"ME 599 – Texel Modeling": Final Report

Danny Sale
University of Washington
Mechanical Engineering

Winter 2013

1) Introduction

My main motivation for taking this class was to get an introduction to general-purpose computing on graphics processing units (GPGPU computing), and to start thinking about how to utilize GPGPU computing in conjunction with vortex particle methods applied to simulation of incompressible flows. The following sections describe what I have learned during this class related to topics in GPGPU computing, Linux operating systems, and vortex methods. This report covers the following topics:

- Prior Experience in GPGPU
- Setting up the PC Hardware and Operating Systems for GPGPU
- GPU Acceleration and Benchmarking of the Biot-Savart Law
- Simulation of Vortex Rings Using a GPU Accelerated Vortex Particle Method
- Conclusions and Future Work

2) Prior Experience in GPGPU

Prior to enrolling in this class, my only programming experience was with Matlab (my most proficient language) and a little bit of Fortran 90/95 (which is also very similar to Matlab). I had never written any C/C++ or CUDA code, but I have begun learning these language and working through a couple highly recommended instructional books (c.f. [1, 2]). I quickly realized that writing my own GPU accelerated computational fluid dynamics (CFD) code in CUDA C/C++ was overly ambitious, so I decided to work with Matlab and the Parallel Computing Toolbox which was a more familiar language for me.

3) Setting up the PC Hardware and Operating Systems for GPGPU

This section discusses the hardware and software used to produce the results presented in this report. Two different hardware configurations were tested in this study, and each was configured to multi-boot Windows and Linux operating systems. Table 1 lists the hardware and software configurations of each machine.

3.1) Installing CUDA on Unsupported Linux Operating Systems

It would be beneficial to make use of already developed open source software as much as possible for writing my own computational fluid dynamics (CFD) code. For this reason, I also thought it would be best to develop a proficiency with Linux operating systems, since most of the useful open-source libraries are supported best on Linux.

As shown in Figure X, Nvidia CUDA 5 is officially supported on only a limited number of operating systems. I discovered that the reason for this limitation is related to the gcc compiler, because CUDA 5

relies on the now obsolete versions of gcc (I think it relies on gcc v4.3 and/or v4.4 compilers). This limitation becomes problematic when one strives to be on the bleeding edge of computational hardware and software. I thought it might be best to figure out how to compile CUDA code on the latest and greatest versions of the gcc compiler because many "modern" operating systems ship with newer versions of gcc (now typically gcc 4.7 and greater). Fortunately, I discovered a method to compile CUDA code on operating systems which are not officially supported by Nvidia, after I spent several weeks doing nothing but reading Linux forums and pounding my head against walls after suffering numerous computer crashes.

Here is one possible solution, and the one I think is the best. If your operating system ships with a newer version of gcc, install an older and CUDA compatible version of gcc to a new directory. Do NOT touch the gcc you already have, it can wreak havoc on the OS. The reason for installing multiple versions of gcc is because you don't want to compile forever with the old gcc. When you want to compile CUDA code, use the nvcc compiler flag "export CC=gcc-4.47". More information is found in the nvcc documentation, which only amounts to an extra compiler flag or environmental variable. After researching and experimenting with most of the OSes listed in Figure 1 and using the "trick" to install CUDA with newer versions of gcc, I found that openSUSE v12.1 (and greater) provided the drivers with best compatibility between my specific CPU/GPU hardware and the Nvidia CUDA 5 libraries.

Table 1. Hardware and Software Configurations. "Host" refers to specification of the CPU and its available memory, and "Device" refers to specifications of the GPU.

Configuration:	Desktop	Laptop
Make/Model:	"home brew" built by me	ASUS Zenbook UX51Vz-XH71
Approx Cost:	~$2400 USD	~$2300 USD
OS:	multi-boot w/ GRUB: openSUSE 12.1 + Windows 7	multi-boot w/ UEFI: openSUSE 12.2 + Windows 8
Host:	Intel Core i5-3570K 4 cores @ 3.4GHz 16 GiB DDR3	Intel Core i7-3612QM 4 cores @ 2.1GHz 8 GiB DDR3
Device:	GeForce GTX 680 1536 CUDA cores @ 1006 MHz 4 GiB GDDR5 CUDA Compute 3.0	GeForce GTX 650M 384 CUDA cores @ 900 MHz 2 GiB GDDR5 CUDA Compute 3.0

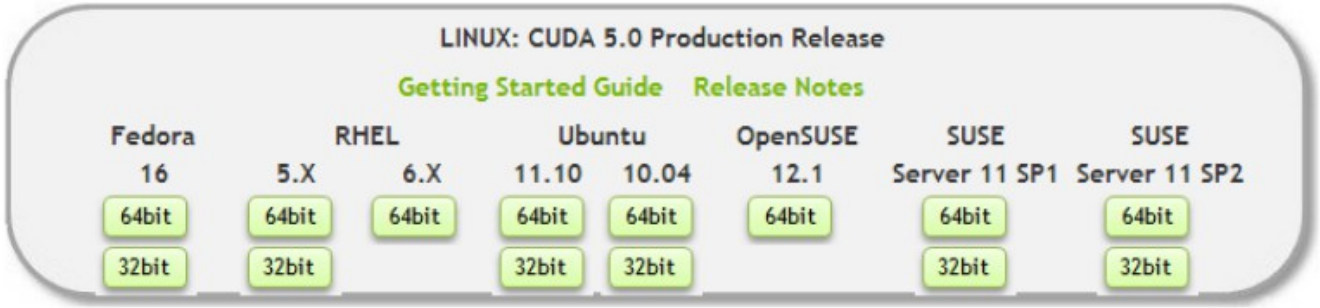


Figure 1: Operating systems compatible with CUDA 5, according to NVIDIA official release notes. (https://developer.nvidia.com/cuda-downloads)

4) GPU Acceleration and Benchmarking of the Biot-Savart Law

The Matlab Parallel Computing Toolbox was used in this study in order to take advantage of the multi-core CPU and GPU hardware. Matlab cannot achieve the same speed as CUDA C/C++ or Fortran; however, it is often faster to develop code in Matlab due to the great number of built-in functions and toolboxes available for Matlab. The Matlab Parallel Computing Toolbox was used in this study in order to take advantage of the multi-core CPU and GPU hardware, and in order to investigate the potential of utilizing GPUs within vortex methods.

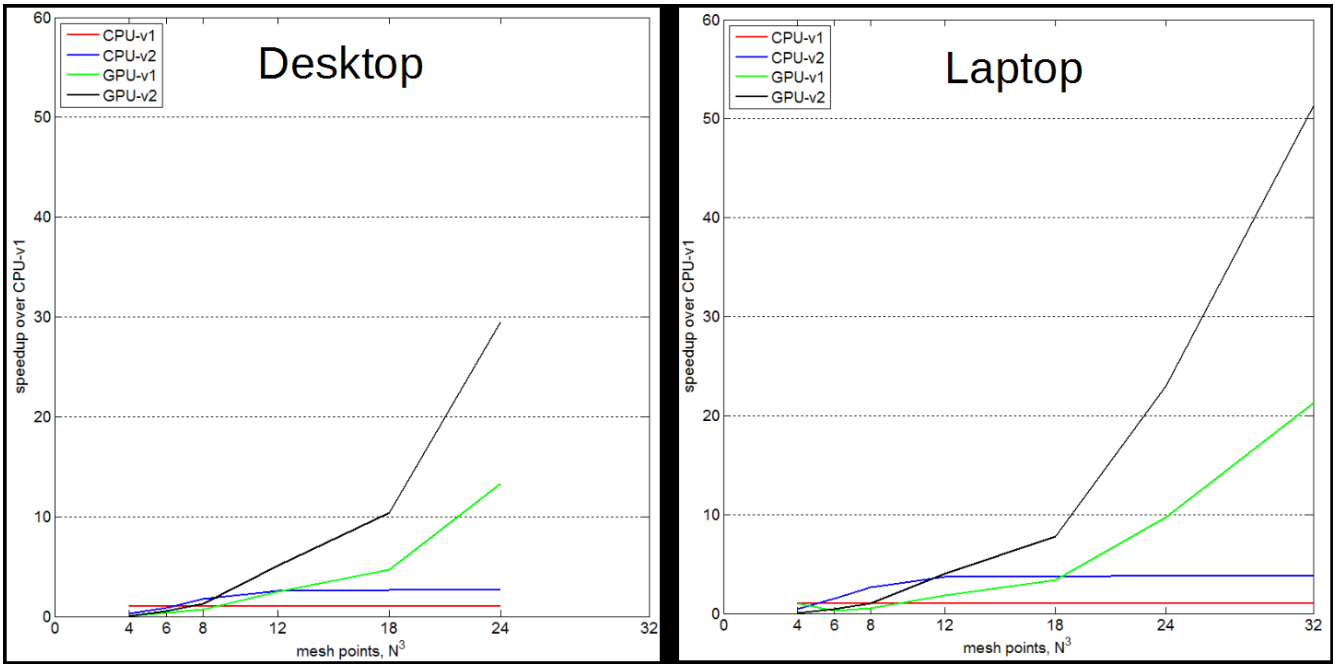
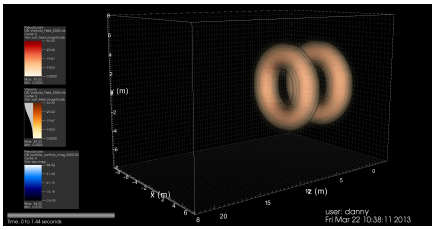


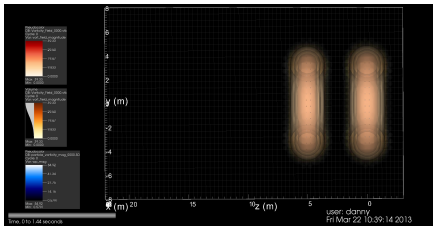
Figure 2: Speedup over a single threaded CPU code when calculating the velocity on an N^3 grid using the Biot-Savart law. The different "versions" of the code represent successive improvements made by me to parallelize the code, and both multi-core CPU and GPU accelerations were made.

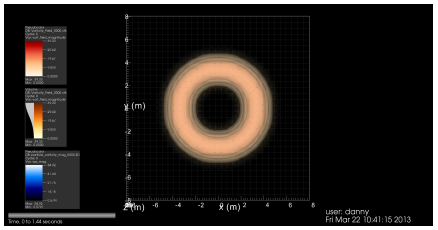
5) Simulation of Vortex Rings Using a GPU Accelerated Vortex Particle Method

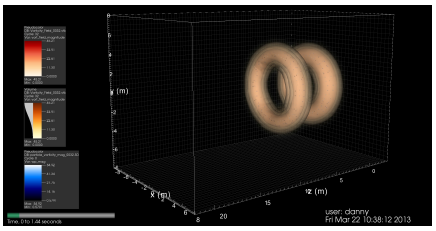
A vortex particle method, using the method described by [3] was implemented using the Matlab Parallel Toolbox to simulate the interaction of leapfrogging vortex rings. Two examples of the vortex ring simulations are shown in Figures 3 and 4, and the source code to produce these results are included with a copy of this report. The visualizations were created using the VisIt software, an open-source code used for visualizing partial differential equations.

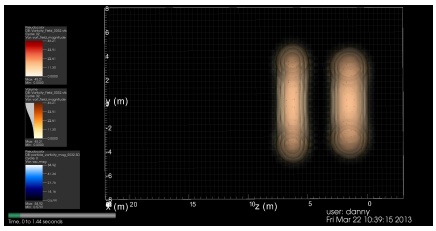
The simulation begins with two distinct vortex rings (see first row) and ends with turbulent breakdown of the rings (see last row). As the simulation evolves, the above visualization reveals fusion of the rings as the trailing vortex ring is pulled through the leading ring, followed by the development of azimuthal instabilities in the rings which eventually leads to viscous dissipation of the rings.

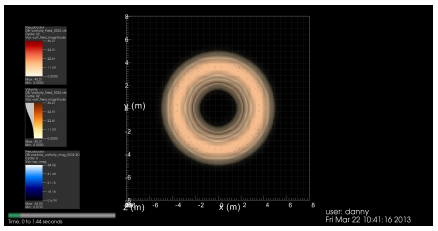

(a-1)

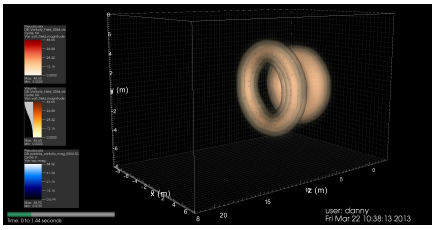

(b-1)

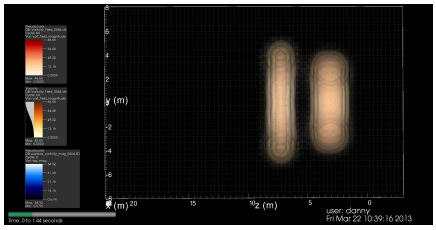

(c-1)

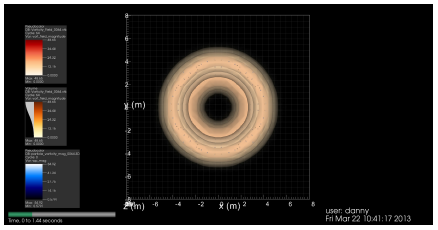

(a-2)

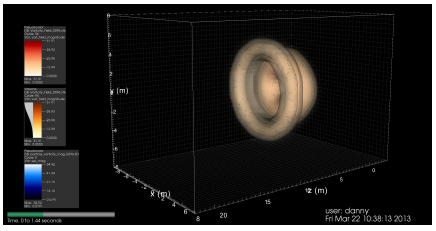

(b-2)

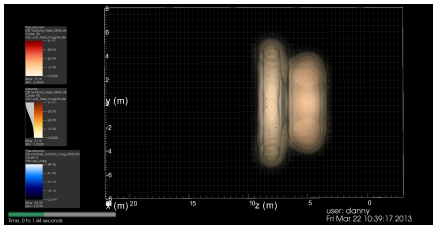

(c-2)

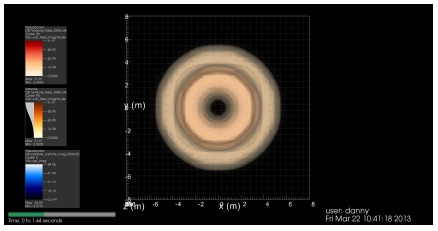

(a-3)

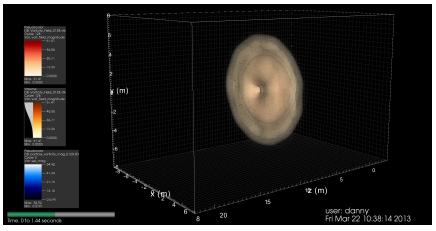

(b-3)

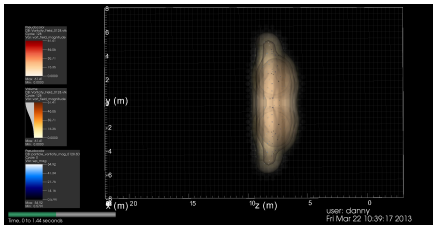

(c-3)

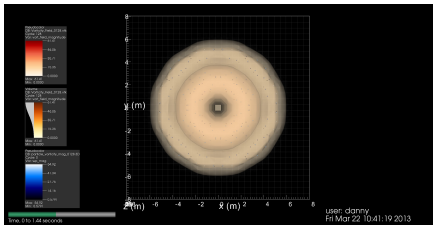

(a-4)

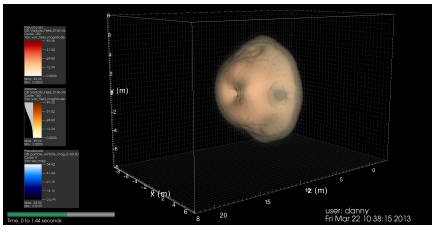

(b-4)

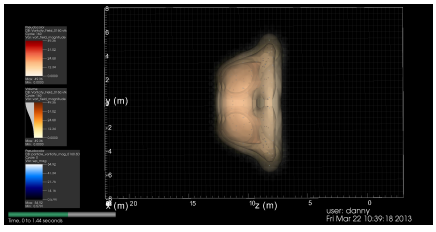

(c-4)

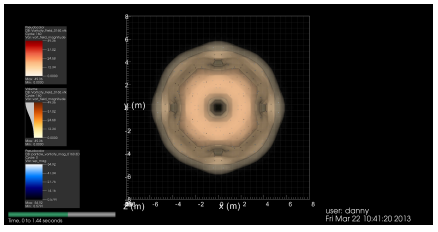

(a-5)


(b-5)


(c-5)


(a-6)


(b-6)


(c-6)

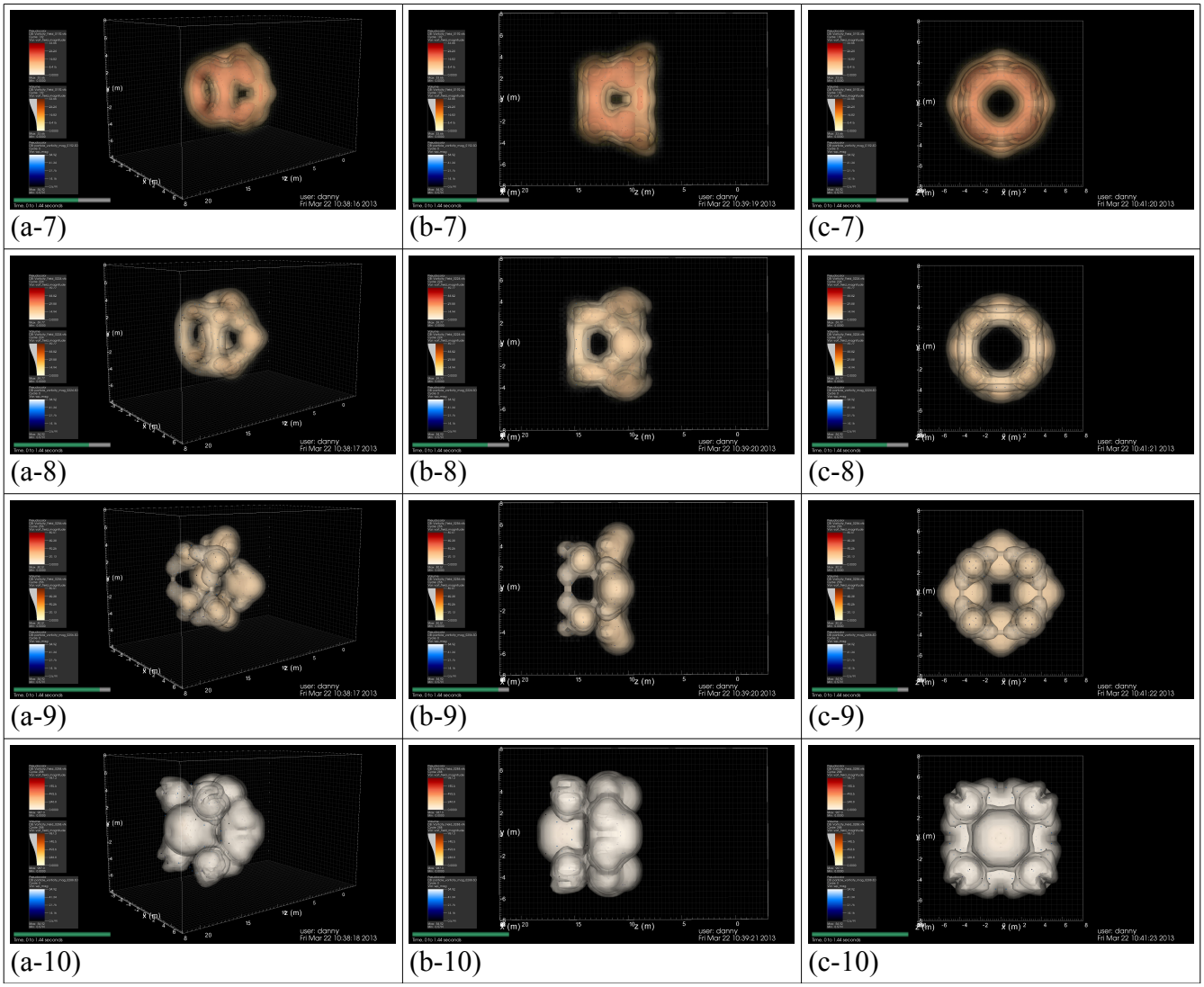
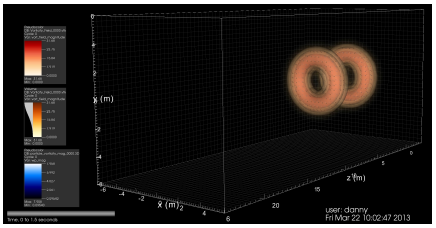
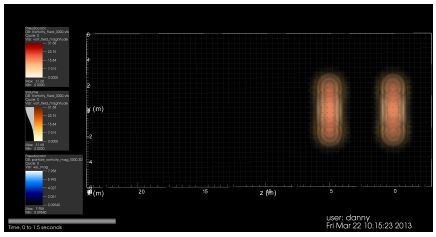
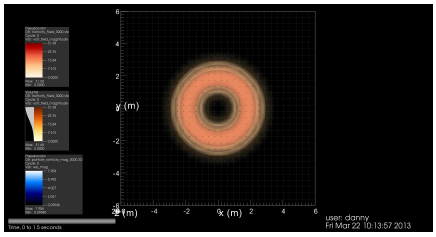
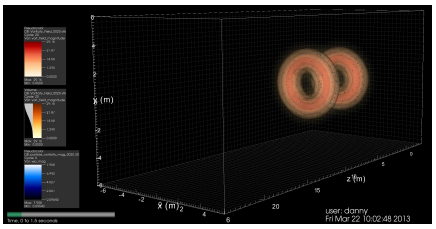


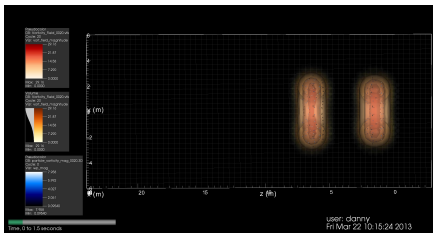
Figure 3: Simulation of leap-frogging vortex rings "Case A: Skinny Rings, Re=400". The three columns show a different view of the simulation at corresponding time steps. THIS IS A HIGH RESOLUTION IMAGE, ZOOM IN PDF TO MAKE ANNOTATIONS LEGIBLE.

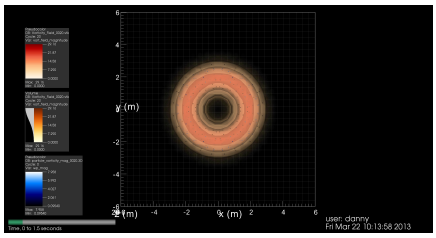

(a-1)

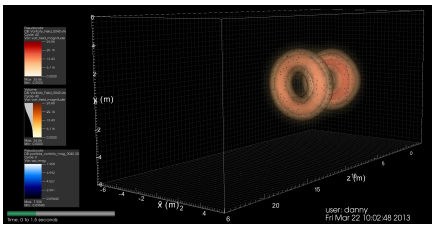

(b-1)

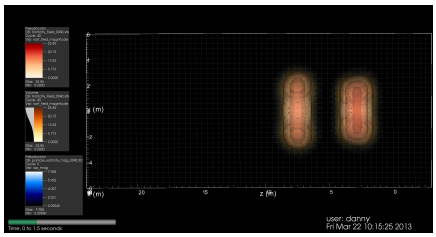

(c-1)

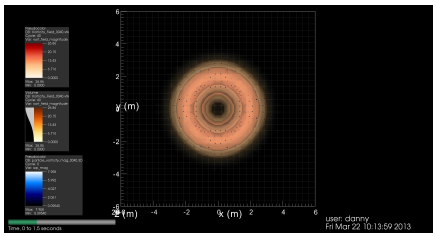

(a-2)

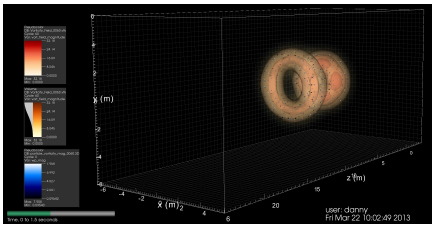

(b-2)

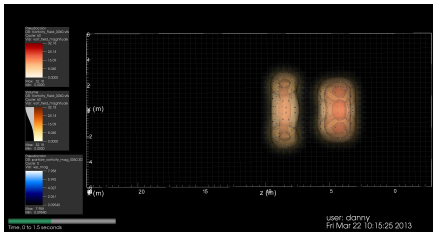

(c-2)

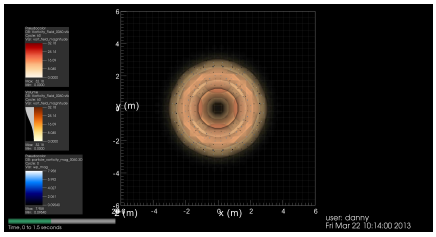

(a-3)

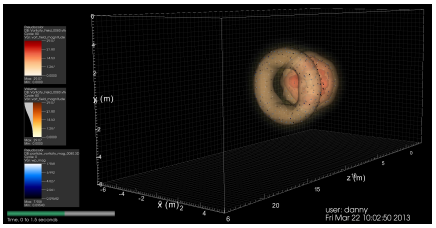

(b-3)

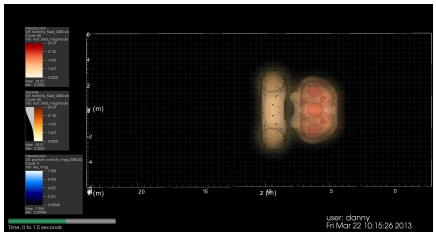

(c-3)

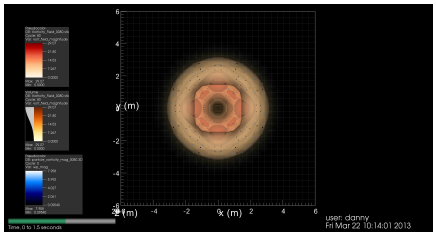

(a-4)

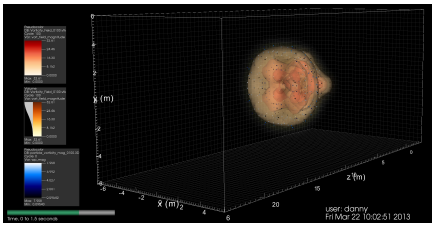

(b-4)

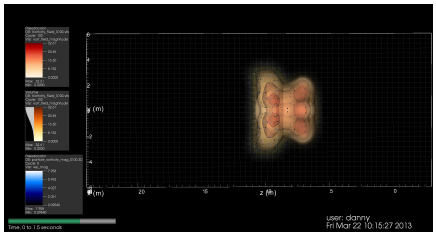

(c-4)

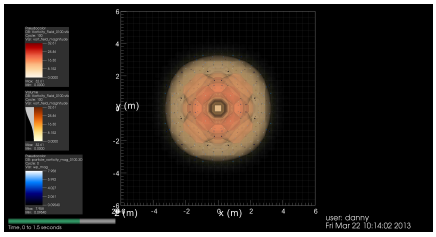

(a-5)


(b-5)


(c-5)


(a-6)


(b-6)


(c-6)

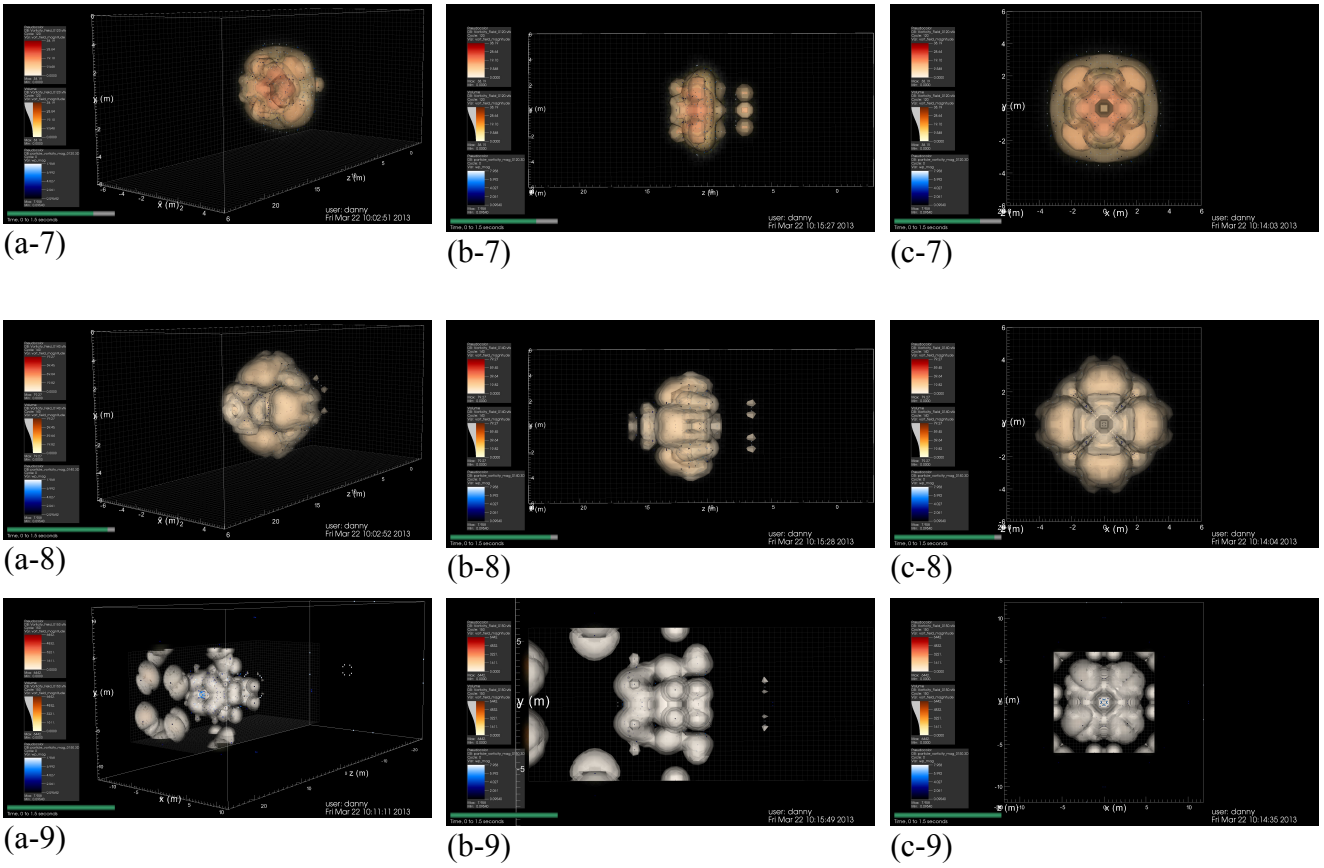


Figure 4: Simulation of leap-frogging vortex rings "Case B: Thick Rings, Re=400". The three columns show a different view of the simulation at corresponding time steps. THIS IS A HIGH RESOLUTION IMAGE, ZOOM IN PDF TO MAKE ANNOTATIONS LEGIBLE.

6) Conclusions and Future Work

The use of GPUs was shown to significantly speed up my code, success! Once I become more proficient with CUDA and its hooks into the C/C++ and Fortran languages, I will begin porting my Matlab code into C/C++ or Fortran in order to significantly speedup my simulations.

7) References

[1] A. Koenig and B.E. Moo (2000). Accelerated C++: Practical Programming by Example.

[2] J. Sanders and E. Kandrot (2010). CUDA by Example: An Introduction to General-Purpose GPU Programming.

[3] G.S. Winckelmans and A. Leonard (1993). "Contributions to Vortex Particle Methods for the Computation of Three-Dimensional Incompressible Unsteady Flows."