Class Project: Gravitational Field Visualization

Matthew Hogan and Scott Merkley

April/15/2023

Department of Computer Science, University of Utah, Salt Lake City, 84112, UT, USA

Introduction

This report covers the results and figures for the Class Project for CS 5635 at the University of Utah. This project overviews simulating the gravitational field between the Earth-Moon system and a binary black hole merger in Paraview using different filters to gain an intuition for how the gravity propagates through space. The visualization methods used include glyphs, contours, volumes, and streamlines to help display the information in a thought provoking manner.

Newtonian gravitation is a first-order approximation of Einstein's theory of General Relativity, and is a common study in introductory physics courses. The theory was first presented in Newton's *Principia Mathematica* and states that the force $\vec{F}$ felt by an object with mass m_1 is mathematically defined to be:

$$\vec{F} = -G\frac{m_1 m_2}{r^2}\hat{r} \tag{1}$$

where r is the object's distance to the body producing the gravitational field, with mass m_2, and G is the universal gravitational constant. Since this force is a vector quantity, when multiple objects with gravitational fields are present, to total force on a test mass in the field is the superposition of each of these.

Earth-Moon System:

First, we set out to simulate the gravitational field between the Earth and Moon in Python using Newtonian physics. A space of points was created using three Numpy arrays, then the Earth's and Moon's masses, radii, and positions were set. In order to better visualize the gravitational field, the unit length of our space is in terms of 100,000 km, which is reflected in the gravitational coefficient G. Two functions were created, one to check if the current vector was inside the Earth's radius, the second to calculate the gravitational acceleration at an individual location in space. Every coordinate in space was then looped over, calculating the total gravitational acceleration at that point. Then the coordinates and acceleration were appended to the end of a list. Once every point in space had been looped over, the list was converted to a Panda's data frame, given column names, and exported to a CSV file.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd


numberOfPoints = 101
maxSpace = 800

# Make a space for everything to live
x = np.linspace(0, maxSpace, numberOfPoints)
y = np.linspace(0, maxSpace, numberOfPoints)
```

```
z = np.linspace(0, maxSpace, numberOfPoints)

XX, YY, ZZ = np.meshgrid(x, y, z)

# Gravitational Constant
G = 6.67e-21

# Generate a Earth
massEarth = 5.97e24
earthRadius = 6.37e-2
earthPosition = np.array([400, 400, 400])

# Generate a Moon
massMoon = 7.34e22
moonRadius = 1.7e-2
moonPosition = np.array([17.5, 400, 400])

# Function to check the distance of the given vector from the center of the earth
def insideEarth(vec):
    return np.sqrt((vec[0] - earthPosition[0])**2 + (vec[1] - earthPosition[1])**2 + (
                                                    vec[2] - earthPosition[2])**2) <
                                                    earthRadius + 100


# Function to generate gravitaional field force data
def gravity(vec):

    if(insideEarth(vec)):
        return [0, 0, 0]

    rM = vec - moonPosition
    rE = vec - earthPosition

    rm_mag = np.linalg.norm(rM)
    re_mag = np.linalg.norm(rE)

    return -G * (massMoon * rM / (rm_mag) ** 3 + massEarth * rE / (re_mag)**3)


# CHOOSE WHICH FILE NAME AND FORMAT TO EXPORT AS
fileName = 'test100PointsRadiusExtension'
# fileFormat = 'RAW'
fileFormat = 'CSV'

data = []
normData = []
# data.append(["x", "y", "z", "fx", "fy", "fz"]) # Naming the columns for numpy array
for i in x:
    for j in y:
        for k in z:
            r = [i, j, k]
            f = gravity(r)

            if fileFormat == 'RAW':
                data.append([np.linalg.norm(f)])

            if fileFormat == 'CSV':
                data.append([f[0], f[1], f[2], i, j, k, np.linalg.norm(f)])
                # normData.append([np.linalg.norm(f)])



# MAKING A NUMPY ARRAY AND SAVING TO .RAW
if(fileFormat == 'RAW'):
    data = np.array(data).astype("float32")
    print(data)
    data.tofile(fileName + ".raw")


# MAKING A PANDAS DATAFRAME AND SAVING TO .CSV
if fileFormat == 'CSV':
    data = pd.DataFrame(data)
```

```
        data.columns = ["fx", "fy", "fz", "x", "y", "z", "norm"]
        print(data)
        data.to_csv(fileName + ".csv")

        # Only Norm Data
        # normData = pd.DataFrame(normData)
        # normData.columns = ["norm"]
        # normData.to_csv(fileName + "Norm.csv")



    print("\nDONE")



    generatePlots = False

    if generatePlots:
        # Plotting in 2D
        plt.plot(moonPosition[0], moonPosition[1], 'r.', ms = moonRadius, zorder = 10, label
                                                    = "Moon")
        plt.plot(earthPosition[0], earthPosition[1], 'b.', ms = earthRadius, zorder = 10,
                                                   label = "Earth")
        plt.plot(XX, YY, 'k.', ms = 0.5)
        plt.grid()
        # plt.legend()
        plt.show()

        # Plotting in 3D
        plt.figure(figsize = (15, 15))
        ax = plt.axes(projection='3d')
        ax.plot3D(moonPosition[0], moonPosition[1], moonPosition[2], c = 'r', marker = 'd')
        ax.plot3D(earthPosition[0], earthPosition[1], earthPosition[2], c = 'b', marker = 'd
                                                   ')
```

Once the CSV file was created, it could not be directly opened using Paraview due to a difference in the formation of the file. Instead, Excel was used to delete the row ID of the original CSV and then saved again. The file was then able to be opened and manipulated using Paraview. The file had to be converted from table to points using the TableToPoints filter, and then a calculator was used to allow the force data to be read by the glyph filter. The calculation used was $fx*iHat+fy*jHat+fz*kHat$ giving us a result array with the magnitude of the gravitational force at every point. A glyph filter was then applied giving the figure below.

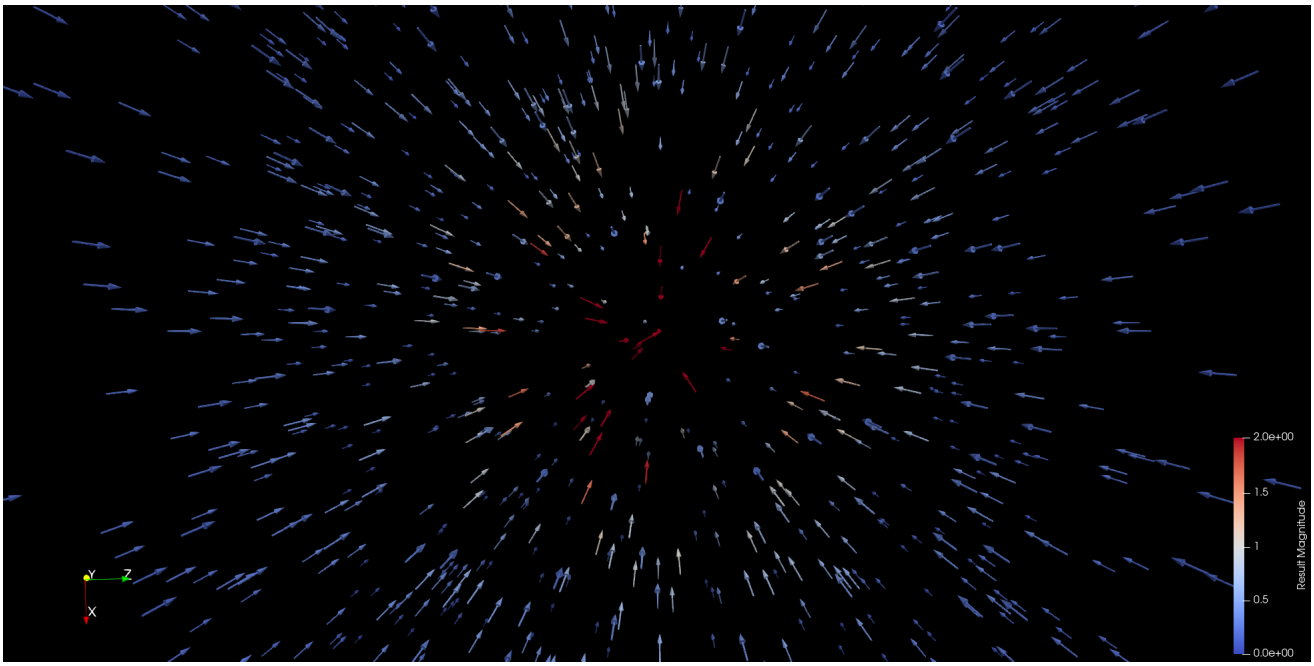

Figure 1: Glyph Visualization of Gravitational Field Between Earth and Moon

The glyphs were then scaled to the result array, giving a better representation of the force of gravity and how quickly it fades as the distance from the radius increases.

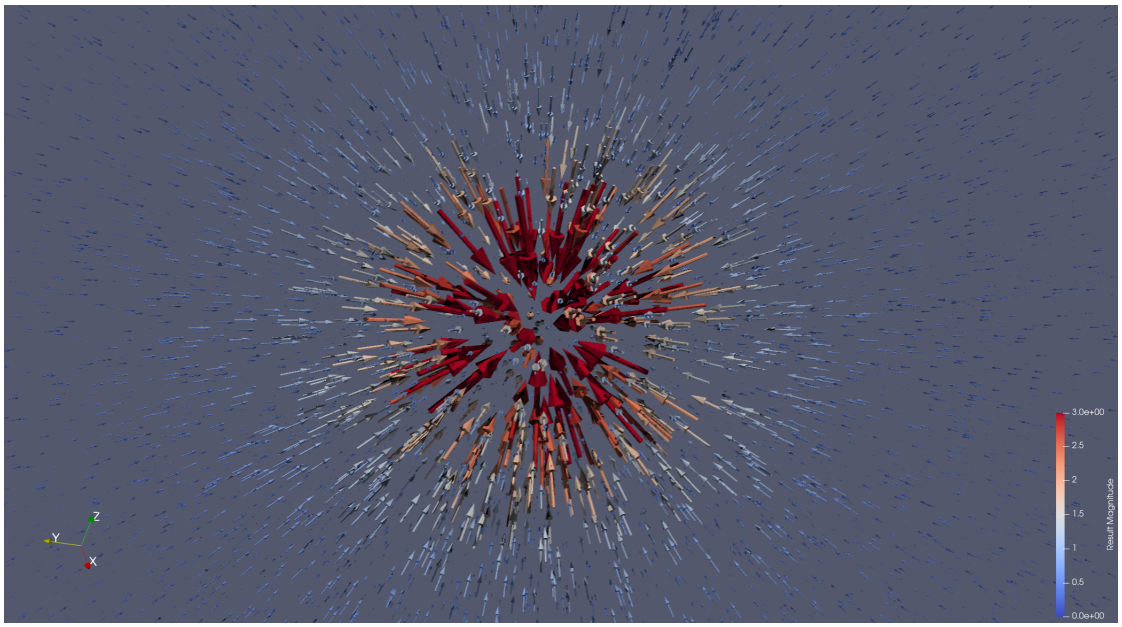

Figure 2: Scaled Glyph Visualization of Gravitational Field Between Earth and Moon

It was then desired to show the Earth and Moon in the system with the glyph. This was done using the TableToStructuredGrid filter, and defining the extent of the X, Y, and Z columns. Once this was done, a Contour filter was used with an isosurface value of 3 to give the following visualizations. Note that the isovalue chosen was arbitrary, and does not show the actual relative size of the Earth and Moon; in reality, the Earth and Moon are much smaller than what is shown in the following figures.

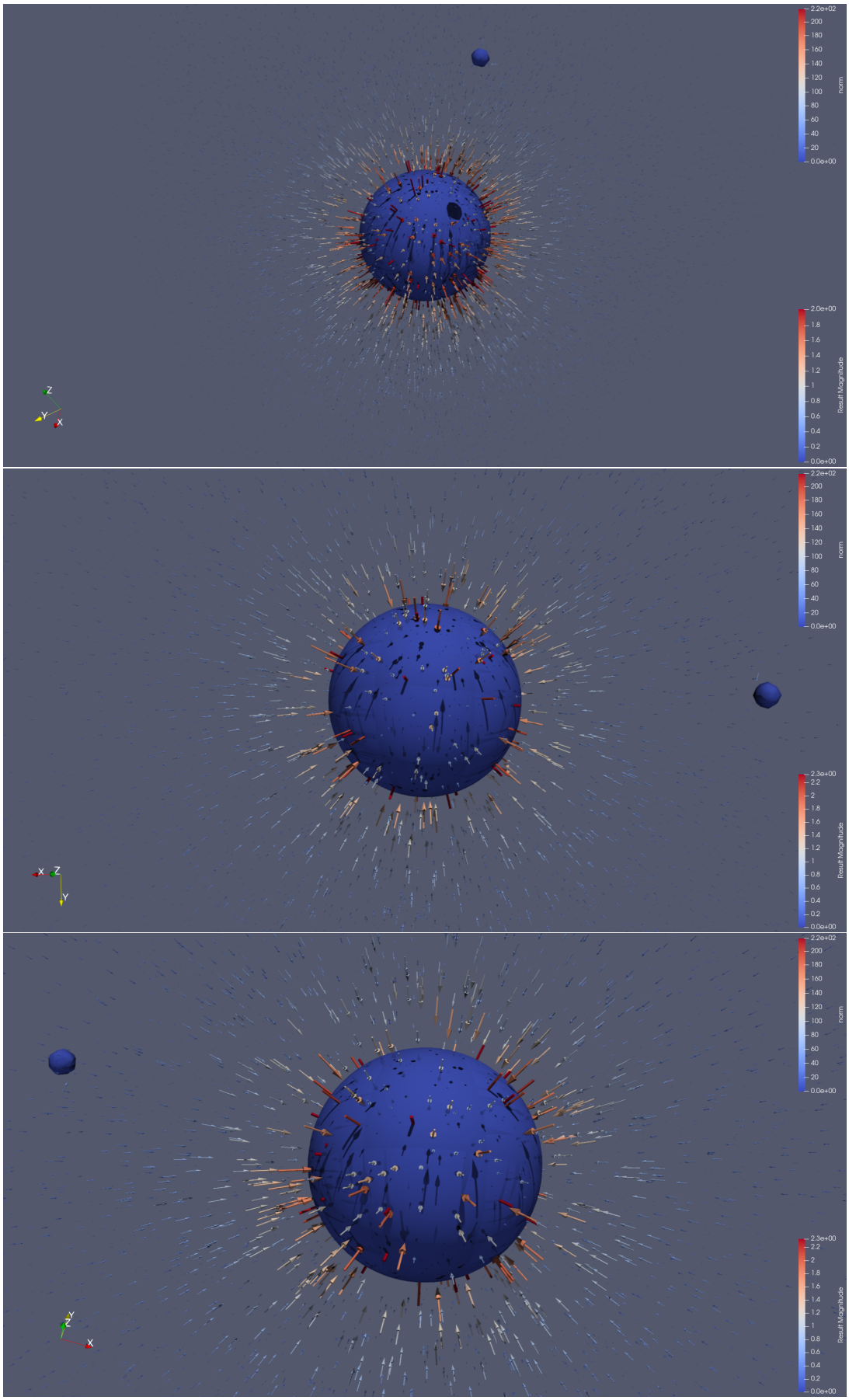

Figure 3: Visualizations of Gravitational Field Between Earth and Moon with Isocontours

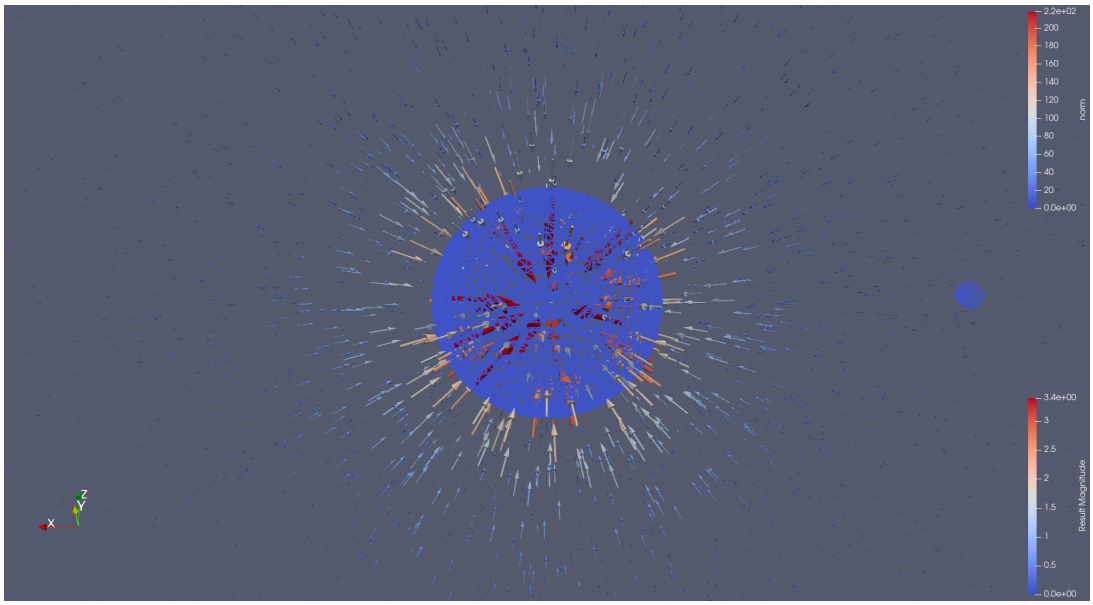

Figure 4: Visualization of Gravitational Field Inside Earth and Moon

These visualizations give us a good understanding of the gravitational field around and inside the Earth-Moon system. However, these visualizations are limited in showing the details of interesting features we originally planned on viewing; namely the effect of tidal forces.

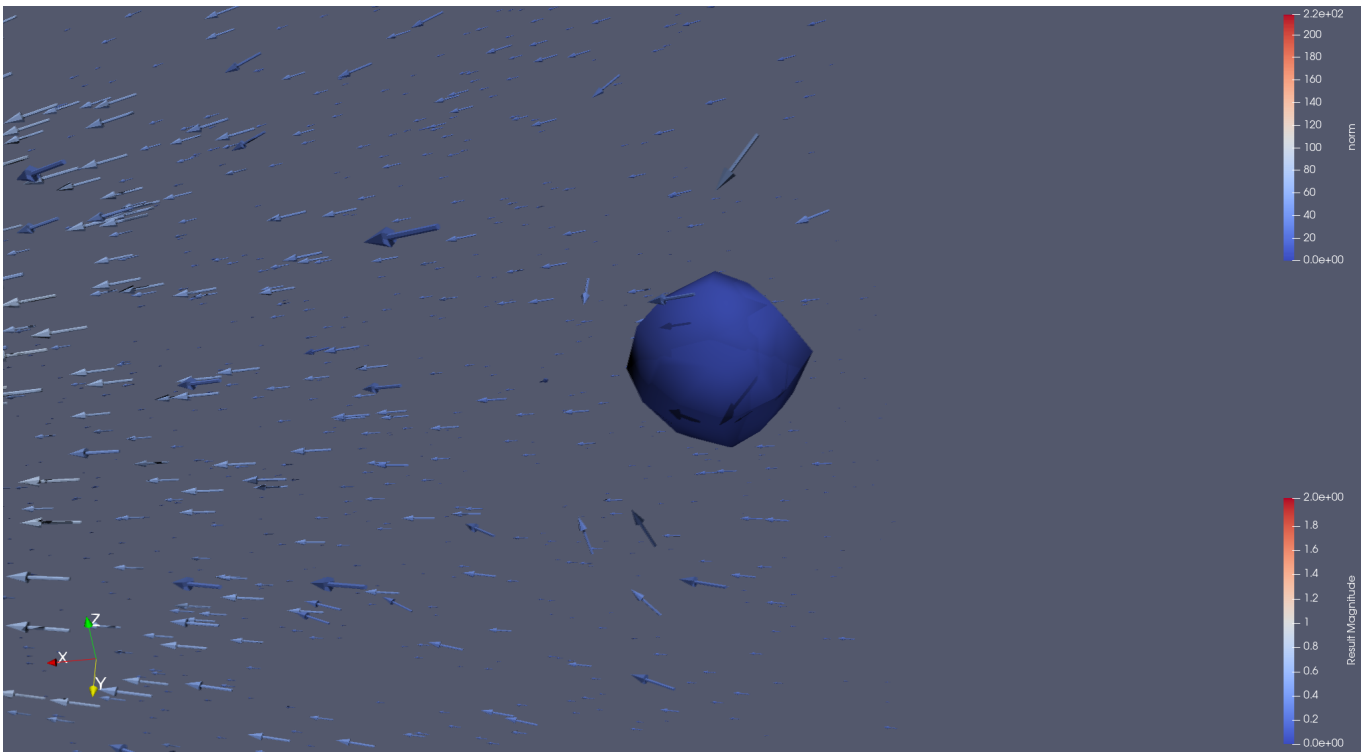

Figure 5: Visualization of Gravitational Field Around the Moon

Lagrange Points:

Next, we set out to view the Lagrangian point between the Earth and Moon: the point where the force of gravity is cancelled out. The figures above show the point to be very close to the Moon, far closer than was originally anticipated; it was thought that it would be somewhere nearer to the center of the two bodies. However, the glyph representation of our data and the relatively small size of the moon made this visualization challenging. In order to illustrate this further, we amended our code by increasing the mass of the two bodies, made them roughly the same size, and produced volume renderings of the magnitude of the field, shown below. These figures clearly show the Lagrange point of the two bodies between them, and the spherical field around them becomes disfigured near this point.

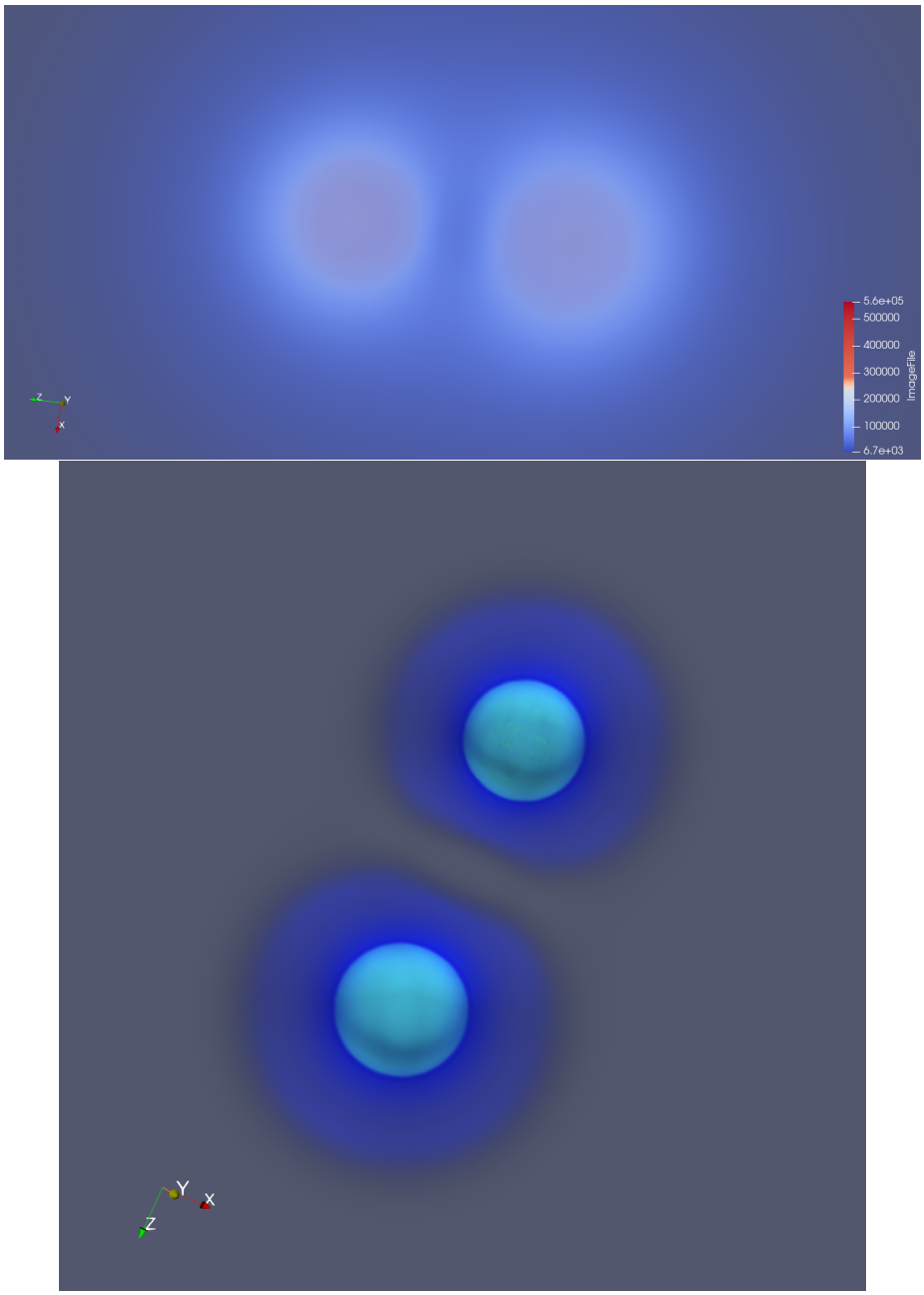

Figure 6: Volume renderings of a simulated binary neutron star system colored with 1D (top) and 2D (transfer functions)

Binary Black Hole Merger:

For the last part of our project, we received binary black hole merger simulation data from Milinda Fernando at the University of Texas at Austin. The data simulates two supermassive black holes rotating around one another, showing the gravitational waves emitted through space and the immense explosion when the two collide. The main reason we chose to visualize this data was to illustrate the differences between classical Newtonian gravitation and the full theory of General Relativity. A large file was sent and we were told to look at one of the folders containing PVTU files. These files were able to be directly loaded into Paraview, and using the points representation a view of the black holes was able to be seen.

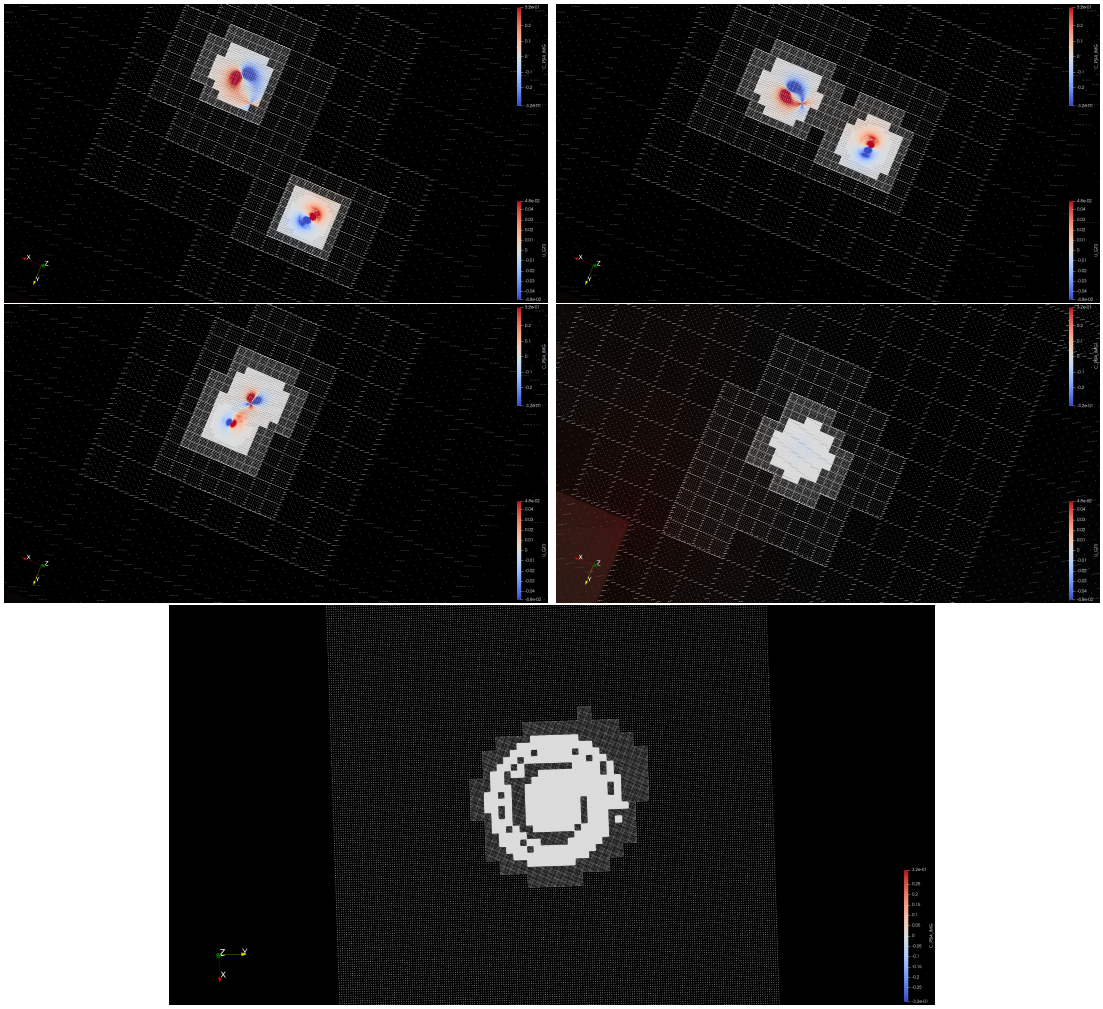

Figure 7: Visualization of Gravitational Field of Two Merging Black Holes

After rotating around each other, the black holes collide and form a new, larger black hole. This collision causes a massive disruption in space-time and can be seen above from the white spiral coming out of the middle of the screen. These gravitational waves are quickly emitted to many times the size of the system in a single time step.

It is unknown to us what the red and blue colors in the beginning figures mean, that was not explained to us when we received the data. We were only able to visualize the data, there are many different variables to view in the file but we are unsure of what their relation is to the real world.

Results:

In this project, we have been able to accomplish many of the goals we set out to achieve. We also ran into many more road blocks and difficulties than anticipated but ultimately chose a project well suited for the time span and our abilities. In the very beginning we were able to get the Python script up and running quickly but ran into more issues with reading the files in Paraview.

At first, we tried using raw files exported into vtk files, since we had used them before for other assignments in the class. They worked, but didnt give all the flexibility in manipulation we were looking for. We then tried CSV files, finding them more intuitive and easy to use. Once we figured out that Excel was needed to get the CSV files into a format that was readable by Paraview we were able to make headway. Glyphs and contours also proved to be a challenge. The glyphs would show on the screen, but wouldnt orient themselves in the direction of the magnitude of the force. We realized that this problem was due to not having a magnitude vector which was then created using a calculator filter, as described above. After this, the glyph data was oriented in the correct direction but was improperly scaled. When scaled, there would be gigantic glyphs coming from the center of the coordinate space. It was realized that this was due to using Newtonian gravity and having points at the center of the Earth. This problem can be seen below.

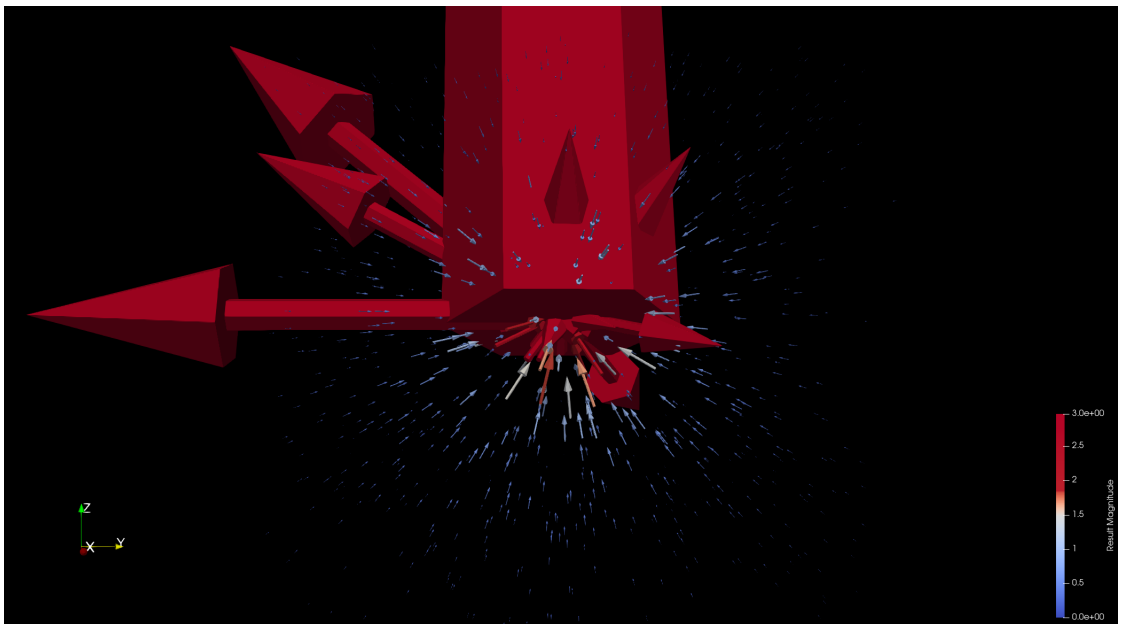

Figure 8: Incorrect Glyph Scaling

This was fixed using the insideEarth method in the Python script to identify if a vector was inside the earth and setting it to 0 if it was.

Trying to visualize contours while visualizing glyphs proved to be the most difficult part of the project. All the data was input into Paraview and magnitude vectors were available for the contour to use. Whenever the contour was applied to the TableToPoints filter, it did not show anything. Later, it was found that this was because a structured grid had to be used in order to get contour plotting to work. After applying the TableToStructuredGrid filter and defining the extent of the data, the contour filter could then be correctly applied. This allowed more real visualizations to be made.

Conclusion:

This project showed how the different tools learned in this class are helpful in visualizing different forms of gravitational field data, allowing users to get an informed intuition as to how gravity behaves in our local system. From glyphs showing the magnitude of gravitational force around the Earth and Moon to Lagrange points in neutron stars and gravitational waves rippling out from binary black hole mergers, gravitation can be seen to emanate through space and draw other bodies nearer.

We would evaluate this project as around 90% successful. We were able to get the desired visualizations, simulate data using Python, and even visualize black hole data. Although there are still some issues that were not able to be resolved, like having the magnitude of our data at 3 for our contour plots instead of the correct 9.8 (Earth surface gravity), and not being able to put textures on the Earth and Moonl however, we were still able to accomplish most of the goals we set out to achieve. We got the glyphs, contour, Python script, and coloring working.

One thing we were not successful at was showing the tidal forces of the Moon as it moves around the Earth. Something that was not anticipated at the beginning of the semester was how small the tidal acceleration caused by the Moon was relative to acceleration produced by the Earth itself. Upon further research, we found that tidal forces are an order of 10^{-7} smaller than the force of gravity generated by the Earth at the surface, which is nearly impossible to visualize given the resolution of our data. We thought the issue was related to the resolution of our space, and so we did try scaling our data from 100 points to 1000. The data generation took almost 10 hours to complete and resulted in a 4GB file which was unexpected. Even with this high-resolution data set, we were still not able to visualize the differences in the field due to the tidal acceleration. However, this exercise did illustrate how quickly data sets can grow at least.

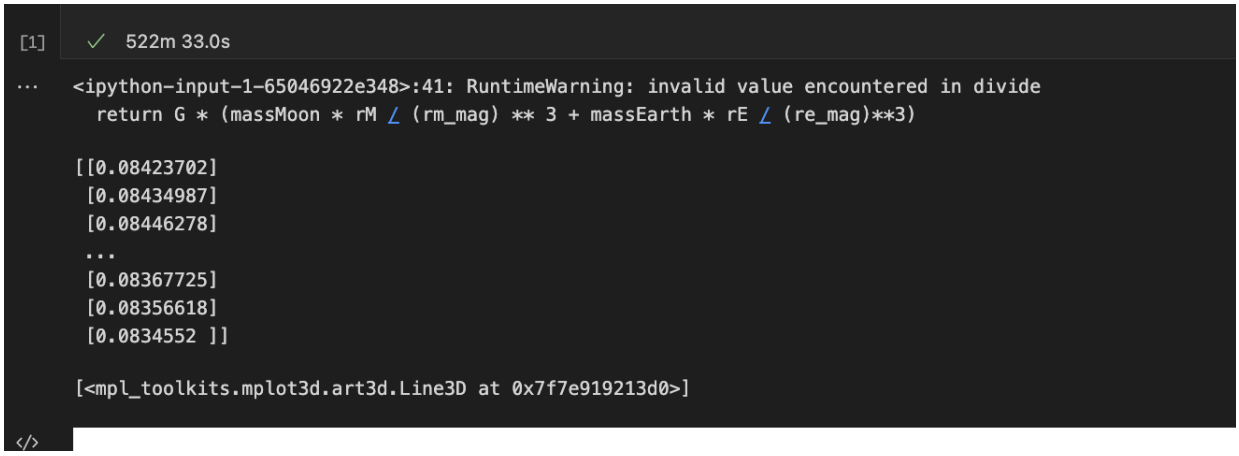

Figure 9: Timer Showing a Nearly 10 Hour Run Time

References

[1] "Newton's Law of Universal Gravity" : https://en.wikipedia.org/wiki/Newton

[2] "Lagrange Point" : https://en.wikipedia.org/wiki/Lagrange_point

[3] "Stack Overflow" : stackoverflow.com

[4] "Paraview Forum" : https://discourse.paraview.org/

[5] "Black Hole Merger Data" : https://www.linkedin.com/in/milinda-fernando-406887201