Physically Based Simulation and Animation of Gaseous Phenomena in a Periodic Domain

Andrew Nealen*
Department of Computer Science, University of British Columbia
CPSC 533b: Algorithmic Animation, *Project Report*

Abstract

In this project report I present my research and accompanying implementation of the unconditionally stable Navier-Stokes partial differential equation (PDE) solver presented to the computer graphics community by Jos Stam at the 1999 *SIGGRAPH* conference [Stam 1999]. I give a brief introduction to computational fluid dynamics (CFD), the governing Navier-Stokes equations and an overview of previous work in the field. My software implementation (the `flowanim` demo) is based on Stam's simple fluid solver [Stam 2002] and therefore also makes use of a *black box* Fourier transform, the FFTW package [Frigo and Johnson 1998]. The smoke animation system is further enhanced by addition of a powerful technique known as *vorticity confinement*, which was introduced to CG at *SIGGRAPH* 2001 [Fedkiw et al. 2001]. Finally, I visualize the solution of all terms of the Navier-Stokes equations, including the vorticity confinement term, to help students and researchers unfamiliar with the equations of fluid flow to easily understand the underlying concepts.

Keywords: Stable Fluids, Navier-Stokes Equations, Fast Fourier Transform, Vorticity Confinement

1 Introduction

1.1 Previous Work

Computational fluid dynamics have been extensively researched and been widely applied to computer graphics and animation. For these purposes, specifically for real-time simulation and rendering, the accuracy of a solution is less important than obtaining approximations as fast as possible. Jos Stam's recent development of an unconditionally stable solution scheme for the full Navier-Stokes equations [Stam 1999; Stam 2002] has been a major breakthrough, since even the state-of-the-art fluid animation system used for the computer animated and academy award winning film *Shrek* employs this method [Foster and Fedkiw 2001].

Over the past two decades many different methods of simulating and animating fluid and gas-like flows have been proposed. Chen et. al. [Chen and Lobo 1995] solve the two-dimensional Navier-Stokes equations and then raise the surface of the fluid according to the corresponding pressures in the flow field. Their solution of the equations is based on finite differencing and an explicit time solver and therefore suffers from potential numerical instabilities if the timestep is chosen too large (this is later explained in more detail). Foster and Metaxas [Foster and Metaxas 1996; Foster and Metaxas 1997a; Foster and Metaxas 1997b] solve the full three-dimensional Navier-Stokes equations, yielding visually impressive results at the cost of additional complexity ($O(n^3)$) and potential numerical instability as outlined above. Kass and Miller [Kass and Miller 1990] linearize the shallow water equations, which assume zero viscosity and only two-dimensional motions. This model has been extended by Layton and van de Panne [Layton and van de Panne 2002] to nonlinear shallow water equations using an implicit and unconditionally stable semi-Lagrangian integration method. Stam, Fedkiw and Jensen recently presented an extension to the stable solver [Stam 1999], which is designed to reduce the amount of numerical dissipation inherent in semi-Lagrangian schemes [Fedkiw et al. 2001]. They use a higher order interpolation for the particle trace used to solve the advection term and additionally apply an extra force to the velocity field to keep small scale vortices alive. This technique, also known as *vorticity confinement*, was invented by Steinhoff approximately a decade ago [Steinhoff and Underhill 1994]. And finally, Foster and Fedkiw [Foster and Fedkiw 2001] use the semi-Lagrangian integration scheme for the advection step, but still enforce a Courant-Friedrichs-Levy (CFL) condition on the timestep Δt for updating the surface particles and the implicit surface (the level set). This allows to alternate between a large timestep for updating the Navier-Stokes equations and series of small timesteps for the particles and the level set, resulting in large computational savings.

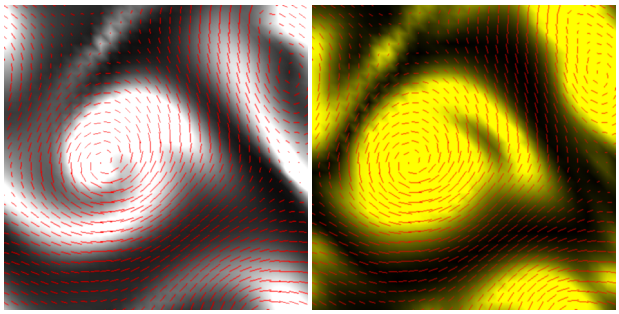

Figure 1: Left: smoke density and the velocity vector field. Right: the curl (vorticity) of the velocity vector field

1.2 The Model used in this Project

The model I use for this project is tailored toward a better understanding of the underlying concepts and computations through use of extensive visualization of the solution process. For simplification reasons (and because the solution is quite elegant) I make the assumption of a periodic (wrapping), two dimensional domain. This allows for simple animations without boundaries, which can be used as alpha textures to simulate rising smoke, as well as wrapping textures, which tile nicely and can be used to seamlessly texture map arbitrary meshes. All tweakable parameters can be modified with simple keypresses (see appendix for a brief description of the simulator). The solver I implemented for this project uses the semi-Lagrangian scheme and a particle tracer based on linear interpolation to solve the advection step of Navier-Stokes, resulting in increased numerical dissipation. In [Fedkiw et al. 2001] the authors propose two methods to fight this effect. For this project I have implemented *vorticity confinement*, which can be tweaked/toggled on

*nealen@cs.ubc.ca

the fly to observe the visual effect on the velocity and density fields.

The assumption of a periodic domain leads to an elegant solution to the Navier-Stokes equations for incompressible, Newtonian fluids and serves as a good entry point for anybody new to fluid dynamics and interested in coding and experimenting with a basic but efficient and visually convincing fluid solver.

2 Navier Stokes and the Stable Solution Scheme

2.1 Fundamentals of Navier-Stokes

All references presented in Section 1.1 use a physical model based on the Navier-Stokes equations to simulate the evolution of a two- or three-dimensional velocity field (and in some cases a pressure field) over a timestep Δt. The compact (vector) form of the Navier-Stokes equations is:

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

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

where ν is the kinematic viscosity of the fluid, ρ is the density, $\mathbf{f}$ is an external force, $\mathbf{u}$ is the velocity (vector)field, p is the scalar pressure field and ∇ is the vector of spatial partial derivatives, more precisely $\nabla = (\partial/\partial x, \partial/\partial y)$ in two and $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$ in three dimensions. The Navier-Stokes equations constitute that the fluid conserves both mass (1) and momentum (2). The derivation of the equations is beyond the scope of this report, but there exist many standard texts on the topic. I found much useful information on the Internet using a commonly used search engine.

Most methods solve each term on the right side of (2) sequentially and then make the resulting field mass conserving, thereby enforcing (1). Before Stam introduced the stable solution scheme, solvers used finite differencing and an explicit timestep, which can result in numerical instabilities when the timestep is chosen too large and/or the spatial discretization is very fine. In mathematical terms, Δt must satisfy $\Delta t < \Delta\tau/|\mathbf{u}|$, where $\Delta\tau$ is the spacing of the discretization grid. Intuitively, no particle of the fluid may travel more than the distance of one grid cell per timestep, otherwise the simulation may "blow up" and have to be restarted with a smaller timestep or a coarser grid.

The stable solver I used in this project uses a modification of the Navier-Stokes equations which I will briefly describe. For a more detailed overview, see [Stam 1999]. A mathematical result, known as *Helmholtz-Hodge Decomposition*, states that any vector field $\mathbf{w}$ can be uniquely decomposed into the form:

$$\mathbf{w} = \mathbf{u} + \nabla q, \tag{3}$$

where $\mathbf{u}$ is mass conserving, that is $\mathbf{u}$ satisfies (1), and ∇q is the gradient field of the scalar field q. By multiplying both sides of (3) with the gradient operator ∇, an operator $\mathbf{P}$ can be defined which projects any vector field $\mathbf{w}$ onto its divergence free part $\mathbf{u} = \mathbf{P}\mathbf{w}$:

$$\nabla \cdot \mathbf{w} = \nabla^2 q. \tag{4}$$

This is a Poisson equation which can be solved for the scalar field q. This step is greatly simplified by assuming periodic boundaries as I will describe later. After solving for the unknown quantity q the projection can be computed as follows:

$$\mathbf{u} = \mathbf{P}\mathbf{w} = \mathbf{w} - \nabla q. \tag{5}$$

Applying this projection on both sides of equation (1) and using the fact that $\mathbf{P}\mathbf{u} = \mathbf{u}$ and $\mathbf{P}\nabla p = 0$ yields:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{P}(-(\mathbf{u} \cdot \nabla) \cdot \mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{f}). \tag{6}$$

From this equation the stable solver is developed, sequentially solving the three terms on the right side of (6) and then projecting the resulting velocity field onto its divergence free part. As I assume periodic boundaries in this project, I can perform the diffusion ($\nu\nabla^2\mathbf{u}$) and projection steps in the Fourier domain. This is described in more detail in Section 3. Applying external force and the advection step ($-(\mathbf{u} \cdot \nabla) \cdot \mathbf{u}$) are identical in both cases (except for the fact that periodic domains make the particle trace, which is used to solve for the advection step, somewhat easier as I will soon describe).

2.2 Applying External Force

The by far simplest step is applying external force $\mathbf{f}$. In assuming that the user-applied force does not vary considerably during the timestep I simply compute $\mathbf{w}_1 = \mathbf{w}_0 + \Delta t\, \mathbf{f}(\mathbf{x}, t)$. The external forces are applied in the accompanying `flowanim` demo by left clicking and dragging in the main window. The speed and locality of the mouse drag is then translated into forces which are stored in a grid and applied at the beginning of each timestep. The stored forces are then dampened by a factor of 0.85 after each timestep. This results in a nice visual control of the velocity field and the smoke density therein. Vorticity confinement forces may be added to the external forces in each timestep (this will be covered in Section 5).

2.3 Advection

The advection term $-(\mathbf{u} \cdot \nabla) \cdot \mathbf{u}$ makes the Navier-Stokes equations non-linear and has previously been solved using finite differencing [Foster and Metaxas 1996], which can be unstable as described in Section 2.1. Stam proposes a method new to computer graphics known as the *method of characteristics*. For the mathematical background see the appendix in [Stam 1999]. Nevertheless, this method can also be described more intuitively: At each timestep, the fluid particles are moved by the fluid itself. So to compute the new velocity at some arbitrary grid location $\mathbf{x}$ at time $t + \Delta t$ it is necessary to backtrace the particle through the previously computed velocity field $\mathbf{w}_1$. The new velocity at location $\mathbf{x}$ is then set to the velocity that the particle (now at $\mathbf{x}$) had at time Δt ago.

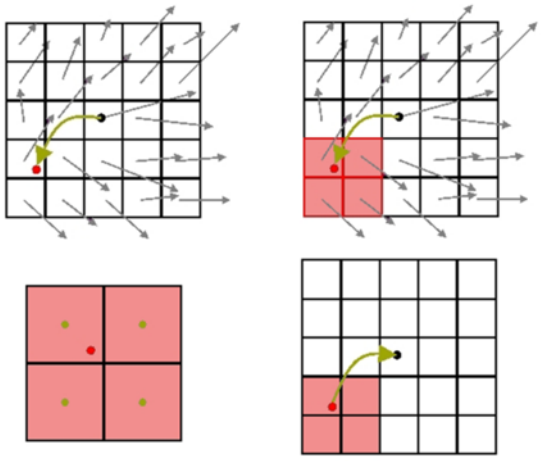

Figure 2: Solving the advection step

The result of this operation is the velocity field $\mathbf{w}_3$.

To visualize this, examine Figure 2: In the top left, we backtrace the particle located at grid cell $\mathbf{x}$ until we find its origin (Figure 2 upper right, red dot in shaded area). We then interpolate between the

four surrounding cells (in 2D) and set the new velocity at $\mathbf{x}$ to the interpolated value (Figure 2 bottom right). This ensures that the simulation never "blows up" as the new velocity can never be greater than the velocity at time Δt ago. In Stam's implementation [Stam 2002] the particle trace is done using simple linear interpolation, increasing the amount of numerical dissipation. This can be reduced by implementing the particle trace using higher order interpolation such as monotonic cubic interpolation [Fedkiw et al. 2001]. Stam himself mentions though, that linear interpolation combined with vorticity confinement already leads to visually satisfying results. The implementation (the `flowanim` demo) for this project therefore uses linear interpolation and vorticity confinement (see Section 5). The periodic boundaries of the domain simplify the particle trace insofar as the particles which leave one side of the domain simply re-enter on the other side. This can be realized in software using a few *modulo* operations (see source code).

3 Periodic Boundaries and the FFT

3.1 The Velocity Field in the Fourier Domain

To transform the velocity field into the Fourier domain I used the *fastest Fourier transform in the west* [Frigo and Johnson 1998], the same blackbox transform used by Stam in [Stam 2002]. In the Fourier domain, the gradient operator "∇" is equivalent to multiplication by $i\mathbf{k}$, where $i = \sqrt{-1}$ and $\mathbf{k}$ is the corresponding frequency or *wavenumber*. This makes both the diffusion and projection steps much simpler to solve and code. The field is transformed independently for the u and v components of the velocity: each component is treated as a scalar field.

3.2 Diffusion (Viscosity)

The diffusion step accounts for the "thickness" of the fluid and must not be resolved for non-viscous fluids such as air and, to a degree, water. Note that this is not physically correct as even air exhibits differing viscosity at varying temperatures, yet this effect can be neglected in the context of visual simulations; the effect is minimal. In this case, the Navier-Stokes equation of momentum conservation reduces to the incompressible Euler equation [Fedkiw et al. 2001]:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\cdot\mathbf{u} - \nabla p + \mathbf{f} \tag{7}$$

When viscosity is high and must be accounted for, the diffusion term tends to smooth out the velocity field and reduce the influence of higher frequencies or *wavenumbers* (For a more detailed overview of wavenumbers and their interpretation, see [Stam 2002]). It is therefore straightforward to apply a low pass filter to the Fourier transform of the velocity field which is consistent with the equations of fluid flow. A low pass filter with these properties is given by

$$\mathbf{w}_3(\mathbf{k}) = e^{-\mathbf{k}^2 \nu \Delta t}, \tag{8}$$

where $\mathbf{k}$ is the wavenumber in the Fourier domain.

3.3 Projection

The projection step in the Fourier domain is once again based on *Helmholtz-Hodge Decomposition*. As stated above, the gradient operator in the spatial domain is equivalent to a multiplication by the wavenumber in the Fourier domain. To force our velocity field to be mass conserving we must assure that equation (1) holds: in two dimensions this means that in the Fourier domain the dot product of a velocity vector by the corresponding wavenumber *vector* must result to zero. Geometrically it is simple to see that all velocity vectors in the Fourier domain must be perpendicular to their corresponding wavenumber vectors, so that the dotproduct of the two vectors is zero. This can be achieved by projecting each vector onto the line perpendicular to the wavenumber. The result is the mass conserving field $\mathbf{w}_4$. Section 4 will discuss how I visualize the spatial and Fourier representations of the velocity field and present the decomposition described in this section.

4 Visualization of the Velocity Field Computation

Stam presents some static visuals in his paper [Stam 2002] and also mentions dynamic visualization in his talk on stable fluids. This inspired me to cache the intermediate steps of the Navier-Stokes computation, both in the spatial and Fourier domain, so they can be visualized in real-time during the simulation. The solver presented in [Stam 2002] computes both the diffusion and the projection step in one loop over the entire Fourier domain. To not impair simulation performance, I decided to combine these two effects and discard the idea of adding an extra loop to the code to visualize the two steps explicitly.

In Figure 3 (screenshot taken from the `flowanim` demo) the columns are (from left to right): the velocity field $\mathbf{w}_2$ after the advection has been computed, the final field after a timestep Δt ($\mathbf{w}_4$) and on the far right we see the difference between the two previous fields. Note, that as described above this is only the *Helmholtz-Hodge Decomposition* of the vector field $\mathbf{w}_3$ when the viscosity is very low, that is ν is set to a value near zero. In the example in Figure 3, the viscosity ν is set to 0.001, which results in a slight dampening of the field as can be observed in the Fourier domain (the high frequency values are reduced).

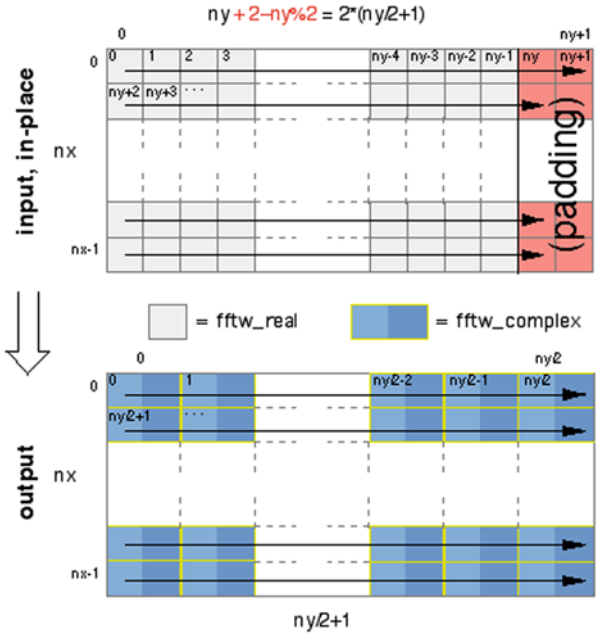

Figure 4: FFTW's *In-place* computation of the fast Fourier transform: the input format as FFTW expects it and how the data is organized after the transform.

The top row in Figure 3 shows the velocity field in the spatial domain, the bottom row shows the equivalent velocity field transformed to the Fourier domain. In the Fourier domain, each transformed value has a real and an imaginary part as the transforms out-

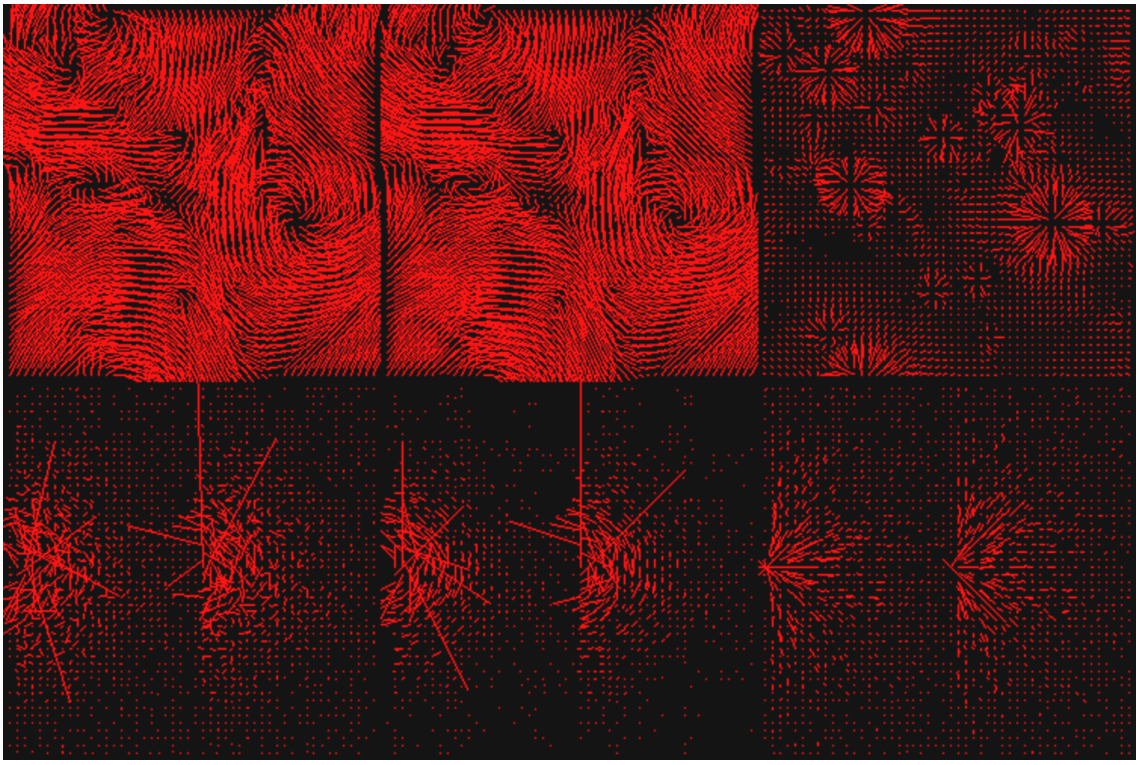

Figure 3: The decomposed velocity field and its Fourier transform: the top row shows the spatial domain while the bottom row is the Fourier transform of the field above it. The fields from left to right: field $\mathbf{w}_2$ after the advection step, the final field $\mathbf{w}_4$ and on the far right the difference field between $\mathbf{w}_2$ and $\mathbf{w}_4$

put consists of complex numbers, stored in FFTW's `fftw_complex` format (a simple `c` struct made up of two `fftw_real` datatypes , which are defined as type `double`). The FFTW stores the transformed data as shown in Figure 4, discarding redundant data from the real-to-complex transform. Each displayed velocity field (in Figure 3) in the Fourier domain consists of the imaginary part (the left half) and the real part (the right half).

Though the effect is not as strong as in Stam's diagrams [Stam 2002], the result of the projection step in the Fourier domain is very well visible. The difference of the two fields $\mathbf{w}_2$ and $\mathbf{w}_4$ (the far right column) shows the divergent gradient field and its Fourier transform with all transformed velocity vectors running parallel to their corresponding wavenumber. We can also nicely see that the vortices in the final field $\mathbf{w}_4$ (center) are nicely localized, which is not the case in the field $\mathbf{w}_2$ on the far left.

Various control methods described in Section 6 allow to pause the simulation, set up initial smoke densities and forces and then dynamically visualize the computation of the final velocity field $\mathbf{w}_4$. This helps in understanding the solving process better and also shows the elegance of this method.

5 Computation and Visualization of Vorticity Confinement

Fedkiw et al. [Fedkiw et al. 2001] extend the method from [Stam 1999] by adding an additional force term to the equation known as *vorticity confinement*. This method was initially invented by Steinhoff [Steinhoff and Underhill 1994] approximately a decade ago for the numerical computation of complex turbulent flow fields around helicopters where it is not possible to add enough grid points to accurately resolve the flow. The idea is to re-inject the energy which is lost by means of nonphysical numerical dissipation. The first order integration scheme used, especially when implemented using linear interpolation, tends to dampen the more interesting features of the flow such as rotational and turbulent structures. So the first step is to locate where these features should be added. These areas are defined as areas with high *vorticity* (ω):

$$\omega = \nabla \times \mathbf{u} \tag{9}$$

In the case described in this report, the result can be interpreted as a vector perpendicular to the (x,y) plane and can be treated as a scalar field computed as $\partial v/\partial x - \partial u/\partial y$ (note, that u,v corresponds to velocities in x,y direction). Each piece of vorticity can be seen as a measure for the "curl" of the velocity field, including the sign for the direction of the curl. Numerical dissipation damps out the curl of the field, so the solution is to add it back by applying a force perpendicular to the gradient field of the vorticity. The normalized gradient field is computed as

$$\mathbf{N} = \frac{\eta}{|\eta|} \qquad (\eta = \nabla|\omega|) \tag{10}$$

The gradient vectors from each grid cell points to the largest slope in the magnitude of the vorticity (the magnitude of the vorticity $|\omega|$ can be interpreted as a heightfield). Now a force is applied perpendicular to this gradient with magnitude proportional to the vorticity

$$\mathbf{f}_{conf} = \varepsilon\,(\mathbf{N} \times \omega), \tag{11}$$

which accounts for the direction of the velocity curl. Fedkiw et. al. [Fedkiw et al. 2001] propose adding an h factor which accounts

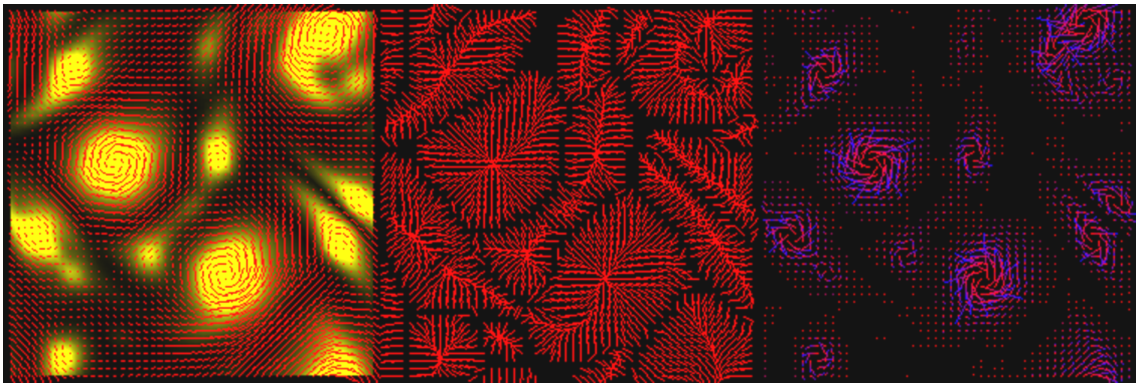

Figure 5: Vorticity confinement computation(from left to right): the velocity field $\mathbf{w}_4$ and the scalar quantity of *vorticity* (yellow), the normalized gradient of the vorticity, the confinement forces

for the spatial discretization and reduces the confinement force $\mathbf{f}_{conf}$ for finer grids. In my implementation I use a dimensionless grid spacing of 1.0 for all grid sizes, so this factor would have no influence at all and is therefore not included in equation 11 (Steinhoff [Steinhoff and Underhill 1994] also neglects this term). The ε factor controls the amount of small scale detail added to the velocity field.

In Figure 5 (screenshot taken from the `flowanim` demo) I display the magnitude of the vorticity $|\omega|$ as a scalar field with intensities of yellow to visualize the heightfield (overlayed by the velocity field $\mathbf{w}_4$). The vector field in the middle shows the normalized gradient field $\mathbf{N}$ and the field on the right shows the vorticity confinement forces, pointing from red to blue. These quantities are animated in real time with the rest of the simulation and can be examined at any time to estimate the proper ε factor for visually pleasing results. The higher the ε factor is set, the smaller and more localized the vortices become. This can diverge for too high ε settings (which is also dependent on the timestep and machine speed) and leads to quite interesting, highly turbulent visuals (Figure 3 was created using the turbulence resulting from a *very high* confinement forces).

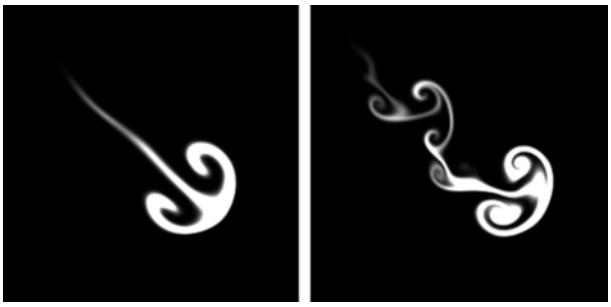

Figure 6: Two frames of animation from two mpeg movies created using `flowanim` and `mpeg2encode`. Both frames depict the 60th frame of the movie. The left animation is created without vorticity confinement, the one on the right with vorticity confinement and a *relatively* high force factor

Figure 6 shows the results of vorticity confinement in animating a simple smoke scene. In this case, the confinement force factor was set to 0.5 (with a timestep of 0.6 and a gridsize of 256×256 on a pentium III 1GHz with GeForce2 GTS), which results in strong turbulence and lots of small scale detail compared to the left image without vorticity confinement. Both animations were created using identical initial force and smoke density configurations and each screenshot resembles the same frame of animation (the 60th frame in both cases).

6 Controlling the Smoke and Making Movies

The control in my demo `flowanim` is keyboard/mouse driven. The easiest way to set up a good control scheme is to pause the simulation, inject matter and forces into the static grid, save the initial configuration and then play it back, altering the settings for timestep, viscosity, dissipation and confinement force until the desired smoke animation is achieved. I have tried this with varying grid sizes, and on my test machine (a pentium III 1GHz with GeForce2 GTS) the framerates are still interactive up to a grid size of 256×256 with a framerate of approximately 3 frames per second (this is only possible when all other visualization windows for the velocity field and vorticity computation are disabled). The stills in Figure 7 are all created using the method described above.

To create mpeg movies of the smoke animations I have added a screenshot functionality to `flowanim` which flushes out a sequence of `ppm` images. These I then feed to `mpeg2encode` [1] to create an mpeg video sequence. Additionally, each still `ppm` image wraps and can therefore be used as a seamlessly tiling texture for an arbitrary mesh.

7 Results and Possible Extensions

The animation stills in Figure 7 show the increased visual realism added by tweaking the amount of vorticity confinement force added. But even if realism is not what the animator desires, by setting the confinement force factor to values beyond simulation stability, the field tends to exhibit an unnatural yet visually interesting amount of turbulence which can still be controlled by the animator (to a certain degree). In Figure 7 this effect is already introduced in the far right column where, physically, "too much" small scale detail is added, leading to a more "comic" like effect (which can be interesting and useful in some cases).

Due to certain time constraints and the reduced scope of this project I didn't implement some interesting extensions to the software. Some of these are

[1] http://www.mpeg.org/MPEG/MSSG/

- *Liquid textures*
  Advect texture coordinates as in Stam's original paper [Stam 1999]. This method can easily produce very interesting wrapping textures which can be used to texture map arbitrary meshes.

- *Addition of particles*
  An interesting idea, which is already part of the fluid surface trace in [Foster and Metaxas 1996], would be to add inertialess particles to the system and advect them through the flow.

- *Mixing different smoke colors*
  In [Chen and Lobo 1995] the authors experiment with mixing different colors of fluid. It would have been interesting to advect various different smoke colors and examine the visual effect. This extension should be very straightforward to implement.

- *Extension to 3D and volume rendering*
  Fedkiw et. al. [Fedkiw et al. 2001] use Jensen's *Photon Map* for their offline renderer and a hardware based renderer also outlined in [Fedkiw et al. 2001]. It would have been interesting to implement the entire solver in three dimensions and render the result in hardware. This extension is obviously beyond the scope of this project.

A yet unsolved problem in the field is the more precise control of the fluid flow, such as giving a final situation and computing the necessary initial conditions (densities, forces) to generate this situation. This is mainly due to the nonlinearity of the Navier-Stokes equations. I personally find this to be an interesting area for further research.

All videos presented in this report, the source code and the precompiled binaries for win32 and linux are available on my website at `http://www.nealen.com/projects`.

8 Acknowledgements

The codebase I used to create `flowanim` was originally invented and written by Jos Stam [Stam 2002]. The extension to convecting densities and rendering them using OpenGL's `GL_TRIANGLE_STRIP`'s and `glut` was written by Gustav Taxen (`http://www.nada.kth.se/~gustavt/fluids/`).

References

CHEN, J. X., AND LOBO, N. V. 1995. Toward interactive-rate simulation of fluids with moving obstacles by navier-stokes equations. In *CVGIP: Graphical Models and Image Processing*, 107–116.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *SIGGRAPH 2001 Conference Proceedings*, 15–22.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proceedings of SIGGRAPH 2001*, 15–22.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graphical Models and Image Processing 58*, 5, 471–483.

FOSTER, N., AND METAXAS, D. 1997. Controlling fluid animation. In *Proceedings of Computer Graphics International (CGI'97)*.

FOSTER, N., AND METAXAS, D. 1997. Modeling the motion of a hot, turbulent gas. In *Computer Graphics Proceedings, Annual Conference Series*, 181–188.

FRIGO, M., AND JOHNSON, S. G. 1998. Fftw: An adaptive software architecture for the fft. In *ICASSP conference proceedings*.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *Computer Graphics Proceedings, Annual Conference Series*, 49–55.

LAYTON, A. T., AND VAN DE PANNE, M. 2002. A numerically efficient and stable algorithm for animating water waves. *The Visual Computer*.

STAM, J. 1999. Stable fluids. In *SIGGRAPH 99 Conference Proceedings*, 121–128.

STAM, J. 2002. A simple fluid solver based on the fft. *Journal of Graphics Tools 6*, 2, 43–53.

STEINHOFF, J., AND UNDERHILL, D. 1994. Modification of the euler equations for vorticity confinement: Application to the computation of interacting vortex rings. *Physics of Fluids 6*, 8, 2738–2744.

A Controls in the `flowanim` demo (quick reference)

Key	Function
z	Set velocities to zero
a	Set smoke densities to zero
c	Switch between smoke and vorticity display in main window
shift-c	Enable/disable vorticity confinement forces
u	Toggle display of velocity field in main window
w	Draw scalar fields as wireframes
t	Decrease timestep by 0.01
shift-t	Increase timestep by 0.01
v	Decrease viscosity by a factor of 10
shift-v	Increase viscosity by a factor of 10
d	Decrease dissipation factor by 0.001
shift-d	Increase dissipation factor by 0.001
s	Decrease injected matter at mouse location by 1.0
shift-s	Increase injected matter at mouse location by 1.0
o	Decrease vorticitiy confinement force factor by 0.01
shift-o	Increase vorticitiy confinement force factor by 0.01
q	Exit the simulation
p	Pause the simulation
r	Reset the simulation
f	Hide/show velocity field decompostion and FFT
F	Hide/show vorticity confinement computation
n	Save the current simulation state in memory
shift-n	Restore a saved state from memory

Start the demo by typing
`flowanim [gridsize]`
on the command line. `flowanim` defaults to a gridsize of 64 when no gridsize is entered. Gridsizes must be even, but if uneven gridsizes are entered, the software corrects for this. Dragging the mouse in the main window and holding the left mouse button down adds smoke and force to the field. by pressing 'p' the simulation can be paused/unpaused. When paused, it is possible to set up an initial situation, which can also be saved by pressing 'n' (the saved state can be restored by pressing 'shift-n').

All tweakable parameters of the simulation are accessible to the user via keypresses. Note, that tweaking these values is extremely machine dependent, as the demo does not monitor its own performance. Most of the keys listed above are also accessible via the `glut` menu when right clicking the main window.

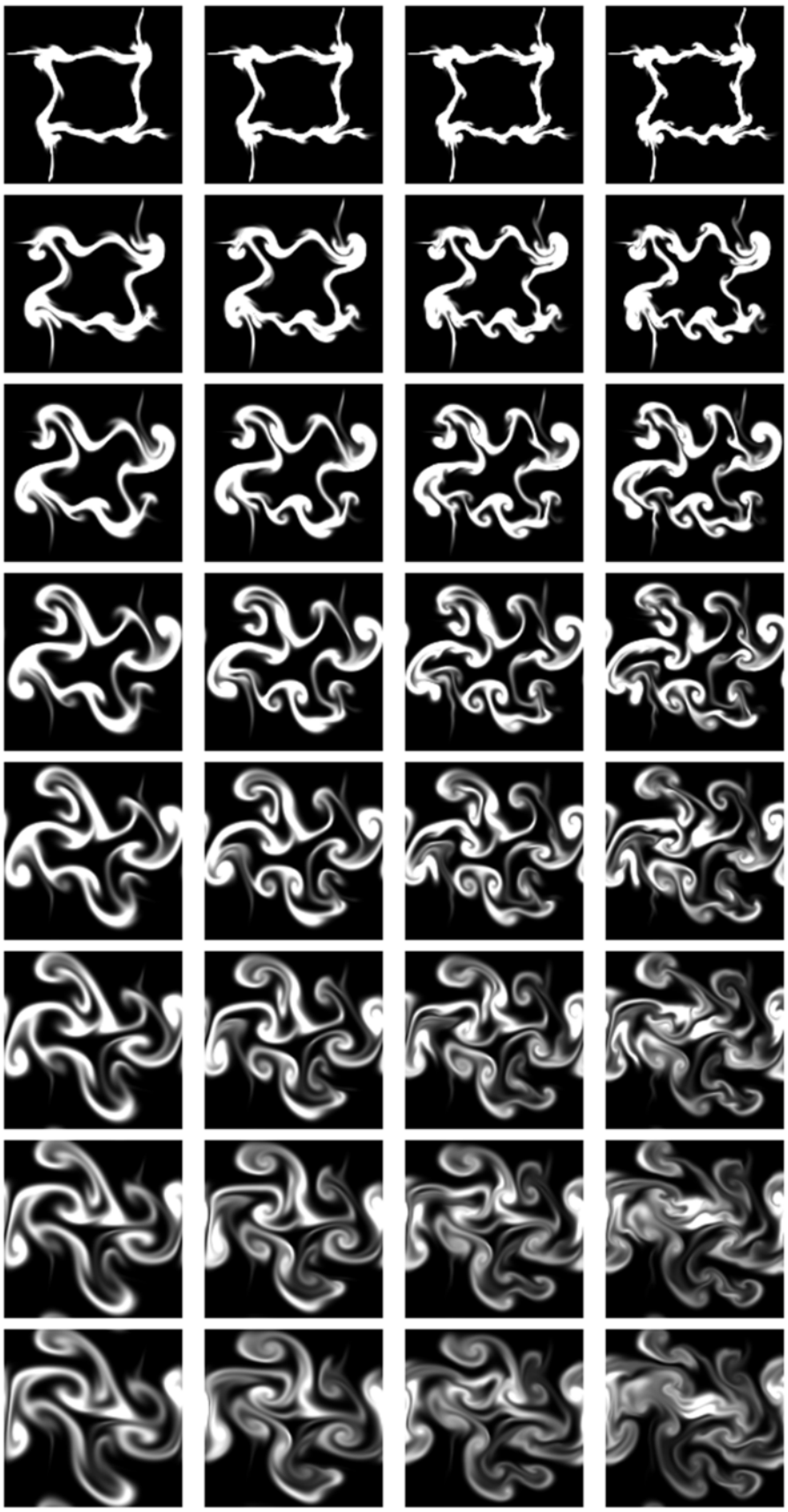

Figure 7: Some examples. Eight frames from four independent animations, spaced 10 frames apart (from top to bottom). Each column resembles an animation with a different confinement force factor (from left to right): 0.0 (no vorticity confinement), 0.15, 0.30, 0.45. All animations were created on a Pentium III 1Ghz with a GeForce2 GTS with grid size of 128×128, timestep 1.0, viscosity 0.001 and dissipation 1.0