Virtual Reality: An Experiment in Physics Outreach

M. Thapa 8936289

School of Physics and Astronomy, University of Manchester, UK

ABSTRACT

Virtual Reality is a recent technology with unique properties that create new prospects for phyics communication. It's popularity yet scarcity amongst the public make it attractive, perfect for Physics Outreach. In this 6 week project, the possibilities of VR as an outreach asset is briefly explored. Two "fly through space" type programs are constructed to accept particle position and velocity data from the University of Manchester Physics Department (or elsewhere) in .csv file format. One program is a real-time, N-Body simulation that evolves with newtonian gravity and demonstrates emergent classical mechanics. It can compute up to 50,000 particles. The second program is a visualisation of particle position data in 3D accepting the order of 10^6 particles. The use of these programs is detailed. Other potential VR projects, as well as improvements to the completed programs are discussed.

I. Introduction

Technology is becoming a larger and larger part of education. In just the last 25 years, classrooms have gone from having blackboards, to whiteboards, to projectors, to interactive whiteboards, to linked slideshows on student's own phones and the lecturer's computer, and so on. iPads are commonplace in primary schools as a way of interactive learning. What was once just a static sketch of a concept on set of chalk axes is now a real-time, interactive animated object on a screen. DavidBau's interactive conformal mapper [1] is an excellent example of how thoughtful visualization can present a new perception of an abstract concept. All of these developments and concepts capitalize on one thing: Human's innate ability to rapidly process visual information.

Effective outreach involves imparting the maximum amount of understanding to a prospective student within the attention span they subconsciously provide. If just one 'Eureka' moment can be invoked, then that student has felt the true reward of science – and hopefully fallen in love with it! Humans have short attention spans, so visual stimuli is key to getting concepts across quickly. Technology, on the other hand, extends the attention span in correspondence with Arthur C. Clarke's third law [2][3]. Virtual Reality (VR) is the perfect combination of visual information with technological gimmick.

Purposed for immersion, VR supplies a very effective way to hold people's attention. However, VR also allows for depth perception, a unique feature within virtual media. This can (and should) be used capitalized on for use in education.

In this project, an N-Body simulation was chosen as a physical system that would demonstrate the benefits of visualization with depth perception. The primary goal was to create a program that would allow members of the University of Manchester Physics Department to input data they provide. No prior knowledge of Unity or C# was held when starting. Two programs were compiled: "Oculus - Real Time Gravity Simulaiton.exe" and "Oculus - 3D Plot.exe". Both were created in the game engine "Unity" [4].

This document will begin by explaining how to deploy these programs to get the most out of an Outreach event, then go on to describe their workings. The limitations of the programs and difficulties experienced will be detailed, followed by an outline of some concepts that were beyond the scope of this project.

II. Manual I - How to set up and use the executables

Note: I have done my best to explain using words and screenshots, however this manual is best read in conjunction with some experimentation with the programs.

Oculus - Real Time Gravity Simulation.exe

As the name suggests, Real Time Gravity Simulation is a real-time, N-Body Simulation of Newtonian gravity. Upon launching the program, a cube of particles (represented by 2D sprites) will be generated and immediately collapse in on itself due to gravity. The colour of each particle is based on the absolute magnitude of their velocity, highlighting some physical properties of the system.

Launch:

- Ensure that all cables for the Oculus Rift and the Xbox controller are plugged in to the PC.
- Find the .exe shortcut shown in **fig.1** on the desktop and open it.

Figure 1: The "Oculus – Real Time Gravity Simulation.exe" shortcut. Highlighting of the shortcut will display the full name.

- A window will appear. Ensure that the settings match those shown in **fig.2**.

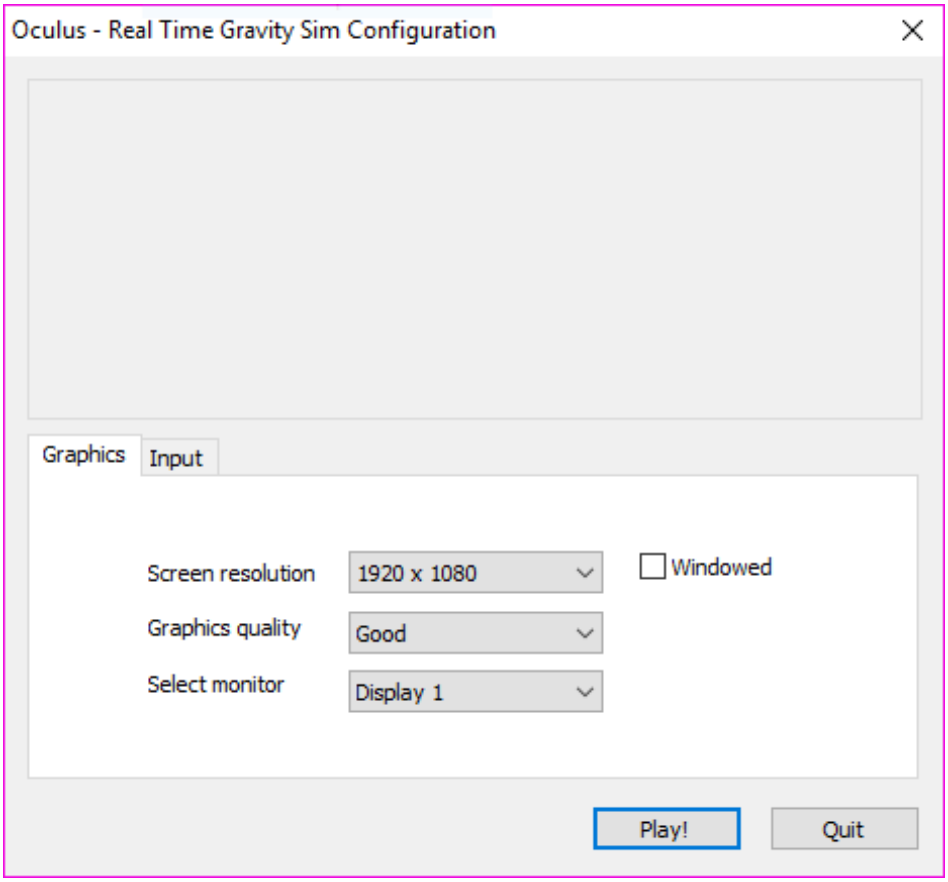

Figure 2: The launch window for "Oculus – Real Time Gravity Simulation.exe". The 'Input' tab is not relevant.

- Click "Play!"
- The program will now launch, and the screen will be mostly or all black. Either placing the headset on, or by blocking the light sensor to allow it to activate, rotate the headset until something is visible.

Controls:

Keyboard	**Controller**	**Function**
WASD	Left Stick	Move player relative to view direction
Space	Right Trigger	Ascend
Z	Left Trigger	Descend
L Shift	A	Hold to increase speed
P	Start	*Pause Particles*
Right Arrow	X	*Next Initial Condition*
Left Arrow	B	*Previous Initial Condition*
Down Arrow	Left Bumper	*Decrease Colour Scale*
Up Arrow	Right Bumper	*Increase Colour Scale*
Enter	-N/A-	Enable Mouse/Controller looking (not recommended)
Backspace	-N/A-	Disable Mouse/Controller looking
G	-N/A-	Toggle Controller Permissions (see below)
Mouse	Right Stick	Rotate camera (Mouse/Controller looking)
Esc	-N/A-	Quit Application

Table 1: The control schemes for "Oculus – Real Time Gravity Simulation.exe". -N/A- are unmapped functions – these actions cannot be performed on the controller.

The force on each particle is accurate to a small softening term which prevents division by zero. There are no collisions in this simulation, so close approaches lead to orbital or slingshot effects. It has four hard coded initial conditions that are discussed in Section V. Other than these, there are three initial conditions which are read from file. Files are optional for Real Time Gravity Simulation, though recommended. Each one can be loaded during runtime, and the resulting motion observed. Details on how to supply files are provided in Section IV.

Inputs from an Xbox controller and a keyboard are accepted at all times. The control scheme for the inputs is shown in **Table 1**. The Xbox controller is intended for the user wearing the Oculus Rift, while the keyboard allows for more remote control of the simulation, for example by a demonstrator.

It is possible to allow for the controller or mouse to change the orientation of the camera, although it does not work very nicely in combination with the Oculus' own orientation mechanics. For someone wearing the Oculus Rift, the effect can be very nauseating and it may also misalign the movement controls making the simulation difficult to navigate. For this reason it is NOT RECOMMENDED that this is enabled. It is disabled by default. It is useful for debugging or demonstrating the sim on a screen if the Oculus Rift is not hooked up to the PC.

The keyboard button "G" toggles whether the controller can input the controls shown in italics. This allows a scenario in which a demonstrator (with a keyboard) can control all aspects of the simulation, while a user with the controller is only able to fly around. This may be preferable if there are additional people watching a TV screen showing the Oculus' view.

Oculus – 3D Plot.exe

Unlike Real Time Gravity Simulation, 3D Plot has no physics interactions. It plots a particle distribution from file into 3D space and allows one to fly through and explore the distribution. The colour of each particle is based on how many particles are in close proximity to it. The range within which neighbouring particles are counted can be changed. This allows different aspects of a distribution to be highlighted.

Files can be loaded up successively allowing for a "slideshow" of distributions. There are 30 snapshots of a dark matter distribution evolving over time to demonstrate this slideshow ability.

Launch:

- Ensure that all cables for the Oculus Rift and the Xbox controller are plugged in to the PC.
- Find the .exe shortcut shown in **fig.3** on the desktop and open it.

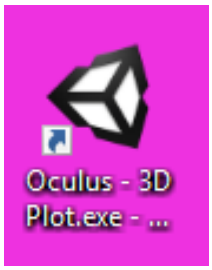

Figure 3: The "Oculus – 3D Plot.exe" shortcut. Highlighting of the shortcut will display the full name.

- A window will appear.

- Ensure that the settings match those shown in **fig.4**.

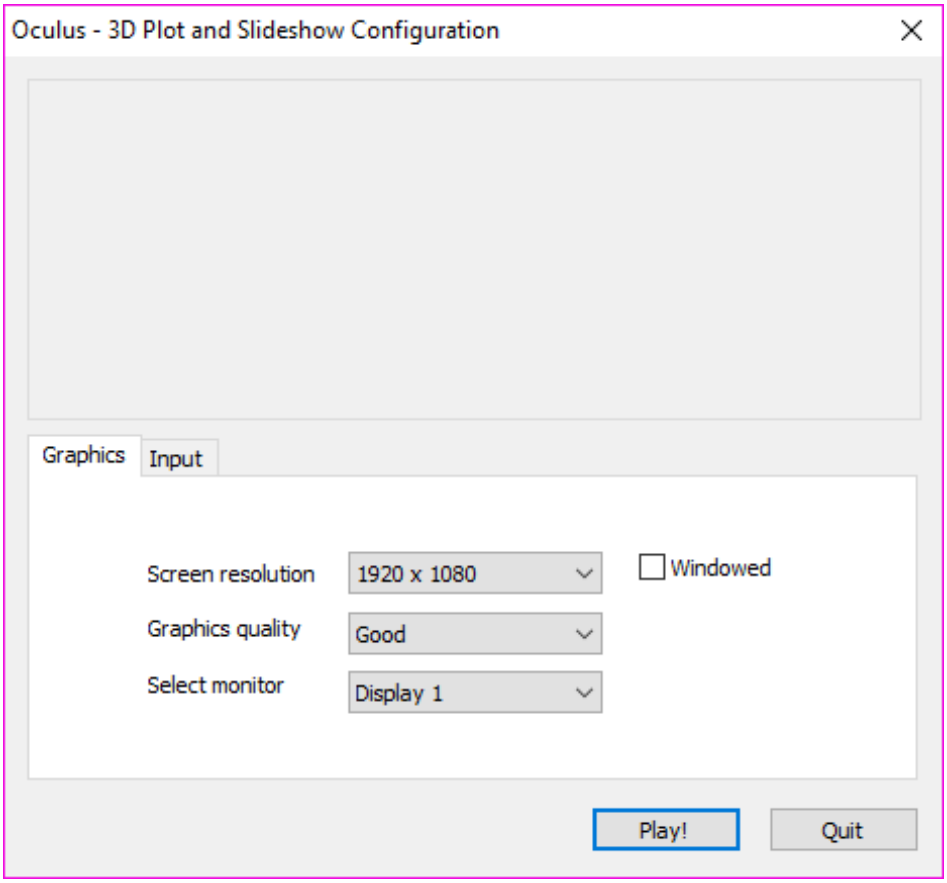

Figure 4: The launch window for "Oculus – 3D Plot.exe". The 'Input' tab is not relevant.

- Click "Play!"
- The program will now launch, and the screen will be mostly or all black. Either placing the headset on, or blocking the light sensor to allow it to activate, rotate the headset until something is visible.

Controls:

Keyboard	**Controller**	**Function**
WASD	Left Stick	Move player
Space	Right Trigger	Ascend
Z	Left Trigger	Descend
L Shift	A	Hold to increase speed
P	Start	*Pause Particles*
Right Arrow	X	*Next File*
Left Arrow	B	*Previous File*
Down Arrow	Left Bumper	*Increase Colour Scale Precision*
Up Arrow	Right Bumper	*Decrease Colour Scale Precision*
Enter	-N/A-	Enable Mouse/Controller looking (not recommended)
Backspace	-N/A-	Disable Mouse/Controller looking
G	-N/A-	Toggle Controller Permissions (see below)
Mouse	Right Stick	Rotate camera (Mouse/Controller looking)
Esc	-N/A-	Quit Application

Table 1: The control schemes for "Oculus – 3D Plot.exe". -N/A- are unmapped functions – these actions cannot be performed on the controller.

For simplicity, the launch method and the controls shown in **Table 1** are almost exactly the same as for Real Time Gravity Simulation.

Again, it is possible to allow for the controller or mouse to change the orientation of the camera by pressing "G". To avoid rapid nausea, it is NOT RECOMMENDED that this feature is activated, for the same reasons as described in Real Time Gravity Simulation's instructions.

An important distinction from Real Time Gravity Simulations is the *requirement* of files to read particle data from. 3D plot exclusively plots files and will break if there are no files to read.

III. MANUAL II – CUSTOM PARTICLE DATA

Both programs support reading custom particle data from .csv files. These files can be added or removed from /Oculus – Real Time Gravity Sim_Data/StreamingAssets/ or /Oculus – 3D Plot _Data/StreamingAssets/ for each respective program. Files must be perfectly formatted .csv files with each line in the form:

$$x,y,z,v_x,v_y,v_z,m$$

Due to the nature of the programs, the number of particles must be multiple of 256. If there are a non-divisible number of particles, extra particles will not be plotted. For Real Time Gravity Simulation, no more than 50,000 particles are to be supplied. Exceeding this number will cause framerate drops in the program which is highly nauseating for users. For 3D Plot, an order of magnitude of 10^6 particles is the suggested limit.

The scaling of the units must also be taken into account when providing particle data. Unity uses its own arbitrary units which are presumably chosen to be relative to the default camera. Given the velocity of the player and the level of depth perception, it is recommended that a particle distributions is no more than 200 units in diameter.

To help with the conversion of numbers to a reasonable scale, 3 programs are provided that will parse and edit .csv files of the format described above. Each program is a very simple C++ build that will bring up a command prompt with instructions. Place the desired program in the same directory as the file(s) you wish to edit, launch it and follow the instructions.

Oculus_CsvOriginSetter_and_UnitScaler.exe

This program will automatically find the centre of the distribution and place it at the origin. It will scale the distribution by a desired power of 10, and write a new file with the data. The scaling is engineered to preserve the relative trajectories of particles in Newtonian gravity. Therefore, particle masses are also scaled (by double the power). The program will also write a secondary file called ShiftAndScale_Vector.csv which contains the vector to the original centre of the distribution as well as the scaling parameter that was input.

This secondary file is for use in Oculus_CsvShiftAndScaleFromFile.exe.

Oculus_CsvShiftAndScaleFromFile.exe

This program will parse the particle data file as well as a file called ShiftAndScale_Vector.csv which needs to be located in the same directory as this program. It will shift the particle distribution backwards along the vector found in ShiftAndScale_Vector.csv as well as scale it by the scale parameter found inside. It is possible to make the ShiftAndScale_Vector.csv from scratch, but it must be of the format:

$$x,y,z,s$$

on a single line, and with no other lines. The first three terms form the translation vector and '*s*' is the exponent of 10 that the distribution is to be scaled by.

Oculus CsvDataCuller.exe

This program simply takes an input .csv file and extracts only every *n*th particle to a new file. The input file needs to be in no particular order if the physics of the resulting particle distribution is to be approximately the same as the original.

IV. MANUAL III – HOW TO MAKE THE MOST OF THE FEATURES

In this section some of the features and visible physics of the programs is described. It may be interpreted as a loose suggestion of what to demonstrate at an Outreach event.

Oculus - Real Time Gravity Simulation.exe

The four hard coded initial conditions were designed to highlight some simple physical effects for demonstration to prospective physics students.

The Cube distribution loads a stationary set of particles in a cube formation. This quickly collapses in on itself due to gravity, then explodes. A core of quickly orbiting particles is formed and one can observe individual particles being thrown out, slowing down due to gravity and falling back into the core. The outer parts of the cube experience the most acceleration and surpass the escape velocity of the system. These particles are launched to infinity. The symmetry of the system is preserved; immediately after the cube collapses it is possible to observe bands of particles emerging from where the corners were, each of which contains the same pattern of particles and co-moving subsystems.

The 2 Cube distribution is given a spin around an axis at the centre of the two cubes. This sets the two bodies into a binary system which gradually decays over the course of about 2 minutes. While they decay, each body slowly gets more energetic and so gradually change in colour. This is a demonstration of the conservation of angular momentum. While looking at the system side-on, one can talk about binary systems experiencing a temporary dip in luminosity as the stars pass over each other. It is most interesting to observe the moment of coalescence from a position perpendicular to the plane of motion. From here, several defined waves of particles are seen propagating from the system as it merges, a demonstration of the violence of this type of event. The waves can also be used as an analogy for gravitational waves, as they spawn from a similar process.

The Vertical Square initial condition begins with no initial velocity. The immediate collapse shows some interesting perturbations in the distribution as the edges come in. Furthermore, the middle of the distribution barely moves until the edges collapse inward – visualising the 'cancelling' of gravitational forces from either side.

The Horizontal Square Distribution has some spin given to it. This leads to the edges of the square gathering up particles and forming four arms that form into four bodies. As it is, the number of particles is not a square number so the arms aren't exactly symmetrical and quickly decay into three bodies. Eventually these bodies merge.

Both Square distributions demonstrate an important aspect of symmetry – the conservation of linear momentum: While the simulation is entirely 3D, both distributions are initially in a plane and so the motion stays entirely in the plane. This is best visualised by flying directly up to, or through the plane of motion.

The three dark matter particle data files provided are also interesting to observe. Most notably, the second file evolves many self-orbiting clusters of particles quickly. These then gravitate towards the middle of the distribution. After about 2-3 minutes a 10-20 body system evolves, which is very beautiful to watch. One might describe the motion of these bodies much like the stars observed close to the centre of the galaxy and lead on to how we detect black holes in the Milky Way.

Oculus – 3D Plot.exe

After launch, when a file is first loaded into the program, it is parsed by a simple and rather slow C# script into the program. It can take several seconds to load a file for the first time, depending on the number of particles. To mitigate this happening on every call of the file, it is stored in a more accessible format within the program after being plotted. Future calls of that file are then loaded from the program instead of the file.

For this reason, it is recommended that the demonstrator cycles through all of the files in /StreamingAssets/ at least once, immediately after starting the program.

The observable physics of the 3D Plot is mainly dictated by the distribution itself, so to get the most out of this program one should carefully choose what files to include. However, note that files are cycled in alphabetical order, as they appear in the directory. Therefore, naming the files appropriately is paramount to producing a desired slideshow effect.

V. CODE DESIGN I – NEED TO KNOW ABOUT UNITY

Unity was chosen for this project because it is designed to make cross platform for games easy. When it comes to VR, programs need only be built for a normal PC screen; the process to port to VR simply involves checking a "VR supported" Box in the Unity editor.

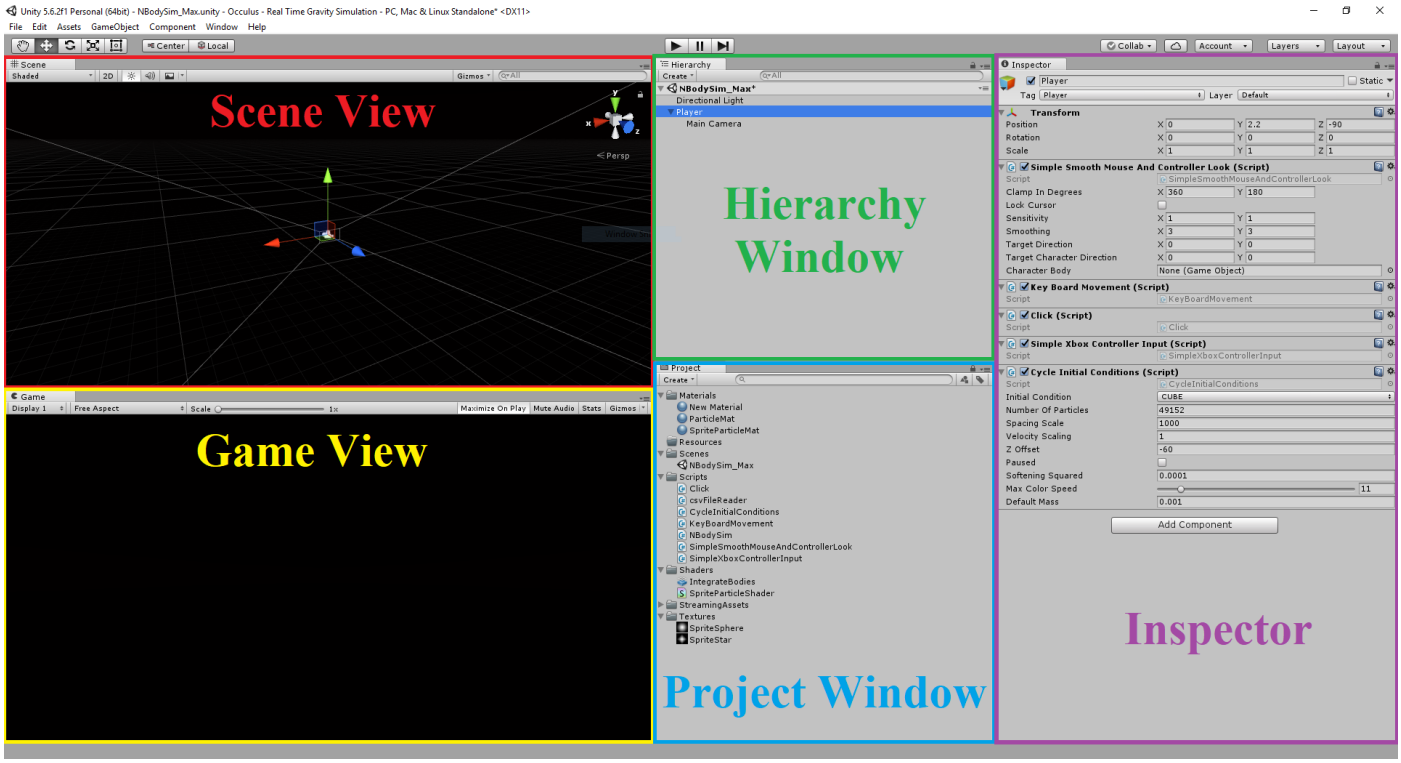

Figure 5: The Unity editor window layout, with each of the 5 windows highlighted. The inspector window is displaying information about the "Player" object, which is selected in the Hierarchy Window.

Fig. 5 shows the Unity editor and its 5 main windows. The scene view provides a view of the game objects in the scene, and allows direct interaction with game objects using various tools. The game view displays a preview of the game during runtime. When the "play" button is pressed, the rendered view of the camera will be displayed in the game view. The hierarchy window contains the game objects in the scene. Indented game objects are children of other game objects. The project window shows the /Assets/ directory of the project. In this directory are all the relevant files for the project including scripts, shaders, textures, materials and scenes. The inspector is a "context sensitive" window. It shows the properties and information of any game object selected in the editor. For example, in the figure it is showing the properties of the "Player" Game object. The Player game object displays several scripts attached. The "Cycle Initial Conditions" script has various editable properties. When edited in the inspector, these properties will overwrite the corresponding public members declared inside the script. For an animated and more comprehensive and overview, see the Unity tutorial [5].

The "VR" check box is found under Edit>Project Settings>Player which brings up "PlayerSettings" in the inspector. Under "Other Settings", check "Virtual Reality Supported" and add Oculus (if it does not automatically appear).

The Unity website provides many short but in-depth video tutorials on how to use its software, as well as general game design relevant to this project. Rather than attempt to reproduce them in a less efficient format here, it is recommended to complete them yourself. The Roll A Ball set of tutorials [6] offers a great kinaesthetic way to become familiar with Unity's basic features. Unity's scripting tutorials [7] demonstrate some aspects of the MonoBehaviour class. Inherited by all game objects, MonoBehaviour allows use of all Unity's inbuilt functions and classes. Unity's full documentation can be found here:
https://docs.unity3d.com/Manual/index.html

VI. Code Design II – Real Time Gravity Sim and 3D Plot

Unity does much of the background communication between scripts, creates a space within which to render the particles to, and makes each program run nicely with the Oculus Rift. The programs created do not make much use of the Unity editor, except for debugging, testing and file management. For this reason, the focus here will be explaining how the scripts work. The commented code can be found on the respective repositories via GitHub here: https://github.com/MaxThapa8936289/.

There are 9 main scripts in each program, 3D plot containing one extra script. Each program has a script that cycles through distributions, a script that can read .csv files, a script that sets up the coordinates of particles and evolves each distribution, a shader to render the particles to the screen (including their colour), and a compute shader which performs calculations on the GPU. The remaining scripts manage inputs from the keyboard, mouse and controller.

The compute shader of Real Time Gravity Simulation, IntegrateBodies.compute, was taken from GitHub where user 'Scrawk' ported a similar shader designed by NVIDIA from CUDA to Direct Compute [8], so that it is compatible with Unity. The original shader's design is detailed in a GPU Gems publication [9].

NBodySim.cs and SpriteParticleShader.cs were taken from GitHub user 'jakedowns' [10]. Both provided skeleton code for this project but both, especially NBodySim, have been heavily modified.

SimpleSmoothMouseAndControllerLook.cs is almost identical to a script found on the Unity forums called SimpleSmoothMouseLook.cs [11]. The only addition is the acceptance of custom axes that are mapped to the Xbox controller's right analogue stick.

Oculus - Real Time Gravity Simulation.exe

The CycleInitialConditons.cs script has public members which are visible in the inspector window in the Unity editor. The values selected in the inspector override those initialised in the script when the program is launched, so one may ignore any initialisation written in the script. Note that private members do not appear in the inspector so are not overridden. The purpose of this program is to initialise NBodySim.cs and then run it. It has public functions that disable NBodySim.cs, re-initialise it and then launch it again. Using these functions, the program is able to change what initial condition is output to the screen. CycleInitialConditons.cs also collects and stores files form /StreamingAssets/ on launch.

NBodySim.cs essentially sets up a distribution then manages its subsequent evolution. If it is reading from a file, it uses the csvFileReader.cs script to parse the file into two 4 vectors, one for position and mass $[x,y,z,m]$ and one for velocity $[v_x,v_y,v_z,u]$, where u is an empty parameter set to 0. An array of these vectors, holding data for every particle, is then passed to a compute buffer which is dispatched to IntegrateBodies.compute.

IntegrateBodies.compute calculates the forces on each particle then updates the positions and velocities of each one. This is done in parallel to make use of the computing power of the GPU.

After IntegrateBodies.compute updates the particle data, NBodySim.cs passes the data to SpriteParticleShader.shader. This divides the magnitude of each particles velocity by some arbitrary maximum speed that can be set in the inspector of CycleInitialConditons.cs or modified during runtime using inputs. The result is a number, μ, between 0 and 1. The red, green and blue colour channels for the particle are each functions of μ, chosen to create a contrasting and intuitive colour scale. The particles are then rendered to the screen at their coordinates.

Click.cs, KeyboardMovement.cs and SimpleXboxControllerInput.cs detect inputs from the relevant peripherals and call functions in NBodySim.cs and CycleInitialConditions.cs to the effect of those described in Section III. To reference input from the triggers and right stick of the Xbox controller, three custom axes had to be defined in the Input Manager [12].

Oculus – 3D Plot.exe

3D Plot works in almost exactly the same way as Real Time Gravity Simulation, having the exact same structure of objects and scripts. Apart from member names, there are a few important distinctions.

CyclePlots.cs cycles through files only, and does not command NBodyPlotter to load hard coded distributions. It is therefore critical that there is at least one file in /StreamingAssets/ before launching 3D Plot. When NBodyPlotter.cs loads a file for the first time, the initial parsing of a file using .csvFileReader.cs is slow, especially for large distributions. To mitigate this issue, NBodyPlotter.cs uses the custom class "Plot"

(defined in Plot.cs) to store the particle data. The Plot class contains 3 members, two Vector4 arrays for the position and velocity data, and a string for the filename. When NBodyPlotter.cs begins plot from file via the ConfigFile() function, it first checks the name of the file against the list of stored Plot objects. If it has previously been stored, it loads the particle data from there instead of parsing a .csv.

NBodyPlotter cannot store anything for the full duration of runtime since the script is enabled and disabled during runtime. Therefore the list of Plot objects is stored in CyclePlots.cs instead.

GPUNeighbourCalculator.compute is a modified version of IntegrateBodies.compute. It is intended to take advantage of the parallel computing structure of IntegrateBodies.compute in the absence of compute shader knowledge. The force calculation code inside the function bodyBodyInteraction() is replaced with code that counts the neighbours around a given particle. Some variable types passed between functions are changed too, however the overall structure of the compute shader is untouched. The result is a fast calculation, though not fast enough to be called every frame. Instead, the GPUNeighbourCalculator.compute is only run when a distribution is plotted or upon input from the controller or keyboard.

The rest of the scripts work as in Real Time Gravity Simulation.

VII. ISSUES

Each program comes with its limitations, some of which have been mentioned before but all of which will be summarised here.

The Real Time Simulation can only handle ~50,000 particles. More than this and the framerate begins to drop which is very uncomfortable for users wearing the Oculus Rift. The 3D Plot can handle on order of 10^6 particles before frame drops occur, however the .csv parsing scripts are very simple and very slow. Having many files of that size has not been tested but it is predicted to reduce the ergonomics of the program. The fragility of the parsers also places a strict demand on the format of the .csv files.

IntegrateBodies.compute is mostly untouched from the port provided by Scrawk on GitHub. However, the first attempts to use it in Unity yielded fatal physical errors. This may be due to the porting process done by Scrawk, or because the NVIDIA optimised their N-body simulation for the GeForce 8800 GTX, a GPU that is now 10 years old (and different to the one used for this project). Some experimentation with the variables passed to the compute shader proved successful in restoring correct physics. Specifically, the "tile calculation" method detailed in the article has the "tile height" of 4 which is set in NBodySim.cs. Reducing the tile height to 1 produced correct physical interactions but is expected to have reduced the parallelism of the compute shader. There is also structure for multithreading within IntegrateBodies.compute but it is disabled as its operation are not understood.

Due to computing power issues, it was also not possible to implement nearest neighbour counting in real time. It would have been a more effective way to highlight orbital subsystems of commoving particles, for example in the branches of the cube distribution explosion.

The process in place to reduce file load times in 3D plot only works while 3D Plot is running. If the program is closed (for example to run Real Time Gravity Simulation instead), the list of Plot objects will be lost. Therefore files will need to be parsed again when 3D Plot is launched again, taking away from the program's speed.

VIII. REASEARCH AND SUGGESTED EXTENSIONS

Many concepts explored during the creation of the two programs were deemed too difficult or too time consuming to complete alongside. An account of these concepts is discussed here, as well as some suggestions on how to the finished programs could be improved.

The Oculus Rift is an excellent tool for experiencing VR, but is expensive. An alternative is a cheap android smart phone in combination with Google Cardboard [13]. Unity's advantage as a game engine is its ergonomics in producing cross-platform compatible projects so, although a some amount of extra coding would be required to make a project fully cross-platform between Windows and Android, the technical ability required is not much more than the scope of this project. However, for computing power reasons, it is not possible to transfer either Real Time Gravity Simulation or 3D Plot to Android (or iOS). A potential option would be to stereoscopically record a fly through 'tour' of the programs which would then just be saved to the smartphones as videos, much like the stereoscopic videos found on YouTube [14]. If a video across multiple smartphones were to be played simultaneously, for example for a school demonstration, a specialised

app would be required, although it's likely one already exists. A potential drawback is an increased risk of nausea with cardboard-type headsets or low resolution smartphone screens.

Iso-density contour surfaces are a way of visualising largescale astronomical objects in 3D. To reproduce this type of graph in Unity, each individual contour would need to be a game object with a textured mesh of triangles that make up the surface. Furthermore, Unity only textures one side of a mesh (the other being invisible) to reduce processing on the GPU, so an iso-density contour would need to be comprised of a double-mesh, one with the texture facing outwards and one with the texture facing inwards. Unfortunately, Unity does not have a good method of procedurally generating an arbitrary shape. Using Unity alone, a mesh demands scripting the position of every vertex and then drawing triangles between them by specifying edges individually. Unity is far better suited to importing meshes created in external programs, for example Blender.

Real Time Gravity Simulation and 3D Plot are written a PC with a GTX 980 Ti GPU. The IntegrateBodies.compute shader could be rewritten to optimise the power of the GTX 980 Ti GPU and/or make better use of the "tile calculation" method described in the GPU GEMS article. Some quick research [15] returned suggestions that the GTX 980 Ti is 10x more powerful than the GeForce 8800 GTX. Any increase in computational power would allow the particle limit to accommodate a wider range of real astronomical data. If the particle limit is retained or reduced, some alternative real-time physics calculations could be performed alongside force calculations. For example: Periodic boundary conditions.

Increasing the .csv file readers' speed and robustness would allow both programs to run slightly faster, it would allow more files to be held in 3D Plot's /StreamingAssets/ without the program being difficult to load, and would relax conditions placed on the department to make sure .csv files are perfect.

Adding a GUI to both programs could help navigation of each one, as well as open up doors to add interesting statistics on screen for users to see. To increase user interaction with the simulation, it is possible to add a function to create particles upon user input, much like in Universe Sandbox [16]. Particles would need to be created in clusters of 256, however, due to the structure of the IntegrateBodies.compute.

Finally, it is possible to merge 3D Plot and Real Time Gravity Simulation into one program, using Unity's "scenes" feature. Each program has only one scene which fully describes the relevant game objects and attached scripts. A single program alternative would have both scenes in the /Assets/Scenes/ folder, as well as a script that takes input from the controller or keyboard that switches between scenes during runtime. This would not only be more ergonomic for anyone using the program, but would also solve the issue of losing vector-stored files when closing 3D Plot to run Real Time Gravity Simulation.

IX. Conclusion

The future of VR in education and outreach relies on innovation of its available features. Its unique introduction of depth perception into virtual media allows for demonstrations not possible in the physical world. For example, one can imagine visualising the optical properties of a transparent material by having the refractive index a variable parameter. The depth of an object inside the material would appear to change, a subtlety that may be lost on a screen, but not in VR.

For this reason, VR is a versatile medium that, with Real Time Gravity Simulation and 3D Plot, provides a novel way to demonstrate some aspects of classical mechanics to lay people and prospective students. The programs successfully create a prototype bridge between the physics department's data and outreach events. The programs are relatively simple, but were easily built at an undergraduate technical level. The limitations reached in their design are likely to be overcome with the application of greater experience in the relevant programming languages. Many features and extensions could be put together with time alone, perhaps in a Master's project or a future internship.

Unity has proven to be a flexible and easy-to-use platform for building programs for VR. Many desirable features for an outreach program are simple to implement using Unity. For example, position based audio prompts for audio tours, rollercoaster-like scripted camera motion, GUIs with information or scripting actions and controller based interaction.

X. REFERENCES

[1] ‘Davidbau.Com Conformal Map Viewer’ (Davidbau.com, 2017) <http://davidbau.com/archives/2013/02/10/conformal_map_viewer.html> accessed 20 September 2017

[2] ‘Clarke's Three Laws’ (En.wikipedia.org, 2017) <https://en.wikipedia.org/wiki/Clarke%27s_three_laws#cite_note-2> accessed 20 September 2017

[3] Clarke A, Profiles Of The Future (Pan Books 1973)

[4] 'Unity - Game Engine' (Unity, 2017) <https://unity3d.com/> accessed 20 September 2017

[5] 'Unity - Interface Overview' (Unity, 2017) <https://unity3d.com/learn/tutorials/topics/interface-essentials/interface-overview?playlist=17090> accessed 20 September 2017

[6] 'Unity - Roll-A-Ball Tutorial' (Unity, 2017) <https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial> accessed 20 September 2017

[7] 'Unity - Scripting' (Unity, 2017) <https://unity3d.com/learn/tutorials/s/scripting> accessed 20 September 2017

[8] 'Scrawk/GPU-GEMS-Nbody-Simulation' (GitHub, 2017) <https://github.com/Scrawk/GPU-GEMS-NBody-Simulation> accessed 20 September 2017

[9] 'GPU Gems - Chapter 31. Fast N-Body Simulation With CUDA' (NVIDIA Developer, 2017) <https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch31.html> accessed 20 September 2017

[10] 'Jakedowns/GPU_GEMS_OVR' (GitHub, 2017) <https://github.com/jakedowns/GPU_GEMS_OVR> accessed 20 September 2017

[11] 'A Free Simple Smooth Mouselook' (Unity Forum, 2017) <https://forum.unity.com/threads/a-free-simple-smooth-mouselook.73117/> accessed 20 September 2017

[12] 'Unity - Manual: Input Manager' (Docs.unity3d.com, 2017) <https://docs.unity3d.com/Manual/class-InputManager.html> accessed 20 September 2017

[13] 'Google Cardboard – Google VR' (Vr.google.com, 2017) <https://vr.google.com/cardboard/> accessed 20 September 2017

[14] 'If You Could See All The Asteroids, What Would The Sky Look Like?' (YouTube, 2017) <https://www.youtube.com/watch?v=huC3s9lsf4k> accessed 20 September 2017

[15] 'Userbenchmark: Nvidia Geforce 8800 GTX Vs 980 Ti' (Gpu.userbenchmark.com, 2017) <http://gpu.userbenchmark.com/Compare/Nvidia-GTX-980-Ti-vs-Nvidia-GeForce-8800-GTX/3439vsm9271> accessed 20 September 2017

[16] 'Universe Sandbox' (Universesandbox.com, 2017) <http://universesandbox.com/> accessed 20 September 2017