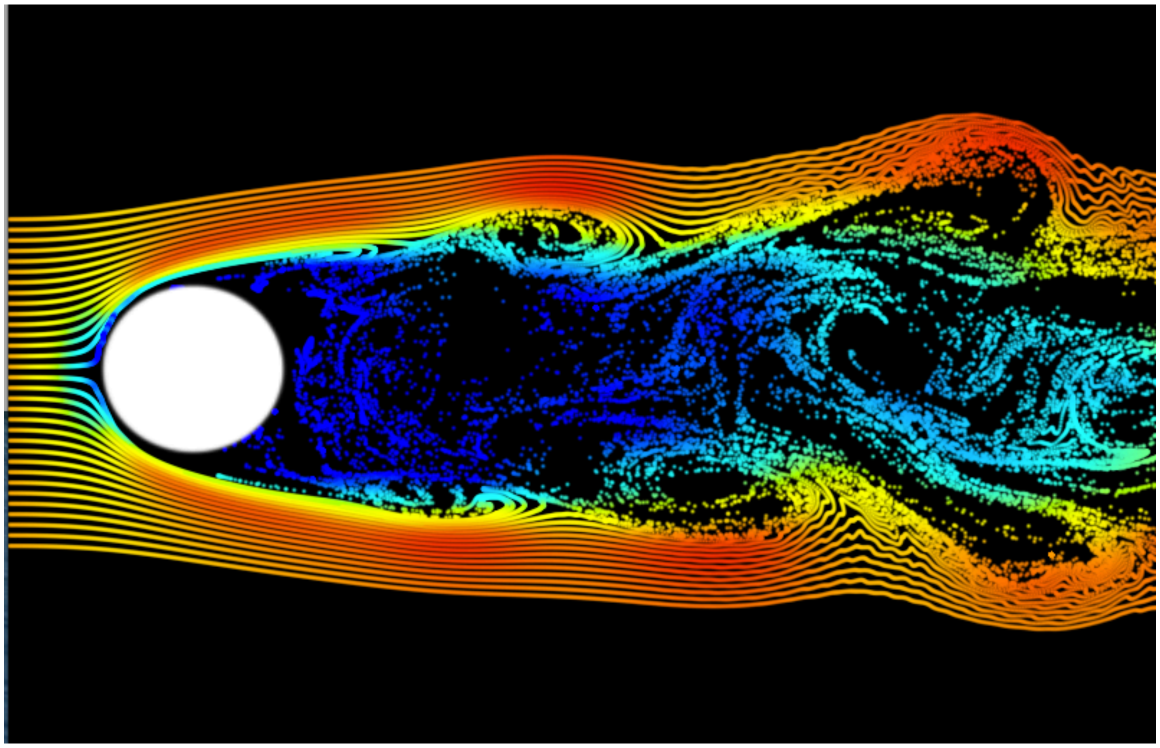
Parallelization and Fluid Visualization

David Futscher

Abstract

Over the past couple of years processor speed as been progressing slower and slower. As we have realized that we cannot push more and more power at our computers and hope they run faster we have changed focus from single core processors to multicore. With the advent of Graphics Processing Units (GPUs) it has become quite clear that massive speed-ups can be achieved by taking advantage of embarrassingly parallel problems. One such area where parallelization can be used is in fluid dynamics and its subsequent visualization techniques.



Introduction

Fluid dynamics is an inherently parallel problem, the calculations are generally done on a large grid solving the Navier–Stokes equations or the Boltzmann equations to advance a velocity field. The clear choice for these simulations is to process them on the GPU. However in if the simulation is in real time and running on the GPU this means that the data can only be accessed on the GPU, its much too expensive to transfer massive amounts of data from the GPU to the CPU to do processing then send more data back to the GPU for rendering.

My main goal when starting this project was to visualize a fluid simulation in real time, however it became quite clear when trying to simulate a 128 by 128 grid of fluid on the CPU that it was just not nearly fast enough for what I wanted to do. In three dimensions the problem was much worse. This lead me to the conclusion that I would have to do all of the calculations on the GPU and the subsequent visualizations techniques that I would like to perform on my simulation would also need to be running on the GPU. The following were the goals I had in place when starting my project.

- Create a fluid dynamics simulation running on the GPU to take advantage of the parallel nature of the problem.
- Develop some of the more basic techniques for visualizing data on the GPU
- Have my simulation run in real time and support a variety of hardware

This paper will discuss how I set up an environment to do parallel Processing with OpenGL 3.3 creating a simulation that supports older hardware as well as discussing the variety of techniques that are possible on the GPU to visualize the flow of fluids.

Technical Details

This project relies on doing massively parallel calculations on the GPU. This required me to pick one of the available frameworks that would give me access to the GPU. I first tried using OpenCL which is designed for doing GPU computations which was fairly nice and worked quite well. Unfortunately it was a pain setting up a work group to use with OpenGL and getting it to run cross platform easily would be a big pain.

Once I decided that OpenCL wouldn't really work for my needs, which was supporting older hardware and being as cross platform as possible I spent some time researching general purpose GPU computing in OpenGL 3.3. Using shaders it turns out that one can simply render the results of calculations in fragment shaders to floating point textures using a FrameBufferObject. Since all operations are completely independent of each other there was no need to use any of the extra features that packages such as OpenCL that allow more inter group communication and some other nice features.

The basic idea is as follows, you create a shader that does the calculations you want. You use textures as inputs; each pixel represents one point in a field. The shader does some sort of calculation and the output is rendered into a texture using a FrameBufferObject. This is used extensively throughout the project from the simulation to the particle system.

The base of the simulation is the one introduced by Jos Stam [1,2]. In essence it uses these operations to advance a velocity field.

Add forces

Advect

Project

The advection step uses a semi-lagrangian scheme to move the velocity at each point. At each point in the velocity field it calculates back in time to figure out what value would be at this point. This means that each value in the new velocity field is written to once. This means that there is more dissipation in the velocity field which is fixed by the projection step.

To create a more interesting simulation and test the different visualization techniques on more turbulent flow an optional fourth step called vorticity confinement is used to intensify all the vortices by adding additional force in the direction of the spin of each vortex.

The first visualization technique I used was texture advection to allow the simulation of density or dyes in the fluid. This essentially is a rendering of a texture that gets advected by the velocity field at each step. This uses the same function that the simulation uses to advect the velocity field by itself. Instead of advecting the velocity by itself the density is now advected by the velocity.

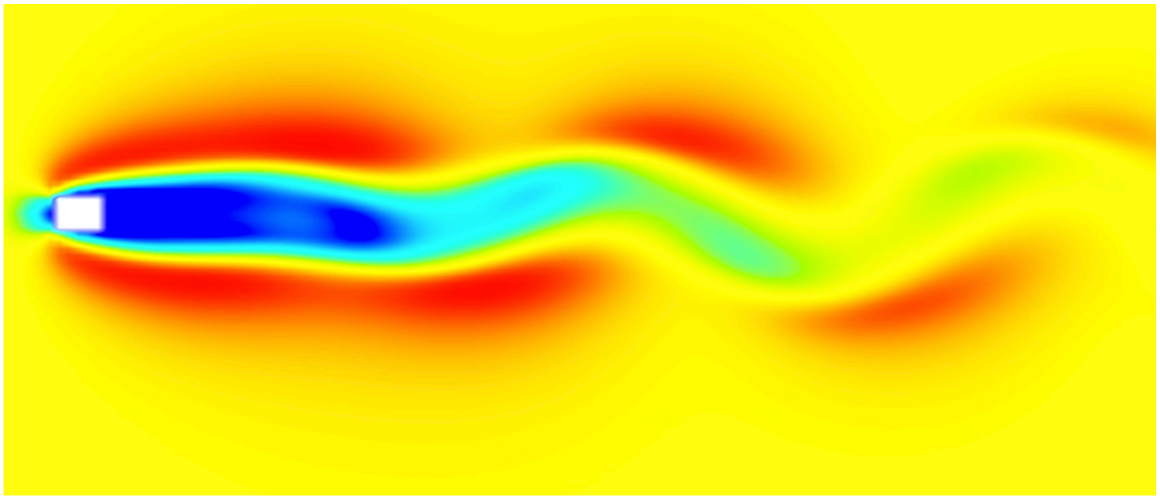


Figure 1 Heat Map

Another visualization I did was directly on the velocity of the simulation. Using a function that took a value from 0 to 1 and output it as a heat map I applied it to the velocity texture to produce the above image. Later on this same function was applied to streaklines and streamlines allowing the user to see the velocity magnitude on the lines themselves as shown below.

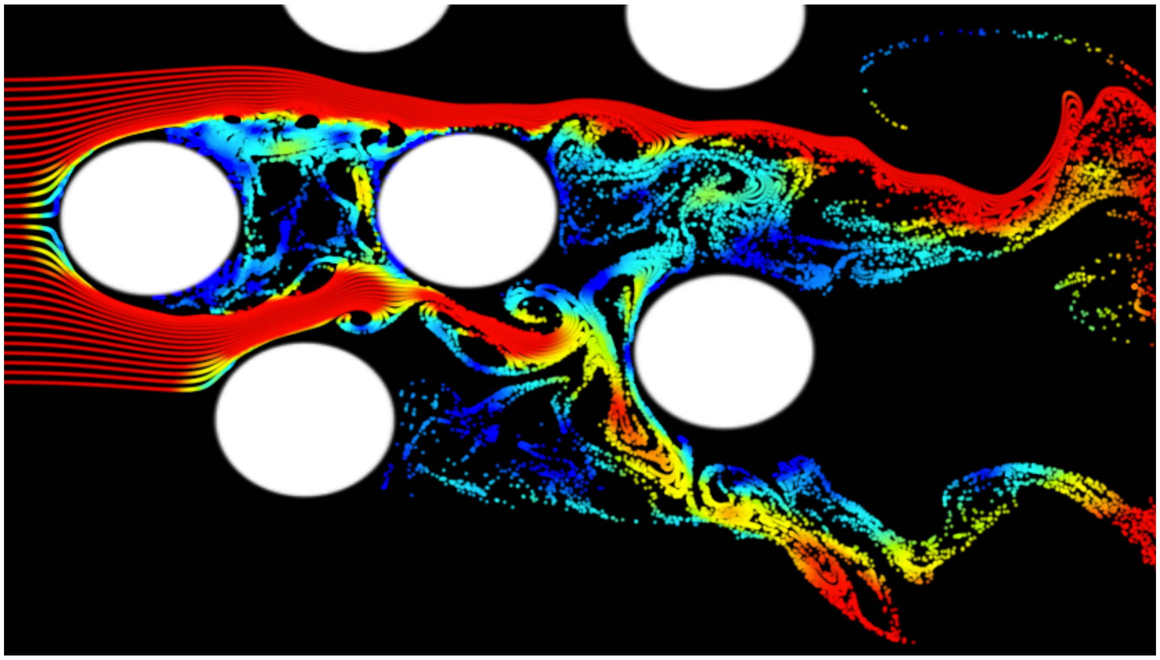
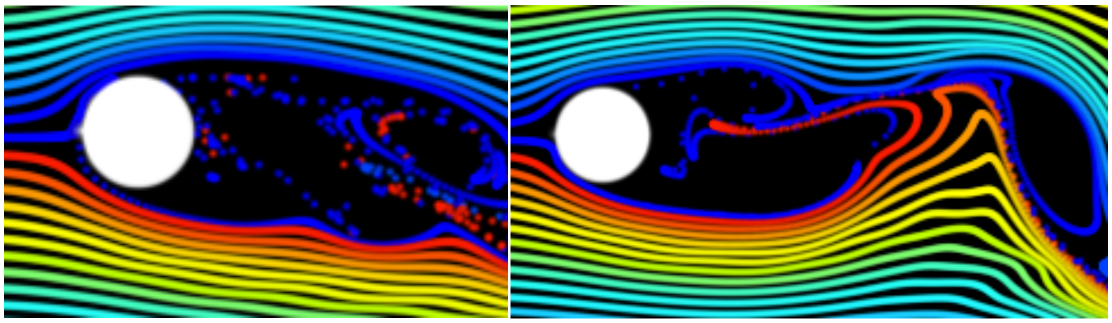


Figure 2 Heat Mapped Streaklines

Particles can also be simulated on the GPU using the same technique of doing the calculations on floating point textures and rendering them to another texture. Essentially the positions of each particle are stored on a texture. The texture is then read in the vertex shader and used to position a set of points. Currently the simulation is advancing and rendering 65 thousand or so anti aliased particles with little to no slowdown. These particles can be used to do a variety of different things. One such use is to render streaklines, since these are very chaotic and particles don’t always form lines to connect its rather tricky to get a good looking rendering that doesn’t have rendering artifacts. I tried some different strategies to break the lines when they started to fall apart however this still left nasty artifacts and there is no right answer when a line is broken or not. This led me to the conclusion that I would need to try another method. Since advancing a lot of particles is very cheap you can output enough that they create the lines themselves. When a particle escapes instead of creating a rather nasty rendering artifact and a line cutting across the screen the particles do their own thing when they get separated until they expire.



The above pictures shows a streakline that would be impossible to render using lines since it is shifting which way it goes around an object. Depending on the turbulence of the flow this becomes more and more of a problem.

Unlike streakliines, streamlines do not have these same problems. Since you are only concerned about the flow at one point in time it is impossible for the line to have particles escape since they are all based on where the previous particle is. This is both good and bad, it means that rendering these with lines can be done without artifacts, however it also means that it is much slower since it cannot be done in complete parallel. Since each particle is based on the position of the last particle you must compute the last particle before advancing to the next, unlike streaklines where every single particle is advanced independently and only based on the previous time step. This uses a similar technique to advance particles to create the lines however instead of advancing them once each step the whole line of particles is reset and advanced. Then lines are drawn between the points

Results

At the end of the day the project now allows a user to input objects into a texture and have get ruff estimate of what the flow might like around the objects with a variety of techniques all in real time. It allows the user to seed streaklines and streamlines as well as see the velocity heatmap as well as input dyes into the fluid.

Allowing the user to input objects with a texture was also something that improved the project and means one can draw something they would like to see the fluid flow past and run the simulation. Although not truly accurate it gives an idea of what the flow would look like and the visualizations only need a velocity texture to run which makes it relatively simple to change the simulation to something new.

Conclusion

Overall the project had some shortcomings. I worked on getting the 2D version to work correctly since it was a lot faster and a lot easier to fix than the 3D version. Unfortunately this means that the 3D version got pushed to the back. This left the 3D version somewhat unusable and I had to abandon it due to a bunch of problems with boundaries, run speed and time constraints. I was caught off guard that most of my problems came with the simulation side, which I was expecting would be rather easy to implement. This left me with a lot less time than I would have liked to develop the visualization part of the project which was more interesting to me.

Other than that I think the project came together quite nicely and it looks quite good running in real time. Allowing someone to input objects with an image and use different visualizations on the fluid is really cool. Exploring the capabilities of the GPU were also quite interesting, we didn’t get to do anything with modern OpenGL in CMPS 160 except a tiny exercise with GLSL in a closed environment so it was a rather new experience for me working with shaders and being able to do pretty much what I wanted without being constrained to the small subset of options fixed function OpenGL provides.

Looking back it probably would have been wise for me to not try and tackle so much at once. Learning modern OpenGL at the same as trying to understand the math behind the simulation and develop multiple different visualizations was just too much. If I had to redo it I would probably use one of the fluid dynamics packages available online, unfortunately if I went this route it wouldn't be running on the GPU and probably wouldn't be running in real time, which makes things a lot less interesting. Either way it wouldn't be ideal.

In the end of the day the project was really fun and I learned a lot about doing computations in parallel both with OpenGL and OpenCL (although this wasn't used in the final project) as well as learning modern OpenGL and Shaders.

References

[1]http://www.intpowertechcorp.com/GDC03.pdf

[2]http://www.dgp.toronto.edu/people/stam/reality/Talks/FluidsTalk/FluidsTalk_files/v3_document.htm

[3]http://www.stanford.edu/class/cs237d/smoke.pdf

[4]https://developer.nvidia.com/sites/default/files/akamai/gamedev/docs/opengl_rendertexture.pdf