke amount: 500
Gathering force: 0.033
Driving force: 0.01
Momentum: 0.007

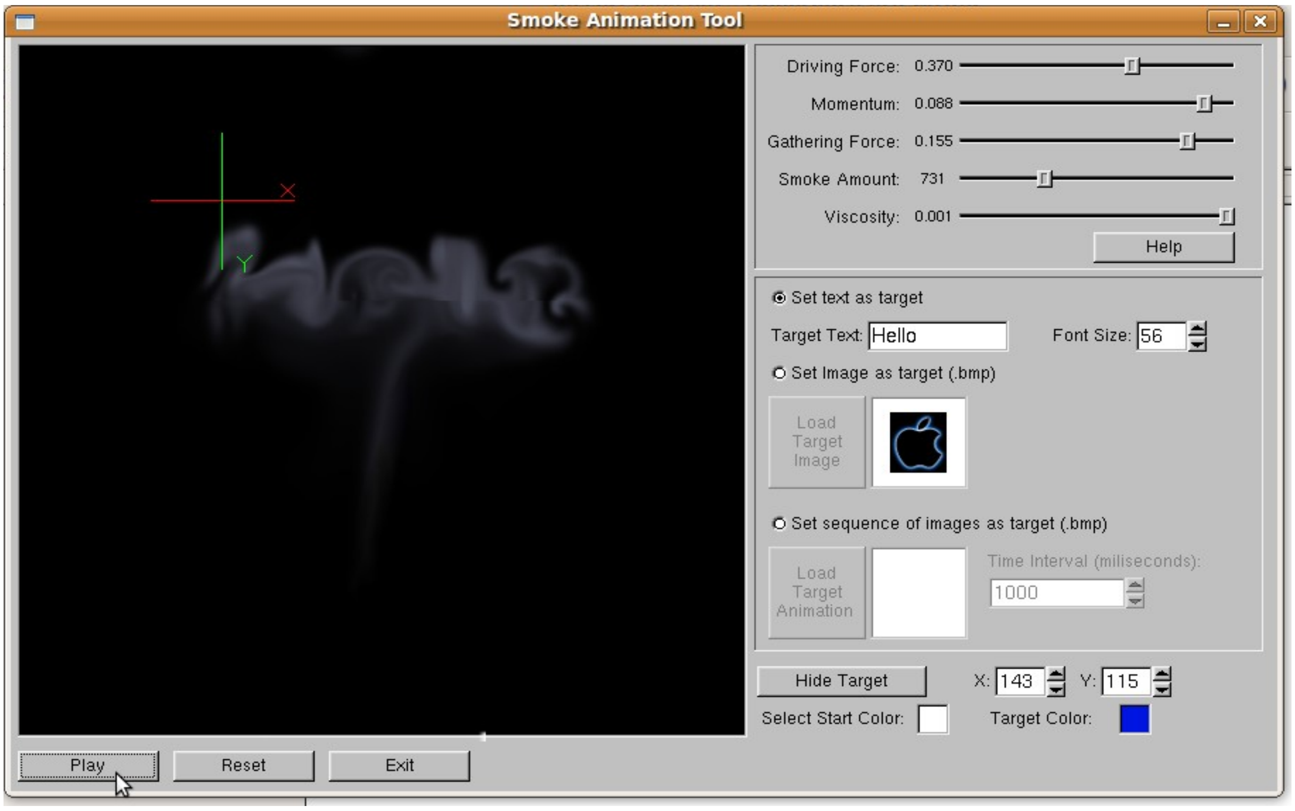
[✓ Set text target]
Set image target

Target Text: UOB

Select Target Image (*.bmp):

example6
glmain.c
fontrender.cpp~
example3.cpp~
walk_bmp/
example2.cpp
glui-2.35.zip
example5.cpp

Figure 5.2 : Image of old user interface (using GLUI)



Driving Force: 0.370
Momentum: 0.088
Gathering Force: 0.155
Smoke Amount: 731
Viscosity: 0.001

[✓ Set text as target]
Target Text: Hello
Font Size: 56
Set Image as target (.bmp)
Set sequence of images as target (.bmp)
Time Interval (miliseconds): 1000
X: 143 Y: 115
Select Start Color:
Target Color:

Figure 5.3 : Image of new improved user interface (using FLTK)

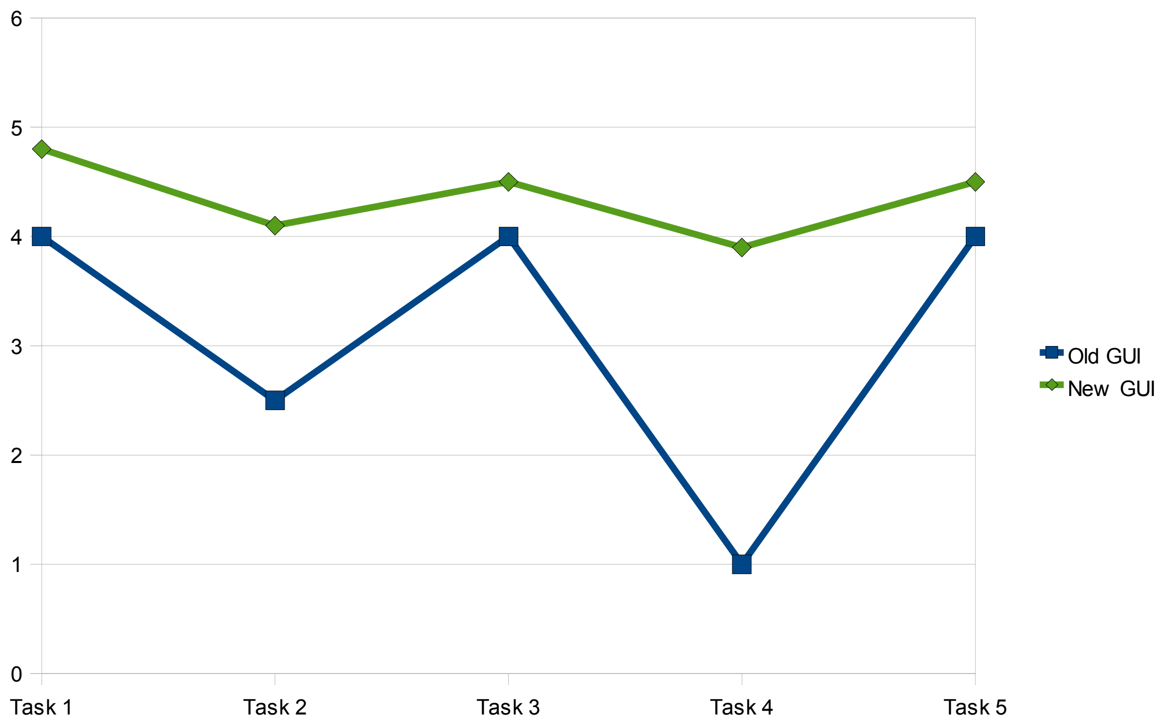


Figure 5.4 : Chart comparing the ratings on how easy or difficult it was to perform the tasks on a

scale of one to five where five was very easy and one was very difficult.

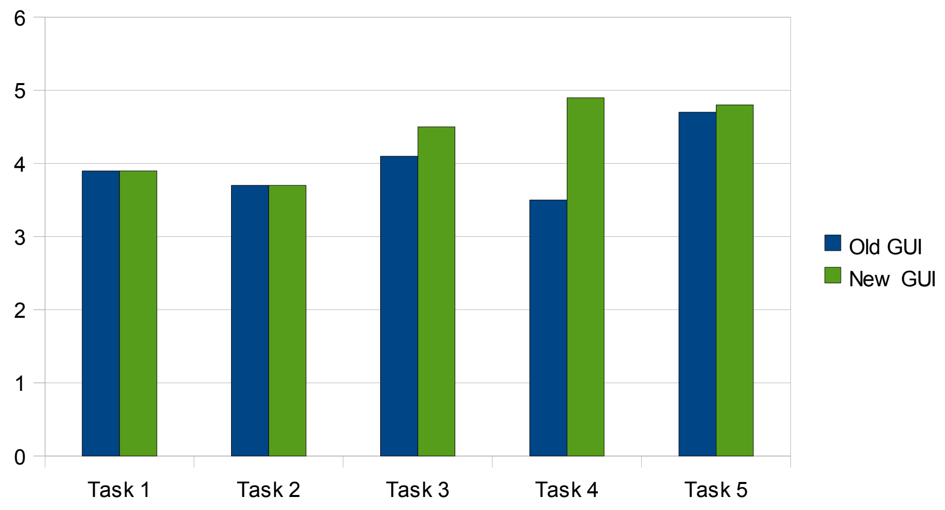


Figure 5.5 : Chart comparing the ratings on how realistic the smoke animation seemed to user for the given tasks on a scale of one to five where five was very easy and one was very difficult.

6 Future Development

Even though lots of work has been done to complete the project starting form implementation of fluid simulation with added control features and designing user interface, there is still much room for improvement. Although users were happy with the animation produced with the tool, being able to make 3d animation would have been much more interesting. Few things can be improved with the control features to make it easier to achieve good results. With further developments this system can be used as a plug-in to any animation software that exists today. This chapter presents some possible improvements that can be made on the system.

6.1 Developing in 3D

Extending the equations for 3D

The system can be implemented to produce smoke animation in 3d. At the current state the fluid solver deals the equation in 2d. Solving the equations for 3d is the same process as 2d. To start with, the 2d grid needs to be transformed into a 3d grid. Then for solving the equations, applying finite differences will produce one extra term for the third dimension. For the initialization step all the arrays need to be 3d and for the Gaussian blue operation a 3d Gaussian kernel is needed to be generated. For all the steps in the fluid solver as it loops into the grid, it needs to consider another extra loop for the extra dimension. This might slow down the computation a little bit. For Fourier transform the FFTW comes with 3d functions for creating plans and executing so it is not much difference in 3d, The functions used need to be change to the 3d version to work with 3d grid.

The derivation of the diving force and Gathering term depends on the terms produced after applying finite differences. Larger and more complicated terms needed to be dealt with for the extra dimension. This might introduce new complications and slow down the computational time.

Implementation

The steps to implement the solver in 3d would be same as in 2d but there will be small changes for the looping through the grid. It would be a nicer design choice to keep the 2d and 3d solver separate from each other. If user requires to see the animation fast then user might want to switch to 2d display and build the animation there. Then they can switch to 3d display which would use the 3d solver thread instead of 2d to display the animation. The 2d solver thread and the 3d solver thread will have the same implementation steps , only with the additional terms and variable to store the data for the extra dimension where needed for 3d. It should be quite simple to make the few additions on top of the 2d solver.

Input 3d geometry

The complicated step might be to give user the ability to input a 3d geometry and set the input as 3d target density. It will not be same as generating target density form an image or text. There are some background research presented on this topic in chapter two. Lots of computation is required to implement this feature and was not suitable for the time span of this project.

6.2 Control Features

The control features can be improved for better performance. The system generates the display form the densities updated by the solver threads. This is implemented using a continuous loop. Even after the smoke reaches the target density the display keeps updating. When user selects a new target shape and hits play button then the movement is viewable on the display. The way the solver is implemented at this stage it is possible to trace how close the current density is to the target density. The velocity field can be observed to trace what kind of movement is taking place in the grid. These informations can be very useful to improve the key-frame feature of the system or even add new features.

Because the increase of gathering term reduces the realisticness of the smoke animation but helps to form the target shape more neatly, user can set the gathering term to automatically be higher when the current density is almost approaching target density. The velocity field approaching towards zero can indicate the situation.

In case of key-framing smoke animation, it is quite unpredictable for a user to know how much time it would take for the smoke flow to reach a target with a specific setting of the control parameters. Setting the driving force term higher will make smoke reach target faster but in case of key-framing the exact time needs to be set at time interval to let the smoke form one target shape before starting to be driven towards another. In this case analysing the velocity field and the density field to see how close the simulation is to reach a target can indicate the time interval automatically. User can set the default to match every key-frame and the system will do the job for it.

6.3 Other Functions

The basics behind all types of fluid simulation is very similar. Considering the fact that the system implements the basic equations for all types of fluid flow, it can be possible to extend the system for other types of fluids instead of staying limited to smoke like animations only. The system then can perform as a tool for making all types of fluid animations. In future it can be developed as a plug-in to any 3d software and can be used for rendering outstanding fluid animations

7 Conclusion:

The main achievement of this project is a new animation tool built for animators for creating smoke animation. Since fluid simulation is one of the most complicated topics in computer graphics, it often makes the job of an animator more difficult. Using this tool an animator can simulate smoke, make any kind of shapes with it and animate the smoke in any desired way. It gives a good level of control over the smoke behaviour with some easy to use control parameters.

For the project the first task was to understand the basic concept behind fluid simulation and the equations involved in it. Most of the background research is presented on chapter two. There are many ways to solve the fluid equations and implement the basic fluid simulations. For this project it was required to be simple and fast enough to show the animation in real time. The basic fluid solver presented in 'A Simple Fluid Solver based on the FFT' by Jos Stam is used as a starting point for this project. The new terms introduced in 'Target Driven smoke animation' by Raanan fattal and Dani Lischinski has been used to control the fluid simulation and form shapes with it. Chapter three presented the project design and implementation of the basic fluid simulation and control features. The two main sections of the project are - the implementation of the fluid simulation with control features and the user interface built around it with a real-time display. The fluid solver runs on a thread and updates the display simultaneously. Because the user Interface is running side by side the changes on the user interface shows the update on the display in real-time. The user interface is designed from a user's perspective considering the usability issues. This makes the system perform as a successful animation tool.

After the development phase of the project it has been experimented with many possible inputs. It shows quite good results for the default setting of the parameters and the animation is very realistic. The system can take three types of input for a target shape – text, image and animation. User can specify the exact position of the target on the screen. The smoke will animation in its natural behaviour to the specified position and form the text or image specified by user. For an animation sequence the smoke with follow the animation. The effects of the settings on the control parameters reflects on the flow of the animation. The animation produced in the experiment has been presented with screen shots on chapter four. It is observed that the smoke animation is not always realistic looking for any combination of settings in the control parameters. It was quite clearly observed in the experiments that larger difference in the values for the driving force term and the momentum term causes the animation to become too much chaotic and sometimes impossible to achieve target shape. Lower difference in these two values and careful setting of the gathering term generates more natural smoke flow. Sometimes realisticness of smoke needed to be compromised adjusting higher value of gathering term to match target shape more closely. User needs to have a little bit of practice in using the tool and adjusting the control parameter which is very common for any kind of design tool.

For evaluation of the project it has been tested using co-operative evaluation (Monk et al., 1993). This method allows the design team to test the tool on users. The process of the user testing and performance evaluation is described in chapter five. The testers were asked to perform some tasks using the tool and requested to express there thoughts on the produced animation. It is noticed that when user gets familiar with the control parameters and their effects, tool performs quite effectively creating desired effect. The learning curve is very common for any kind of software. Among the testers 90% users were very happy with the quality of animation. The satisfaction level increased as they performed the later tasks.

Although from the evaluation it can be said that the system works successfully as an animation tool, it can be improved even more with further development. Chapter six presents some possible improvements that can be made on the system in future. User will be able to generate 3d smoke animation with added 3d feature with the existing solver. It can be used as a new innovative plug-in to existing 3d animation software. This can add new dimension of possibilities for animators as they can make interesting fluid animations like - forming different shapes with it, making it follow a specified path or even key-framing it. It can even turn a simple animation made with geometry into fluid animation. The possibilities are endless. Because they do not need to worry about the complicated fluid simulation, they can concentrate on the geometry and the movements, then turning it into a fluid animation is just a few clicks with this animation tool.

Bibliography

[BF07] BRIDSON R., FISCHER M., Fluid simulation SIGGRAPH (2007) Course Notes

[DFAB04] Dix, A, Finlay, J., Abowd, G. D. and Beale, R., 2004, Human Computer Interaction, Third Edition

[FF01] FOSTER N., FEDKIW R., : Practical animation of liquids. In Proc. Of ACM SIGGRAPH (2001). 1,2,3

[FL04] FATTAL R., LISCHINSKI D. : Target driven smoke animation. ACM Trans. Graph. 23, 3 (2004), 441 – 448. 2

[FM97] FOSTER N., METAXA D, : Controlling fluid animation. In Proc. Of CGI (1997). 2,3

[HK04] HONG J.-M., KIM C.-H.: Controlling fluid animation with geometric potential: Research articles. Comput. Animat. Virtual Worlds 15, 3-4 (2004), 147–157. 2

[HW65] HARLOW, F. H., AND WELCH, J. E. : Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface.(1965) The Physics of Fluids 8 (Dec.), 2182.2189.

[JSW05] JU T., SCHAEFER S., WARREN J. : Mean value coordinates for closed triangular meshes. ACM Trans. Graph, 24, 3 (2005), 561 – 566. 3

[LF02] LAMORLETTE A., FOSTER N.: Structural modeling of flames for a production environment. In Proc. of ACM SIGGRAPH (2002). 2

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. ACM Transactions on Graphics 23, 3 (2004), 457–462. 1

[LIGF06] LOSASSO F., IRVING G., GUENDELMAN E., FEDKIW R.: Melting and burning solids into liquids and gases. IEEE TVCG 12, 3 (2006), 343–352. 1

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle based fluid simulation for interactive applications. In Proc. of Symposium on Computer Animation (2003). 2, 3

[Mon05] MONAGHAN J. J.: Smoothed particle hydrodynamics. Rep. Prog. Phys. 68 (2005), 1703 1758. 2

[MTPS04] MCNAMARA A., TREUILLE A., POPOVIC Z., STAM J.: Fluid control using the adjoint method. ACM Trans. Graph. 23, 3 (2004), 449–456. 2

[MWHD93] Monk, A., Wright, P., Haber, J. and Davenport, L., 1993, Improving your Human-Computer Interface: A Practical Approach. Prentice Hall International, Hemel Hempsted.

[NR90] J. Nielson and R. Molich, 1990, Heuristic evaluation of user interfaces, Proceedings ACM CHI'90 (Seattle, WA, 1-5 April), 249-256.

[REN04] RASMUSSEN N., ENRIGHT D., NGUYEN D., MARINO S., SUMNER N., GEIGER W., HOON S., FEDKIW R.: Directable photorealistic liquids. In Proc. of Symposium on Computer Animation (2004). 2

[Sta99] STAM J., Stable fluids. In Proc. SIGGRAPH, pages 121–128, 1999.

[Sta01] STAM J., A Simple Fluid Solver based on the FFT, Journal of Graphics Tools, Volume 6, Number 2, 2001, 43-52 .

[SY05a] SHI L., YU Y.: Controllable smoke animation with guiding objects. ACM Trans. Graph. 24, 1 (2005). 2

[SY05b] SHI L., YU Y.: Taming liquids for rapidly changing targets. In Proc. of Symposium on Computer Animation (2005). 2, 3, 4

[TKPR06] THŰREY N., KEISER R, PAULY M and RŰDE U. : Detail preserving fluid control. (2006)

[TMPS03] TREUILLE, A., MCNAMARA, A., POPOVIC, Z., STAM, J. : Keyframe control of smoke simulations. .ACM Trans. 22, 716.723

[TVF92] Numerical Recipes in C: The Art of Scientic Computing (1992.), 2nd ed. Cambridge University Press.

[WRLP94] C. Wharton, J. Rieman, C. Lewis and P. Polson. The Cognitive Walk through: a practitioner's guide. In Usability Inspection Method. John Wiley, New York, 1994.

Websites:

http://www.fftw.org/

http://www.fltk.org/

http://glui.sourceforge.net/

http://www.libsdl.org/

Appendix A: User evaluation questionnaire

Introduction:
Thank the participant for agreeing to take part in the test session.
Explain the purpose this test.
Explain what this test will involve.
Explain that their personal information will not be shared with anyone and you want their honest opinions, no one will be offended.

Initial feedback: about the tool
What do you think you can do using this tool?
What makes you say that?
If the participant does not understand the function of the tool, explain what it does. For example, "you can use this tool to create smoke animation."

Task 1:
You want to animate your name using this tool. Where would you go to do this?
[After they were able to create the animation]:
What do you think about the smoke created for the animation?
Is this something you expected from this tool?
How do you think you can manipulate the amount of smoke?

What would you do in order to make the shape faster?

How meaningful these terms are to you? What do you think will happen if you change the momentum?

How easy or difficult was this task to you?

Please rate from 1 to 5 where 1 is very difficult and 5 is very easy to complete this task.

Task 2:
You want to create a smoke animation of an image. What would you do in order to create an animation of an image using this tool?
[If user is not sure, guide them to click on 'Set image as target']

You want to change the location of the animation within this section. Do you think you can create the animation at the corner of this window?
If yes, how would you do that.
If no, guide them to set position target.

How would you change the target using this?

How easy or difficult was this task to you?

Please rate from 1 to 5 where 1 is very difficult and 5 is very easy to complete this task.

Task 3:
You have a series of images that can create a stop motion animation. You want to create the fluid animation for those image. How would you do that?
[Guide them to the folder of images.]
You want to change the colour from white to blue. Do you think you can change the colour of the smoke using this tool? How would you do this?
You want to start your animation with orange coloured smoke and end it with blue. Is there a way of doing it?

How easy or difficult was this task to you?

Please rate from 1 to 5 where 1 is very difficult and 5 is very easy to complete this task.

Closing the session:
What is your over all impression of using this tool?
Please rate how easy or difficult it is to control the smoke animation on a scale of 1 to 5 where 1 is very difficult and 5 is very easy.
Please rate how realistic the smoke simulation is looking as you adjust the parameterson a scale of 1 to 5 where 1 is very unrealistic and 5 is very realistic.
Please rate the design and layout of this tool on a scale of 1 to 5 where 1 is poor and 5 is attractive.

Would you use this tool to create smoke animation?
If someone is looking to create smoke animation, would you recommend this tool to them? Why/why not?

Appendix B: Source Code

```
class display_box : public Fl_Gl_Window {
  void draw();
  int handle(int);
public:

  double vf;
  display_box(int x,int y,int w,int h,const char *l=0)
    : Fl_Gl_Window(x,y,w,h,l) {lasttime = 0.0;}
};

Fl_Window *form;

Fl_Value_Slider *vf_c, *size, *vg_c, *vd_c, *visc_c, *smk_c;
Fl_Button *button, *show, *wire, *flat, *set_target, *picload,
*animload, *help, *reset, *col_box_s, *col_box_t;
display_box *dp, *dp2;
```

```
void makeform(const char *name)
{

  int padding=5, count=0;
  int button_w=100;
  int button_h=23;
  int button_gap=5;
  int button_y=button_h+button_gap;
  int box1_w=516;
  int box1_h=516;
  int box2_w=360;
  int box2_h=168;
  int control_x= box1_w + 2*padding +10;
  int control_y= padding+10;
  form = new
Fl_Window(box1_w+2*padding+box2_w+2*padding,box1_h+2
Fl_Input *txt_in;
Fl_Spinner *posx_in, *posy_in, *fsize_in, *time_int;

Fl_Group *txt_gp, *img_gp, *anim_gp;
Fl_File_Browser                 *files;
Fl_File_Chooser                 *fc;

Fl_Box *seq_box, *seq_box_in, *img_box, *img_box_in,
*time_box, *colorload_s, *colorload_t;
Fl_Window *w;
Fl_Shared_Image *img;

void color_cb_s(Fl_Widget* button, void* v)
{

  if (fl_color_chooser(0,r_s,g_s,b_s))
  {

    col_box_s->color(fl_rgb_color(r_s, g_s, b_s)); col_box_s-
>redraw();
    r_val=(float)r_s/225; g_val=(float)g_s/225;
b_val=(float)b_s/225;

  }
}
void color_cb_t(Fl_Widget* button, void* v)
{

  if (fl_color_chooser(0,r_t,g_t,b_t))
  {

    col_box_t->color(fl_rgb_color(r_t, g_t, b_t)); col_box_t-
>redraw();

  }
}

void draw_axes( float scale )
{


  glPushMatrix();
  glScalef( scale, scale, scale );

  glBegin( GL_LINES );

  glColor3f( 1.0, 0.0, 0.0 );
  glVertex3f( .8f, 0.05f, 0.0 );  glVertex3f( 1.0, 0.25f, 0.0 ); /*
Letter X */
  glVertex3f( 0.8f, .25f, 0.0 );  glVertex3f( 1.0, 0.05f, 0.0 );
  glVertex3f( -1, 0.0, 0.0 );  glVertex3f( 1, 0.0, 0.0 ); /* X axis
*/

  glColor3f( 0.0, 1.0, 0.0 );
  glVertex3f( .2f, -0.8f, 0.0 ); glVertex3f( .3f, -0.9f, 0.0 );
  glVertex3f( 0.4, -0.8, 0.0 ); glVertex3f( .3f, -0.9f, 0.0 );
  glVertex3f( 0.3, -0.9, 0.0 ); glVertex3f( .3f, -1.05f, 0.0 );
  glVertex3f( 0.0, 1, 0.0 );  glVertex3f( 0.0, -1, 0.0 ); /* Y axis
*/


  glEnd();

  glPopMatrix();


}
```

```
*padding+35,name);
  new
Fl_Box(FL_DOWN_FRAME,padding,padding,box1_w,box1_h,
"");
  new
Fl_Box(FL_THIN_DOWN_FRAME,box1_w+2*padding,paddi
ng,box2_w,box2_h,"");
  new Fl_Box(FL_THIN_DOWN_FRAME,box1_w+2*padding,
padding+box2_h+5,box2_w,12*button_h+3,"");
  vf_c = new
Fl_Value_Slider(control_x+100,control_y+count*button_y,230,b
utton_h-10,"Driving Force: ");

  vf_c->type(5);
     vf_c->box(FL_FLAT_BOX);
     vf_c->labelsize(12);
     vf_c->step(0.001);
     vf_c->align(FL_ALIGN_LEFT);
  count++;
  vd_c = new
Fl_Value_Slider(control_x+100,control_y+count*button_y,230,b
utton_h-10,"Momentum: ");
  vd_c->type(5);
     vd_c->box(FL_FLAT_BOX);
     vd_c->labelsize(12);
     vd_c->step(0.001);
     vd_c->align(FL_ALIGN_LEFT);
  count++;
  vg_c = new
Fl_Value_Slider(control_x+100,control_y+count*button_y,230,b
utton_h-10,"Gathering Force: ");
  vg_c->type(5);
     vg_c->box(FL_FLAT_BOX);
     vg_c->labelsize(12);
     vg_c->step(0.001);
     vg_c->align(FL_ALIGN_LEFT);
  count++;
  smk_c = new
Fl_Value_Slider(control_x+100,control_y+count*button_y,230,b
utton_h-10,"Smoke Amount: ");
  smk_c->type(5);
     smk_c->box(FL_FLAT_BOX);
     smk_c->labelsize(12);
     smk_c->step(1.0);
     smk_c->align(FL_ALIGN_LEFT);
  count++;
  visc_c = new
Fl_Value_Slider(control_x+100,control_y+count*button_y,230,b
utton_h-10,"Viscosity: ");
  visc_c->type(5);
     visc_c->box(FL_FLAT_BOX);
     visc_c->labelsize(12);
     visc_c->step(0.001);
     visc_c->align(FL_ALIGN_LEFT);
  count++;
  help = new Fl_Button(control_x+230,
control_y+count*button_y-10, button_w, button_h, "Help");
  help->labelsize(12);

  count++;

    { Fl_Group* o = new Fl_Group(control_x,
control_y+count*button_y, box2_w-20, 250);
     o->box(FL_FLAT_BOX);
     { Fl_Round_Button* o = new Fl_Round_Button(control_x,
control_y+count*button_y, button_w, button_h, "Set text as
target");
```

```
void display_box::draw()
{
		static GLint T0     = 0;
		static GLint Frames = 0;

		LoadGLTextures();
		glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);
		glLoadIdentity();
		glPushMatrix();
		glEnable(GL_TEXTURE_2D);
		glBindTexture(GL_TEXTURE_2D, texture[0]);
		glColor3f(r_val, g_val, b_val);
		drawRect();
		glDisable(GL_TEXTURE_2D);
		glPopMatrix();
		if(target==1)
		{
				glPushMatrix();
				glTranslatef( target_x, target_y, 0.0 );
				draw_axes(.2f);
				glPopMatrix();

		}
}
int display_box::handle(int e)
{
		switch (e)
		{
				case FL_ENTER:
				{
						cursor(FL_CURSOR_CROSS);
						break;
				}
				case FL_LEAVE:
cursor(FL_CURSOR_DEFAULT); break;
				case FL_PUSH: return 1;
				case FL_DRAG:
				{
						target_x=(float)(Fl::event_x()-
256)/256;
						target_y=(float)(256-
Fl::event_y())/256;//printf("\nx=%f y=%f\n", target_x,
target_y);
						posx_in->value(Fl::event_x());
						posy_in->value(Fl::event_y());
						posx_in->redraw();
						posy_in->redraw();
						break;
				}
		}
		return Fl_Gl_Window::handle(e);
}


static char name[1024];
char filename[1024];

void file_cb(const char *n)
{
		static char name[1024];
		int h=0,w=0;
		if (!strcmp(name,n)) return;
		strcpy(name,n);
		strcpy(filename,n);
		img = Fl_Shared_Image::get(n);
		w=img->w(); h=img->h();
```

```
        o->type(102);
        o->down_box(FL_ROUND_DOWN_BOX);
        o->labelsize(12);
            o->callback(txt_cb);
            o->value(1);
      } // Fl_Round_Button* o
      count++;
      { txt_gp = new Fl_Group(control_x,
control_y+count*button_y, box2_w-20, button_h);
            txt_gp->box(FL_FLAT_BOX);

         { txt_in = new Fl_Input(control_x+70,
control_y+count*button_y,button_w, button_h,"Target Text:");
              txt_in->tooltip("Target text");
              txt_in->labelsize(12);
              txt_in->when(FL_WHEN_RELEASE);
              txt_in->callback(set_txt_cb);
              txt_in->value("UOB");
              fsize_in = new Fl_Spinner(control_x+button_w+160,
control_y+count*button_y,button_w/2, button_h,"Font Size:");
          fsize_in->minimum(8);
              fsize_in->maximum(60);
          fsize_in->step(2);
          fsize_in->value(30);
          fsize_in->labelsize(12);
              fsize_in->when(FL_WHEN_RELEASE);
     //posx_in->value(5);
         fsize_in->align(FL_ALIGN_LEFT);
         fsize_in->callback(set_fsize_cb);
         }
        txt_gp->end();
      } count++;
      { Fl_Round_Button* o = new Fl_Round_Button(control_x,
control_y+count*button_y, button_w, button_h, "Set Image as
target (.bmp)");
        o->type(102);
        o->down_box(FL_ROUND_DOWN_BOX);
            o->labelsize(12);
            o->callback(img_cb);
      } // Fl_Round_Button* o
      count++;
      { img_gp = new Fl_Group(control_x,
control_y+count*button_y, box2_w-20, 3*button_h);
            img_gp->box(FL_FLAT_BOX);
         {
                    picload = new
Fl_Button(control_x,control_y+count*button_y,3*button_h,3*bu
tton_h,"Load Target Image");
                    picload->callback(button_cb);
                    picload->align(FL_ALIGN_WRAP);
                    picload->labelsize(12);
                    img_box=new
Fl_Box(FL_DOWN_FRAME,
control_x+3*button_h+3,control_y+count*button_y,3*button_h,
3*button_h,0);
                    //img_box->align(FL_ALIGN_INSIDE|
FL_ALIGN_CENTER);
                    img_box->box(FL_DOWN_BOX);
                    img_box->color(FL_WHITE);

         }
        img_gp->end();
      } count=count+3; img_gp->deactivate();
      { Fl_Round_Button* o = new Fl_Round_Button(control_x,
control_y+count*button_y, button_w, button_h, "Set sequence of
images as target (.bmp)");
        o->type(102);
```

```
		if (img->w() >= img_box->w() || img->h() >=
img_box->h())
		{
			Fl_Image *temp;
			if (img->w() >= img_box->w())
				w=img_box->w()-4;
			if (img->h() >= img_box->h())
			h=img_box->h()-4;
			temp = img->copy(w, h);
			img->release();
			img = (Fl_Shared_Image *)temp;
		}
		img_box->image(img);
		img_box->redraw();
		img_gp->redraw();
}
void seq_file_cb(const char *n)
{
		char* fname="";
		int i=1;
		char num[16];
		fn=strdup((char*)n);
		while(1)
		{
			fname=strdup((char*)n);
			snprintf(num, 16, "%d", i);
			strcat(num,".bmp");
			strcat(fname,"/");
			strcat(fname,num);
			static char name[1024];
			int h=0,w=0;
			img = Fl_Shared_Image::get(fname);if(!img)
break;
			w=img->w(); h=img->h();
			if (img->w() >= seq_box->w() || img->h() >=
seq_box->h())
			{
				Fl_Image *temp;
				if (img->w() >= seq_box->w())
					w=seq_box->w()-4;
				if (img->h() >= seq_box->h())
					h=seq_box->h()-4;
				temp = img->copy(w, h);
				img->release();
				img = (Fl_Shared_Image *)temp;
			}
			seq_box->image(img);
			seq_box->redraw();
			anim_gp->redraw();usleep(500000);;i++;
		}
}
void button_cb(Fl_Widget *,void *)
{
		int count=0;
		fc = new Fl_File_Chooser(".", "*.{bmp}",
Fl_File_Chooser::SINGLE, "Fl_File_Chooser Test");
		fc->show();
		while (fc->visible())
		{
			Fl::wait();
		}
		count = fc->count();
		if(count>0)
		file_cb(fc->value());
}
void *SeqThread(void *threadid)
{
```

```
      o->down_box(FL_ROUND_DOWN_BOX);
          o->labelsize(12);
          o->callback(anim_cb);
     } // Fl_Round_Button* o
     count++;
    { anim_gp = new Fl_Group(control_x,
control_y+count*button_y, box2_w-20, 3*button_h);

         anim_gp->box(FL_FLAT_BOX);
       {
			animload = new
Fl_Button(control_x,control_y+count*button_y,3*button_h,3*bu
tton_h,"Load Target Animation ");
			animload->callback(seq_cb);
			animload->align(FL_ALIGN_WRAP);
			animload->labelsize(12);
			seq_box=new Fl_Box(FL_DOWN_FRAME,
control_x+3*button_h+3,
control_y+count*button_y,3*button_h,3*button_h,0);
			seq_box->align(FL_ALIGN_INSIDE|
FL_ALIGN_CENTER);
			seq_box->box(FL_DOWN_BOX);
			seq_box->color(FL_WHITE);
			time_box = new
Fl_Box(FL_FLAT_BOX,control_x+3*button_h+3+3*button_h,c
ontrol_y+count*button_y,button_w+80,button_h,"Time Interval
(miliseconds):");
			time_box->labelsize(12);
			count++;
			time_int = new
Fl_Spinner(control_x+3*button_h+3+3*button_h+15,control_y+
count*button_y-5,button_w+10,button_h,"");
			time_int->minimum(0);
			time_int->maximum(10000);
			time_int->step(100);
			time_int->value(1000);
			//posx_in->value(5);
			time_int->align(FL_ALIGN_RIGHT);
			time_int->callback(set_pos_cb);
			time_int->labelsize(12);count--;
       }
      anim_gp->end();
     } count=count+3; anim_gp->deactivate();
     o->end();
   } // Fl_Group* o
  set_target = new Fl_Button(control_x-
8,5+control_y+count*button_y, button_w+20, button_h,
"Position Target ");
  set_target->callback(set_target_cb);
  set_target->labelsize(12);
  posx_in = new Fl_Spinner(control_x+button_w+60,
5+control_y+count*button_y, button_w/2, button_h,"X:");
  posx_in->minimum(0);
  posx_in->maximum(512);
  posx_in->step(3);
  posx_in->value(256);
     //posx_in->value(5);
  posx_in->align(FL_ALIGN_LEFT);
  posx_in->labelsize(12);
  posx_in->callback(set_pos_cb);
  posy_in = new
Fl_Spinner(control_x+button_w+60+button_w/2+25,
5+control_y+count*button_y, button_w/2, button_h,"Y:");
  posy_in->minimum(0);
  posy_in->maximum(512);
  posy_in->step(3);
  posy_in->value(256);
		seq_file_cb(fc->value());
		pthread_exit(NULL);
}
void seq_cb(Fl_Widget *,void *)
{
		int count=0, th;
		pthread_t threads[1];
		long t=0;
		fc = new Fl_File_Chooser(".", "*",
Fl_File_Chooser::DIRECTORY, "Fl_File_Chooser Test");
		fc->show();

		while (fc->visible())
		{
			Fl::wait();
		}
		count = fc->count();
		if(count>0)
		th = pthread_create(&threads[t], NULL, SeqThread,
(void *)t);
}
int dvisual = 0;
int arg(int, char **argv, int &i)
{
  if (argv[i][1] == '8') {dvisual = 1; i++; return 1;}
  return 0;
}

void StartAnim()
{
		int i=1;
		char num[16];
		int pos=0;
		char *fname="";
		while(1)
		{
			fname=strdup((char*)fn);
			snprintf(num, 16, "%d", i);
			strcat(num,".bmp");
			strcat(fname,"/");
			strcat(fname,num);
			img = Fl_Shared_Image::get(fname);if(!img)
break;
			memset(ps_fft, 0,
Z_SIZE*Y_SIZE*X_SIZE*sizeof(scalar));
			AnimImageRender(ps_fft, fname, pos);
			MakeSmoothedTargetDensity();
			usleep(1000*time_int->value());
			i++;pos=pos+15;
		}

}
void *AnimThread(void *threadid)
{
		StartAnim();
		pthread_exit(NULL);
}
void reset_cb( Fl_Widget* o, void*)
{
  vf_c->value(0.01);
  vg_c->value(0.033);
  vd_c->value(0.007);
  smk_c->value(500.0);
  visc_c->value(0.001);
  fsize_in->value(30);
  time_int->value(1000);
  Reset();
```

```
    //posx_in->value(5);
  posy_in->align(FL_ALIGN_LEFT);
  posy_in->callback(set_pos_cb);
  posy_in->labelsize(12);
  count++;
  colorload_s = new Fl_Box(control_x-
5,5+control_y+count*button_y, button_w, button_h, "Select
Start Color:");
  //colorload_s->callback(color_cb_s, (void*)1);
  colorload_s->labelsize(12);
  col_box_s=new
Fl_Button(control_x+button_w+5,5+control_y+count*button_y,
button_h, button_h,0);
  col_box_s->box(FL_DOWN_BOX);
  col_box_s->color(FL_WHITE);
  col_box_s->callback(color_cb_s, (void*)1);
  colorload_t = new
Fl_Box(control_x+button_w+5+button_h+15 ,
5+control_y+count*button_y, button_w, button_h, "Target
Color:");
  //colorload_t->callback(color_cb_t, (void*)1);
  colorload_t->labelsize(12);
  col_box_t=new
Fl_Button(control_x+button_w+5+button_h+15+button_w+5,5+
control_y+count*button_y, button_h, button_h,0);
  col_box_t->box(FL_DOWN_BOX);
  col_box_t->color(FL_WHITE);
  col_box_t->callback(color_cb_t, (void*)1);
  count++;
  show = new
Fl_Button(padding,control_y+count*button_y+12,button_w,butt
on_h,"Play");
  show->callback(show_cb);
  show->labelsize(12);
  reset = new
Fl_Button(padding+button_w+10,control_y+count*button_y+12
,button_w,button_h,"Reset");
  reset->callback(reset_cb);
  reset->labelsize(12);
  button = new
Fl_Button(padding+2*button_w+20,control_y+count*button_y+
12,button_w,button_h,"Exit");
  button->labelsize(12);
  count++;
  dp = new display_box(7,7,512,512, 0);
  form->end();
}

void anim_cb(Fl_Widget* o, void*)
{
		deact_all();
		anim_gp->activate();
		obj=2;
}
void set_txt_cb(Fl_Widget* o, void*)
{
		show_txt = strdup((char*)txt_in->value());
}
void set_pos_cb(Fl_Widget* o, void*)
{


    target_x=(float)(posx_in->value()-256)/256;
    target_y=(float)(256-posy_in->value())/256;
}
void set_fsize_cb(Fl_Widget* o, void*)
{
    fontsize=fsize_in->value();
```

```
}
void show_cb( Fl_Widget* o, void*)
{
        char *txt;
        pthread_t threads[1];
    long t=0;
    int anim_t;
    change=1; wt=1;
    int p,q;
        memset(ps_fft, 0,
Z_SIZE*Y_SIZE*X_SIZE*sizeof(scalar));
                if(obj==2)
                {
                        anim_t =
pthread_create(&threads[t], NULL, AnimThread, (void *)t);
                }
                else
                {
                    if(obj==0)
                        {
                                TextRender(ps_fft,
show_txt);

        MakeSmoothedTargetDensity();
                        }
                        if(obj==1 && filename[0]!=0)
                        {
                ImageRender(ps_fft, filename);

        MakeSmoothedTargetDensity();
                        }

                }
}
void set_target_cb(Fl_Widget* o, void*)
{
  if(target==0)
  {
   target=1;
   set_target->label("Hide Target");
   set_target->redraw();
  }
  else
  {
   target=0;
   set_target->label("Position Target");
   set_target->redraw();
  }

}
void deact_all()
{
        txt_gp->deactivate();
        img_gp->deactivate();
        anim_gp->deactivate();
}
```