15-464 Technical Animation
Final Project Report

Evan Shimizu (eshimizu) and Conrad Verkler (cverkler)

Goals

Our goal is to explore fluid simulations through attempting to build a version of Nguyen et. al.'s Physically Based Modeling and Animation of Fire paper. This paper uses a modified version of the Stam fluid simulation. We hope to learn more about how fluid simulations work by starting with a 2D Stam simulator, then adding in another dimension and going to a 3D Stam solver, and finally adding in the techniques detailed in the Nguyen paper.

Simulation Techniques

We developed a 2D Stam fluid simulator based on the techniques and code outlined in the original paper. We followed along with the paper, using all the methods outlined for advection of the fluid and velocities, diffusion of densities and velocities, and enforcing conservation of mass by subtracting the gradient field from the current velocities. Stam's diffusion and advection steps maintain stability by finding values which when acted on in reverse, yield the initial values. Advection traces the values back in time linearly, while diffusion solves a linear system to get the future values. Both techniques are fairly straightforward, with the most complicated part being the linear equation solver. We used the default Gauss-Seidel relaxation technique for solving the Poisson equations in the paper.

Next we moved on to developing the 3D version of the simulator. We added an additional velocity component array to hold the additional values, and proceeded to modify the equations to handle the additional parameter. Advection and diffusion didn't take too much time to convert, though the C-style programming in the original paper caused some confusion when we got to the project

function. The issue was one of memory storage and what values got overwritten and what got reused, and once that was straightened out, the modifications to make project work properly didn't take very long. We still used the Gauss-Seidel relaxation solver in the 3D version.

Transitioning from 2D to 3D lost most of the curls that the fluid made, probably due to that energy being dispersed over the additional dimension and the difficulty in seeing details in 3D. To give fluid a more realistic feel we added vortices described in [6] and in more detail in [2]. This calculation first computes the curl of the velocities $\omega = \nabla \times u$ where u are the velocities. We then calculated $N = \nabla|\omega|/|\nabla|\omega||$. The force on the fluid is $F = \varepsilon h\,(N \times \omega)$ which is applied the same way [7] applies forces. To change the strength of the vortices we changed the value of ε. This is probably not really what this value should be used for but it was the easiest way to alter the animation. Setting it to some small value made no real change to the velocities while making the value too big would make the simulation explode.

To calculate the curl on the velocity field we crossed the gradient at a cell with the original velocity. We only do a rough calculation of the gradient that is straightforward and fast but is not very accurate. The gradient for the x axis is calculated by taking the difference of the values of the neighboring cells in the x direction and dividing by 2 times the length of the a cell, the other axes are calculated the same way.

Rendering

For rendering, we adapted code from Stam's original demo of the fluid simulator, which included drawing the densities and velocity field. For densities, the code drew each cell as a quad and interpolated densities to get the color at the corners, allowing OpenGL to interpolate smoothly across the cell. The velocity field was a series of GL Lines that had their ending position

displaced by the magnitude of the x and y coordinates.

Rendering the 3D version of the simulator took a lot more work. The velocity field rendering translated well into the 3D code but the density rendering required research into volumetric rendering techniques. The rendering code in our project is a combination of a rendering technique from a 2001 paper titled “Visual Simulation of Smoke” by Ronald Fedkiew and Jos Stam et. al. [2] and from a tutorial on volumetric rendering techniques [1].

Fedkiew outlined a method to simulate the self-shadowing of smoke by sending a ray from the light source through the smoke volume and calculating transparency and radiance of each cell, and using those values to render to the screen. The transparency of each ray is initially set to 1: $T_{ray} = 1$. The transparency of each cell is $T_{vox} = exp(-C_{ext}h)$ where h is the grid spacing. Using the cell’s transparency, we calculate the radiance: $L_{vox} = \Omega L_{light}(1 - T_{vox})T_{ray}$. After doing that, the ray transparency is adjusted: $T_{ray} = T_{vox}T_{ray}$. These values are then converted to 1-byte RGBA values and put in a 3D GL texture. The variables for this part of the rendering calculations were adjusted until we were satisfied. We ended up with $C_{ext} = 1,\ \Omega = 0.98,\ L_{light} = 10$. For simplicity, our simulation puts a directional light directly above the fluid area.

After the 3D texture is created, the next stage of rendering samples that texture at various depths and renders the slices on the screen one on top of the other. We use 3D texture coordinates to simplify this process. The slices sampling the volume are rendered as GL Quads and have z-texture coordinates that correspond to the depth of the plane. To achieve different viewing angles, we used the GL_TEXTURE matrix to rotate the texture coordinates appropriately. We used 400 samples (quads) in our final rendering code. Increasing the number of samples drastically increased the render time for the simulation.

Results & Discussion

Results are in the videos handed in with this report.

The 2D results from this project were exactly as expected. They looked like every other 2D implementation of the Stam fluid simulator. The 3D simulator was a bit more difficult to judge, as we haven't seen very many examples of it. We were a bit mystified by the absence of the curls in the 3D version seen in the 2D version, but other parts of the simulation looked correct. Adding the vortices back in helped add back in those curls. The 2D simulator ran around 30-40 FPS and the 3D simulator ran around 20-30 FPS on a 2.7 Ghz i7 processor. The code was not multithreaded.

Because of the added intensity of calculating the 3D simulation the grid was relatively coarse and the cells were often visible, which didn't happen in the 2D. According to [2] adding in vortices merely tried to account for inaccuracies from having too coarse a grid. It would have been nice to either improve the calculation speed or developed a system to render a video instead of displaying it in real time so we could try a much finer grid and see how it improved.

Problems and Solutions

Some of the problems encountered were due to typos or resulted from an incomplete understanding of the algorithm being implemented. Having a second person double check code was invaluable in tracking down the bugs due to typos. One of the most common problems we ran into was not having a full understanding of the underlying mathematics in the fluid simulation. This required us to partially relearn Calc III and to get familiar with concepts such as the divergence, gradient, curl, Poisson equations and others. This resolved mostly by reading

enough reference materials to the point that we could implement something close to what was being described. There were some problems with rendering such as not accounting for the self shadowing, which made the simulation difficult to actually see at times. This was fixed by researching the rendering techniques in Fedkiw's 2001 paper [2].

There were also some issues with calculating the gradient, with the final solution detailed above. We weren't exactly sure how to go about calculating it as almost all references online described calculating the gradient algebraically instead of numerically, as we would have to do. The original website we used as a guide to this calculation gave a poor result and using formulas from [5] we got a simple, working gradient.

We also felt that the vortices went the wrong direction- that they were going inward instead of out. Reviewing the code didn't reveal any small errors so in the spirit of animation and ignoring physics we added a negative to one of the terms and it produced a better looking result.

If We Had More Time...

The Fire paper proved to be more complicated than expected. It would require much more time to get an implementation of the paper functional, not to mention understanding it in the first place. We had hoped to implement the rendering code from the fire paper, but again that proved to be more complicated than expected and would require research into ray marching and related techniques from a few prior papers. We did want to explore the incompressible flow part of the Fire animation paper but couldn't figure it out in time. Given more time we would figure this out and implement it. If we were to do the project again and change something, we'd give the velocity field arrays better names and make the simulator more C++ like than C-like.

We also did not include a dragon in the final demo as stated in the project pitch. Given more time we would rectify this oversight.

Resources

We ended up consulting many more resources than expected. There are a few of note. The Intel series on Fluid Simulation for Games, which currently is up to 15 parts, has a very good introduction to fluid simulations in parts 1 and 2 and nicely explains the math behind them. “Visual Simulation of Smoke” was where the vortex confinement technique came from, and was interesting as an extension of Stam’s previous work. While looking at the Fire paper, we concluded that if someone were to fully implement that paper, a strong understanding of the prior work and mathematics is critical to implementing the paper.

[1] dvnOye. 2012. Getting Started with Volume Rendering. http://www.codeproject.com/Articles/352270/Getting-started-with-Volume-Rendering

[2] FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual Simulation of Smoke.

[3] FOSTER, N., FEDKIEW, R. 2001. Practical Animation of Liquids.

[4] GOURLAY, M. J. 2012. Fluid Simulation for Video Games (Part 1). http://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-1/

[5] GOURLAY, M. J. 2012. Fluid Simulation for Video Games (Part 2). http://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-2/

[6] NGUYEN, D. Q., FEDKIW, R., JENSEN, H. W. 2011. Physically Based Modeling and Animation of Fire.

[7] STAM, J. 2003. Real-Time Fluid Dynamics for Games.