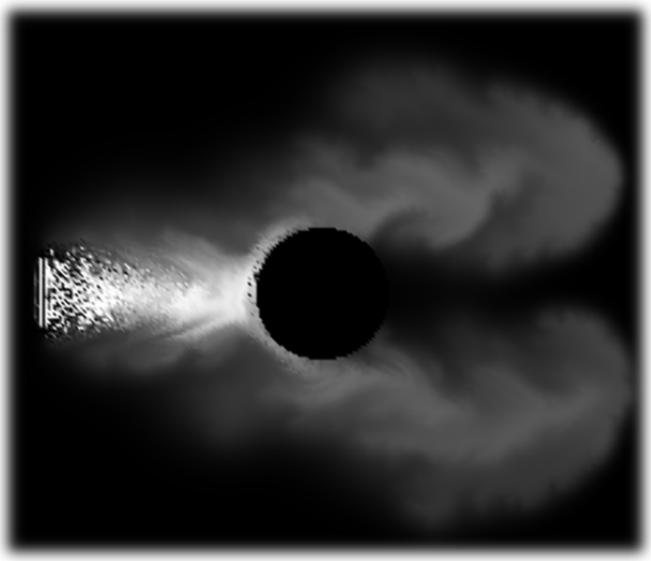
Real-Time Interactive Smoke Simulation

Luis Valverde*
MSc in IET, Trinity College Dublin



Abstract

This paper describes the design and implementation of a CPU-based fluid simulation system that allows interactive frame rates up to 512x512 grid resolutions in a standard PC. The simulation is based on the application of the Navier-Stokes fluid dynamics equations for calculating the evolution of the velocity field over the fluid and other quantities carried by it like the density of suspended particles. An unconditionally stable method is used to solve the equations that allows for arbitrarily large time steps in the simulation. Improvements over the base simulation method have been implemented covering the simulation of internal moving boundaries of any shape, application of vorticity confinement forces to recover details lost due to numerical dissipation and parallelization of the more computationally expensive steps of the algorithm.

CR Categories: J.2 [Computer Applications]: PHYSICAL SCIENCES AND ENGINEERING—;

Keywords: computational fluid dynamics, Navier-Stokes equations, stable fluids, vorticity confinement, smoke

1 Introduction

Fluids like water, fire or smoke are elements present in many environments and forms like rain, burning torches, exhaust fumes, rivers, mist, clouds, etc. Therefore, applications that aim to create an immersive representation of these environments, like videogames, must be able to simulate fluids with realism while keeping the computational cost low enough for the requirements of a real-time application running in desktop PCs. Particle systems have been used for the purpose as they provide a fast way for producing visually attractive results but often lack the realism obtained with physically based methods.

*e-mail: valverl@tcd.ie

Physically based methods usually employ the mathematical model for fluids defined by the equations of Navier-Stokes that provide a precise description of the behavior of many fluid flows present in nature. Unfortunately, these equations can only be solved analytically in very simple cases what limited their applicability for many years. Numerical solutions became possible with computers but their complexity made them slow to compute, restricting the use of this model to offline simulations that required highly realistic results like airplane engineering.

The simulation method used in this project was developed by Jos Stam in [Stam 1999] who employed the fact that in computer games and other interactive simulations what matters most is not high precision but visual believability and speed to develop an stable method for solving the fluid equations. Stable in this context means that arbitrarily large time steps can be taken in the simulation without degrading the results. This provides a faster way of computing a physically based simulation than traditional numerical solutions and improves the visual realism with respect to particle systems.

2 Related work

Instead of the seminal work of Stam about Stable Fluids [Stam 1999] a more recent paper by the same author was used for the development of the basic simulation [Stam 2003]. This paper was presented in 2003 Game Developers Conference (GDC) and explains the stable fluids simulation method step by step targeting game developers that want to create a real-time interactive fluid dynamics solver. The paper provides a detailed description of the algorithm together with a full implementation in C. Although published in 2003 the simulation method is basically the same as the one presented in 99 paper.

Internal boundaries handling method and vorticity confinement force calculation are based on the ideas presented by Fedkiw et al. in [Fedkiw 2001]. This paper proposes a simulation method based

on the stable fluid solver but specialized on smoke. They make the assumption that the effects of viscosity are negligible in gases especially on coarse grids where numerical dissipation dominates physical viscosity and molecular diffusion. This allows them to skip the diffusion step in the simulation without visually noticeable effects in most scenarios. My simulation allows to skip the diffusion step by setting the diffusion and viscosity terms to zero. Another modification introduced by Fedkiw is the use of a staggered grid for the velocity field where velocities are registered in each cell face instead of in the cell center, reducing the effects of numerical dissipation. They advect as well the temperature over the fluid and add forces to account for the effects of gravity on smoke particles and buoyancy due to temperature. The staggered velocity grid, temperature advection and physically based forces were note included in my simulation due to time restrictions but could be easily added in the future.

Finally, [Harris 2005] was used as a complimentary source for the mathematical development and understanding of the fluid dynamics equations. This article describes the implementation in the GPU of Stam's stable fluid solver together with the improvements proposed by Fedkiw. Unfortunately, it was not possible to attempt the implementation described here due to the time constrains of the project and my lack of experience in general purpose GPU programming.

3 Simulation method

The simulation space is a squared region that is divided in a uniform grid cells. Each of the properties carried by the fluid -velocity and density in our case- are sampled on the cell centers. This is the Eulerian approach to simulation. An alternative that is not explored here is using sample points that move freely about the fluid, what is called a Lagrangian formulation - see [Lee 2008] for a more extensive discussion on the topic. These are the Navier-Stokes equations that are used to update the velocity and density:

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

$$\frac{\partial \rho}{\partial t} = -(\mathbf{u} \cdot \nabla)\rho + \kappa \nabla^2 \rho + S \tag{2}$$

The current state of the fluid can be modeled with a continuous function that assigns a velocity vector to each point in space in a given instant of time, i.e. a velocity field. The first equation describes how this field changes over an infinitesimal time step. Besides the velocity, we need to consider the position of each smoke particle but there are so many that it is too computationally expensive to register each of them, so we have to use a density function that gives a density value for each point in space. The evolution of this density field through time is modeled by the second equation.

This are the steps followed to update the velocity:

1. Calculation of vorticity confinement force
2. Application of forces
3. Diffusion
4. Projection
5. Advection
6. Projection

The procedure to update density is similar but without the vorticity and projection steps and adding density sources instead of forces. In the simulation the velocity is updated first and then the density using the updated velocity. Each velocity update step will be described briefly in the following sections.

3.1 Vorticity confinement force

Vorticy is a way to measure the amount of rotational flow in any point of a fluid. The vorticity ω in any point of the fluid is obtained taking the curl of the velocity $\mathbf{u}$ in that point:

$$\omega = \nabla \times \mathbf{u} \tag{3}$$

The next step is to calculate the normalized vorticity location vector $\mathbf{N}$ in the following way:

$$\mathbf{N} = \frac{\eta}{|\eta|} \qquad (\eta) = \nabla|\omega| \tag{4}$$

And finally the confinement force $\mathbf{f}_{conf}$ is computed for each cell center as:

$$\mathbf{f}_{conf} = \epsilon h(\mathbf{N} \times \omega) \tag{5}$$

Where ϵ is used to control the amount of small scale detail added back to the flow field and h is the cell side.

3.2 Application of forces

The forces are applied multiplying their value by the time step and adding the result to the initial velocities.

3.3 Diffusion

Diffusion represents the interchange of value (velocity, density, etc.) between a cell and its neighbors. It corresponds to the term $\nu \nabla^2 \mathbf{u}$ in the equation for the velocity. The diffusion of the velocity in a cell (i, j) is calculated finding the velocity that diffused backward in time yields the initial velocities. This translates to the following formula:

$$\mathbf{u}^0_{i,j} = \mathbf{u}^t_{i,j} - \nu(\mathbf{u}^t_{i-1,j} + \mathbf{u}^t_{i+1,j} + \mathbf{u}^t_{i,j-1} + \mathbf{u}^t_{i,j+1} - 4\mathbf{u}^t_{i,j})) \tag{6}$$

Applying this formula to every cell gives us a system of N^2 equations (N = grid side) with the same number of unknowns that is solved using a Gauss-Seidel iterative solver.

3.4 Projection

After the velocity has been diffused it rarely conserves mass. The projection step uses the Helmholtz-Hodge decomposition theorem that applied to this context states that the velocity field can be decomposed in a mass conserving field and a gradient field. Therefore we can get a mass conserving field computing the gradient field and subtracting it from the current velocity field. To compute the gradient we first calculate the velocity divergence for each cell in the following way:

$$\nabla \cdot \mathbf{u}_{i,j} = -0.5h(\mathbf{u}_{i-1,j} - \mathbf{u}_{i+1,j} + \mathbf{u}_{i,j+1} - \mathbf{u}_{i,j-1}) \tag{7}$$

From the velocity divergence we can compute the pressure field p:

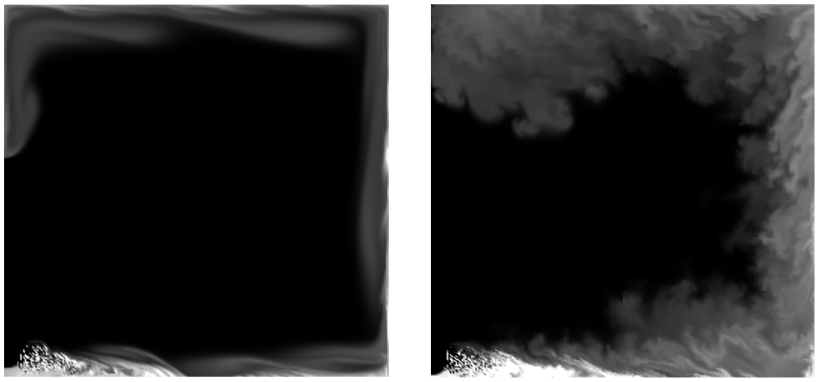


Figure 1: *Smoke flow without (left) and with (right) vorticity confinement*

$$p_{i,j} = ((\nabla \cdot \mathbf{u})_{i,j} + p_{i-1,j} + p_{i+1,j} + p_{i,j-1} + p_{i,j+1})/4 \quad (8)$$

This yields to a system of linear equations that is solved using again the Gauss-Seidel iterative solver. The last is step is computing the gradient of the pressure ∇p and subtracting it from the current velocity field:

$$\nabla p_{i,j} = (0.5(p_{i+1,j} - p_{i-1,j})/h, 0.5(p_{i,j+1} - p_{i,j-1})/h) \quad (9)$$

$$\mathbf{u}_{i,j}^{new} = \mathbf{u}_{i,j} - \nabla p_{i,j} \quad (10)$$

3.5 Advection

The advection step corresponds to the term $-(\mathbf{u} \cdot \nabla)\mathbf{u}$ in the velocity update equation and accounts for the effect of the fluid velocity on all the quantities carried by the fluid, including the same velocity. Velocity advection can be seen as the tendency of the velocity field to move with itself. To compute advection a semi Lagrangian scheme is used where particles placed in the center of each cell are traced backwards in time from the end of the time step to the current state. The values of the four closest cells (a, b, c, d) to the particle position obtained this way are linearly interpolated to obtain the value of the cell corresponding to the future position.

$$\mathbf{u}_{i,j}^{new} = weight_a \mathbf{u}_a + weight_b \mathbf{u}_b + weight_c \mathbf{u}_c + weight_d \mathbf{u}_d \quad (11)$$

3.6 Boundary conditions

There is an extra layer of cells on each side of the simulation grid to account for the external boundary conditions. The density in this cells is set to that of their closest non-boundary neighbor to avoid loss of density through diffusion. The velocity in the direction normal to the boundary is set to be equal in modulus and opposite in direction to the closest non-boundary cell. This forces the fluid to stay in the simulation domain. The tangential component of velocity at the boundaries can be set either equal to the tangent component of the corresponding neighbor, creating a frictionless surface, or less than it, creating a resistance to flow tangent to the boundary.

The internal boundary conditions are handled in a similar way. An array of occupied cells representing an object of arbitrary shape moving inside the fluid indicates what internal cells are occupied. The density at each of these cells is set to that of the neighboring non-boundary cells if any or to zero if it is surrounded by occupied cells. The velocity is set to the objects one.

4 Simulation application

The simulation application was developed in C++ with OpenGL/Glut. It was written from scratch but based largely on the reference application developed by Stam. The main differences with the later are the use of vector data types (Wild Magic 4 library ones) to ease the extension to 3 dimensions -although finally there was no time to carry out this extension- and the use of templates to generalize algorithms applicable to both scalar and vectorial data like the advection step. Another reason to rewrite Stam's code was to gain the necessary insight on the algorithms to perform the intended improvements.

Besides the application of vorticity confinement forces and internal moving boundaries explained in the previous sections, the application was extended by multi-threading some of the fluid solver steps. The application creates a configurable number of worker threads in the initialization stage that remain waiting until a work order is sent to them. Each worker thread has an associated semaphore that is used by the control thread to signal the availability of a new work order. Once the semaphore is set the worker thread awakes and reads the type of order to be executed from a variable accessible to all threads. The orders can be: calculate divergence, project velocity, calculate pressure and solve linear system. The same order is sent to all worker threads but they that work on different data sets given by their thread IDs. Once the order has been completed the worker thread increments a semaphore common to all threads. The control thread is waiting on this semaphore counting the number of times it has been raised. When the count reaches the number of worker threads the control thread knows the order has been processed and can continue to the next step of the simulation.

Two data set partitioning strategies were tested. In the first one the data set was divided in n sequential parts -n being the number of worker threads- that were assigned to the corresponding thread according to their ID. The second strategy tried to take advantage of the cache increasing spacial coherency in memory access by sequentially assigning to each thread one out of n data, i.e. the first of 4 threads works on the data in positions 0, 3, 7, etc., the second works on 1, 4, 8, etc. and so on. This second strategy did not yield better results than the first one, maybe because the CPU slots assigned to each thread were long enough to provoke cache trashing.

Figure 2: *Interaction of the fluid with arbitrary shaped moving internal boundaries*

5 Results

It is difficult to quantify the result of the application of vorticity confinement. It does provide additional detail to simulation by adding nice swirling movement in the fluid with a reasonable increase in the simulation time (around 16% of the total step time). The effect achieved is not necessarily more realistic but in any case it gives more possibilities to the animator/designer to play with.

Internal boundaries interact coherently with the surrounding fluid creating trails and transmitting their velocity to neighboring areas. Although the implementation of this approach is quite straightforward in 2 dimensions the promotion to 3 dimensions would add the difficulty of voxelizeing the objects in the fluid.

Skipping the diffusion step as suggested by Fedkiw saves an important amount of processing time (over 50%) and provides visually plausible results in the smoke simulation. Still in some scenarios, specially when the velocity is low, the lack of diffusion can lead to very unrealistic results.

Finally, multithreading of key steps in the simulation provides notable performance gain, specially in the velocity projection where it performs over 3 times faster. Figure 3 shows a comparison between the single-threaded and multi-threaded implementation running in a 256x256 grid without vorticity and diffusion. As it can be seen the velocity projection takes an average of 29.4 ms in the single-threaded implementation (almost 83% of the total time) while in the multi-threaded (4 threads) takes 9.6 ms reducing the total time over a 65%. These tests were performed in a desktop PC with an Intel Xeon W3520 (4 cores) running at 2.63GHz and 3 GB of RAM.

Unluckily, it was not possible to get the Harris GPU implementation working and no there were no performance results in his article thus making impossible the comparison with our simulation.

Single Thread		Multi-Thread	
Step	Time (ms)	Step	Time (ms)
density source	0.4197	density source	0.4285
density diffusion	0.0001	density diffusion	0.0001
density advection	2.1236	density advection	2.1471
total density	2.544	total density	2.5764
velocity vorticity	0.0002	velocity vorticity	0.0002
velocity forces	0.4063	velocity forces	0.404
velocity diffusion	0.0001	velocity diffusion	0.0001
velocity advection	2.6412	velocity advection	2.6841
velocity projection	29.4092	velocity projection	9.6236
total velocity	32.4584	total velocity	12.7133
total update	35.003	total update	15.2901

Figure 3: *Performance comparison between the single-threaded (left) and the multi-threaded (right) solver versions*

6 Conclusions

The stable solver for fluid dynamics proposed by Stam and the extensions added by Fedkiw have proved to give visually plausible results for interactive smoke simulation. Simulation times around 15 ms per step in a 256x256 grid can be achieved with a multi-threaded implementation running in a multi-core desktop PC, allowing for full screen rendering with resolutions up to 1024x1024 with reasonable smoothness. Another interesting finding is the highly parallelizable nature of the algorithm. This fact together with the locality of the dependencies between cells and the absence of branching instructions makes the algorithm a perfect candidate to benefit from the parallelizing capacities of modern GPUs. It is not unreasonable to speculate that a 3D implementation of the algorithm could run in real time in today's GPUs.

Future work directions could be the implementation of a 3D version of the solver on the GPU or the addition of the remaining improvements proposed by Fedkiw -staggered velocity grid, temperature advection and physically based forces.

A video showing the simulation application at work can be found in: http://www.youtube.com/watch?v=LU7w87bD6rM

References

FEDKIW. 2001. Visual simulation of smoke. *SIGGRAPH 2001 Conference Proceedings, Annual Conference Series.*

HARRIS, M. 2005. Fast fluid dynamics simulation on the gpu. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, ACM, New York, NY, USA, 220.

LEE, R. J. 2008. *Multiresolution Fluid Simulation.* PhD thesis, Department of Computer Science, Trinity College Dublin.

STAM, J. 1999. Stable fluids. *SIGGRAPH 99 Conference Proceedings, Annual Conference Series.*

STAM, J. 2003. Real-time fluid dynamics for games. *Proceedings of the Game Developer Conference, March 2003.*