THE UNIVERSITY OF QUEENSLAND

Visualisation of Orbital Object Uncertainty Using Generative Particle Methods

by
Joel Stuart
Supervisor: Dr. Marcus Gallagher

School of Information Technology and Electrical Engineering,
The University of Queensland.

Submitted for the degree of
Bachelor of Engineering
in the field of Software Engineering

31st October 2016

Prof Michael Bruenig
Head of School
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia, Q 4072

Dear Professor Bruenig,

In accordance with the requirements of the degree of Bachelor of Engineering in the division of Software Engineering, I present the following thesis entitled "Visualisation of Orbital Object Uncertainty Using Generative Particle Methods". This work was performed under the supervision of Dr Marcus Gallagher.

I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Queensland or any other institution.

Yours sincerely,

Joel Stuart.

Acknowledgments

I would like to take this opportunity to thank Dr. Marcus Gallagher for his role as my supervisor, despite his numerous other commitments and students. I would also like to extend my gratitude to my previous supervisor Dr. Tyler Hobson, without whose initial guidance, this project would not have been possible.

Finally, I would like to thank Mr Jovan Harrod and Prof. Mehmet Kizil for their individual assistance with the project, and for sharing their toy with me. I am very grateful for the guidance and kindness I have received throughout the entirety of this project. For all those I have worked with throughout this project, it has been a pleasure.

Abstract

This project creats two visualisation tools and a companion close approach analysis tool using the common TLE/SGP4 approach. It investigates the use of generative Monte-Carlo particles methods and found it to be a useful alternative to the typical Gaussian error volume, particularly when TLE data has aged.

The created visualisations highlight the quick degradation of TLE sets and the need for TLE correction methods, a current area of study. Finally, this project investigated the academic usefulness of Virtual Reality and found VR to be an effective and engaging presentation tool, conferring depth and scale far greater than is possible in other visualisations.

Contents

List of Figures

List of Tables

Chapter 1

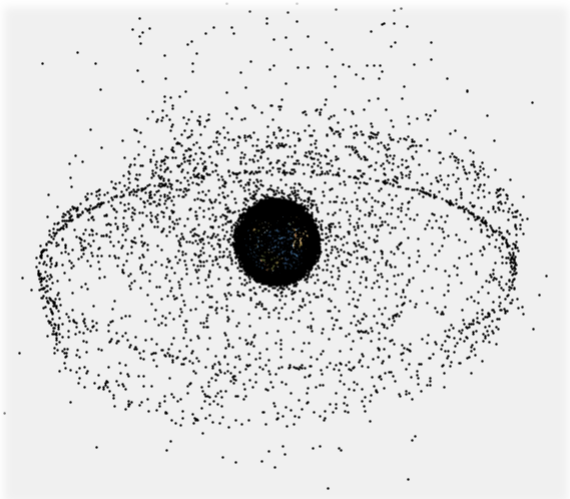
Introduction

Since the beginnings of humanity's operations in space in the 1950s, the space around Earth has been increasingly congested with man-made space debris.

This debris consists of defunct spacecraft, jettisoned and otherwise mission-related debris and the fragmentation debris created when existing debris collide. Fragmentation debris make up the largest share of the debris, and is expected to further inflate as more man-made debris is created with which existing debris may collide with [1].

Due to the speeds space debris travels at, even tiny fragments of paint can cause significant damage to operational satellites [2]. As of 2013, there were more than 500,000 fragments of space debris being tracked [2], with millions more pieces too small to feasibly observe and catalogue.

Figure 1.1: MATLAB Visualisation of catalogued orbital objects surrounding earth



In Australia, Optus is responsible for a large portion of the telecommunications infrastructure. According to the Australian Space Environment Research Centre (SERC) their fleet of geostationary satellites has a commercial worth of over $8 billion [3]. This entire fleet must be monitored to mitigate risks of collision with debris and avoid loss of assets.

An impact with another orbital object can cause catastrophic damage to an operational satellite, such as shown by the 2009 collision between the active Iridium 33 satellite and the non-operational Kosmos 2251, producing many thousands of pieces of debris larger than 1cm [4]. This debris will continue its orbit for decades to come, as space debris cannot currently be plausibly removed from orbit [4]. Thus, Space Situational Awareness (SSA) and Space Surveillance and Tracking (SST) are essential to protecting and maintaining satellite systems.

Chapter 2

Background

The following Chapter intends to briefly introduce the broad area of research to which this document relates, as well as cover some of the approaches and technologies used to explore it. This chapter will specifically discuss Space Situational Awareness (SSA) and its subsidiary, Space Surveillance and Tracking (SST), representations of uncertainty for orbital objects, and some of the existing technologies which this project will utilise.

2.1 Space Situational Awareness (SSA)

Space Situational Awareness refers to both the field of study and the ability to track and predict the physical locations of natural and manmade objects in orbit around Earth, with the intention of avoiding collisions.

SSA has become a significant area of concern due to the increasing quantity of objects in orbit and the increasing investments into and reliance upon satellite technology.

2.2 Space Surveillance and Tracking (SST)

To attempt to address the massive problem of managing and tracking the countless orbital objects, catalogues of debris fragments, active and inactive satellites are maintained by international agencies. One such database named Celestrak, which is publicly posted and maintained by CSSI (Centre for Space Standards and Innovation), contains an extensive catalogue of NORAD (the North American Aerospace Defense Command) tracked objects stored in a format known as Two-Line Element

(or TLE).

These catalogues can be used to analyse, simulate and predict various future scenarios to assess potential risks to assets.

These simulations and subsequent analysis lead to greater threat identification, preparation and responsive deployment of avoidance measures.

2.3 Representations of Uncertainty

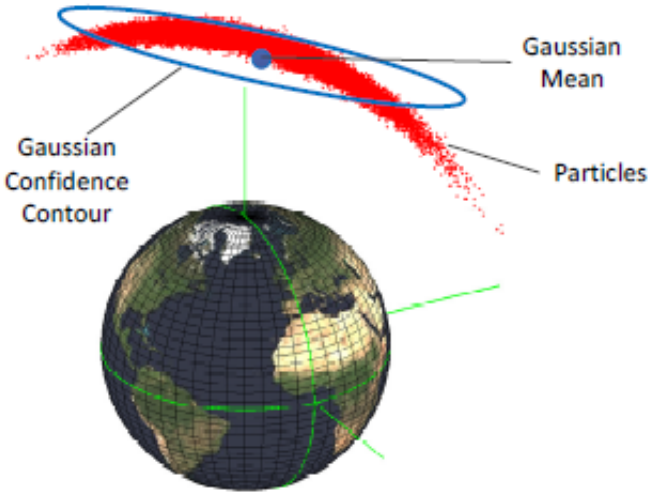
As with any simulation based on an observation, uncertainty and error of measurements must be considered when analysing simulated results. Traditional analysis of orbital collisions (often referred to as conjunctions) asserts that orbital observation uncertainty will distributed by a Gaussian model. Thus the standard error volume is that of an ellipsoid volume centered on an observation.

This assumption suppresses the true non-Gaussian nature of satellite position uncertainties [8, 10]. A proposed solution for this shortcoming is to treat the non-Gaussian error model as a sum of Gaussian elements, modeling variables such as state noise or measurement noise, which can be assumed to be Gaussian [11, 12]. This is known as the Gaussian Sum or mixed-Gaussian method.

Similarly, a particle representations can model the non-Gaussian nature of the object. Particle methods utilise Monte Carlo sampling to, effectively, represent uncertainty to a desired level of fidelity based on the number of samples used (where each sample becomes a new 'particle' within a collection). This resulting cloud of particles can then represent the possible states and thus the uncertainty for any orbital object. [8].

Particle methods are particularly suited to applications with high rates of false detections and extended periods between observations, where a Gaussian-distributed error volume may deviate significantly from the true probability distribution. This point is demonstrated in Fig 2.1, illustrating the discrepancy between a low fidelity Gaussian volume and a higher fidelity particle distribution method.

Figure 2.1: Guassian and particle rerpesentations of an orbital object's uncertainty



Whilst particle methods are significantly more computationally expensive than their Gaussian counterparts, for SSA purposes, the additional fidelity may be desirable.

2.4 SGP4

Over 35 years ago, the United States Department of Defense(DoD) released a document SpaceTrack Report Number 3[5]. Included in this document was the source code used to predict future orbital object positions. The code became known as Simplified General Perturbations-4 (SGP4).

Since the release of this source code, many changes have been made to the official code, which have not been released to the public.Through the many individual efforts from within the scientific community, a non-proprietary version has been created which is highly compatible with its current DoD brethren [7]. As of today, SGP4 is available publicly in several coding languages including Fortran, Python, C++ and MATLAB.

Using these free software libraries, and using Two-line Element data sets available online, it is possible to predict the future locations of all orbital objects in a given SST catalogue.

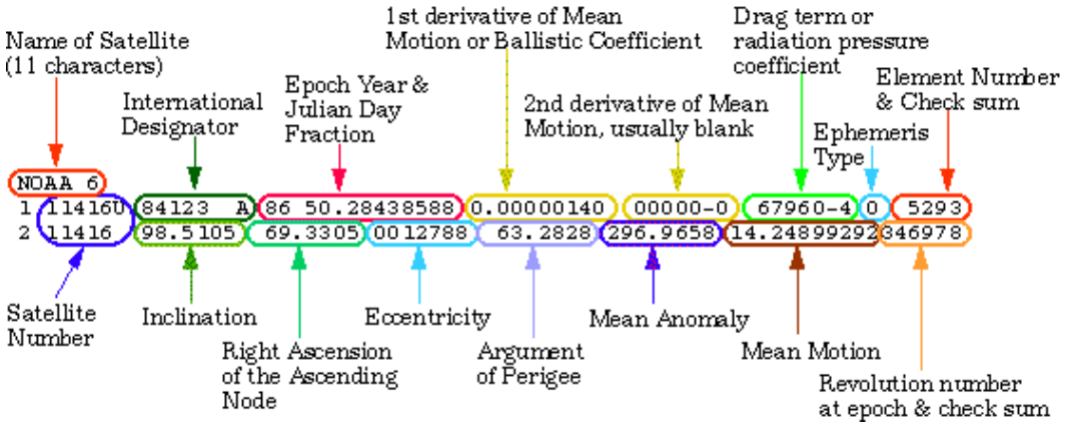
2.5 Two-line Element

Two-line Element (TLE) is the de facto data representation of an orbital object's Keplerian parameters and is the native data format used with SGP4. The format

consists of two lines of ASCII text, each with a length of 70 characters. Some TLE sets contain a title line preceding the TLE data for human readability, but this is not required, as each data line contains a satellite identifier.

Seen in Figure 2.2 below is an example TLE, with the encoding annotated.

Figure 2.2: Annotated Two-line Element Format, image credit to NASA[9]

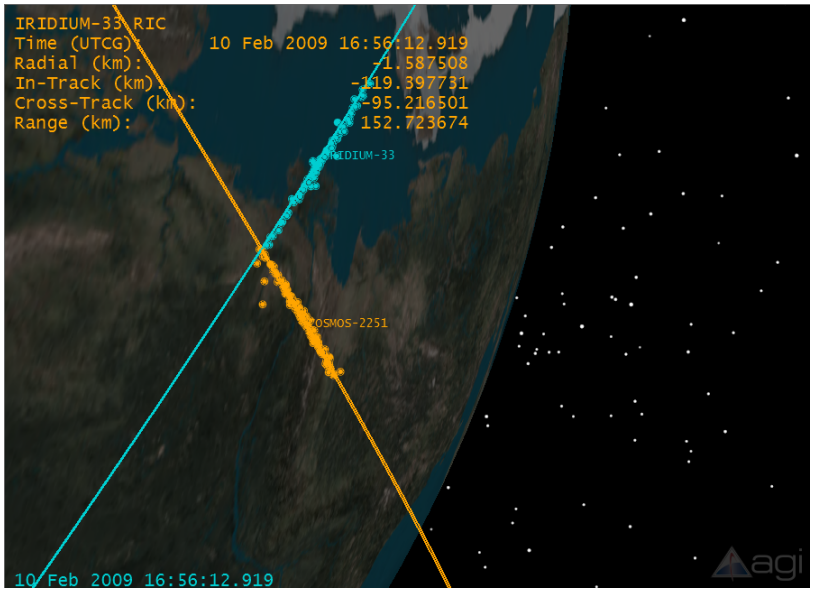


2.6 Existing SSA Visualisations

The main proprietary SSA Visualisations are made by Analytical Grapihcs, Inc. (AGI). The only free offering from AGI is the Systems Tool Kit (STK), which has tools for modelling, analysing and visualising time-dynamic positions of objects.

An example visualisation is shown in Figure 2.3, showing the collision of the active Iridium-33 and inactive Kosmos 2251 satellites on the 10th of February, 2009.

Figure 2.3: Visualisation of Iridium-33 and Kosmos 2251 Collision using Systems Tool Kit



2.7 Virtual Reality and HTC vive

The recent release of the HTC Vive has enabled interactive Virtual Reality (VR) to be experienced by a wide audience. The HTC Vive itself is a headset and hand-held controller combo that delivers head tracking and stereoscopic 3D visuals on the headset and motion tracking on the wireless controllers.
The most useful quality this experience offers is the immediate impression of scale and presence afforded by the stereoscopic 3D view. The effect itself is profound enough to induce phobias of heights in users [13].

While still a fledgling technology, wide-spread adoption and development by major technology companies has lead to extensive VR support in most major graphics engines. This support, coupled with a booming open source development scene, makes creating VR experiences a fairly approachable endeavor.

2.8 Review

Within the broad range of aforementioned topics, there is a significant area of investigation encompassing the following:

- Investigation of non-Guassian representations of orbital object uncertainty using generative particle methods.

- Investigation of the academic applications of cutting edge VR technology to interactive visualisations.
- Creation of SSA related visualisations, facilitating further investigation into orbital phenomena.

2.9 Thesis Definition and Scope

Thus, using TLE data from publicly available SST catalogues and the SGP4 propagation model, a visualisation software package will be created to explore the above areas of investigation.

With regards to scope, this thesis will not:

- Delve deeply into advanced astrophysics theory. It will instead make use of well established astrophysics software libraries in conjunction with sound engineering practices.
- Discuss in rigorous detail the sensor networks that provide the observations contained within the TLE catalogues.

Conversely, this thesis will:

- Discuss the engineering practices demonstrated whilst designing,developing and validating the intended visualisation package.
- Discuss the systematic approaches applied in this investigation in response to challenges and problems.
- Discuss in detail the above identified areas of investigation, which may be lacking in the current literature.

Chapter 3

Methodology

3.1 Overview

There are three seperate programs that were created for this project:

- A simulation and visualisation tool built in MATLAB, which handles the input files and calculation of orbital state vectors at various times and can output to network and to file.
- An interactive 3D visualisation built in Unity, which is streamed data from MATLAB and which supports the HTC Vive VR Headset.
- An analysis tool built in Java, which reads in data stored in the output files created by the MATLAB component.

3.2 Data and Online Catalogues

To source TLE datasets, the public catalogue of Two-line Element data sets known as Celestrak was used. Celestrak's primary source of observations is NORAD (the North American Aerospace Defense Command). NORAD tracks and maintains TLE sets for all resident space objects over a certain size. [5]

Similarly, further customised TLE sets can be obtained from the likewise affiliated online database known as Space Track (after creating an account). [6]

3.3 MATLAB Simulation

As discussed above, the MATLAB simulation and visualisation tool reads input TLE set files and calculates future state vectors for each object in the input file. These

state vectors are then presented in an animated visualisation, and may be stored in file or sent over network.

Thus, the tasks handled by the MATLAB program can be broken down into the following:

- TLE Reading
- Particle Generation
- SGP4 Propagation
- MATLAB Visualisation
- Output to file / network

3.3.1 TLE File Handling

Using the Celestrak database, TLE datasets can be downloaded and stored in plain text files. Processing of these plain text files with TLE data is a matter of reading each line and storing these lines, to be later processed. The file handling routine can be found in Appendix A.1.

These lines are later processed through the the `tleToSatrec` routine, which creates a structure data type 'satrec' which contains the contents of the TLE file seperated into their respective fields. Usage for this routine is as follows:

[satrec] = **tleToSatrecO**(*gravConst, line1, line2, runType, typeInput, startmfe, stopmfe, deltamin*)

Where **gravConst** is the gravity constant used (from a selection of constants of varying fidelity),
line1 and **line2** are the TLE lines
runType and **typeInput** are various methods within SGP4 (manual mode 'm' and a customised input 'j' were used)
The **startmfe** and **stopmfe** are the supplied start and stop times
And **deltamin** is timestep value.

3.3.2 Particle Generation

Particles, p^r, are generated by adding a noise vector ϕ^r (modeling the error as per [8]), to their initial state x_i , read from the TLE. This can be seen in equation:

$$p_r = x_i + \phi_r \tag{3.1}$$

Where ϕ_r is the noise vector defined as

$$\phi_r \sim N(0, \gamma_r^2); \gamma_r^2 = diag \begin{bmatrix} 0.0005^\circ \\ 0.0005^\circ \\ 0.00001 \\ 0.0005^\circ \\ 0.00001^\circ \\ 0.00001 orbits/day \end{bmatrix} \tag{3.2}$$

This above noise / error vector was used in [8], found empirically to model appropriate levels of perturbations experienced by orbital objects.
All particles are then propagated forward in time using the SGP4 model.
The full particle generation routine, which is tied to the tleToSatrec routine detailed above can be found in Appendix A.2.

3.3.3 SGP4 Propogation

Now that the **satrec** structure has been created for each object, **satrec** can be passed to the **sgp4** function, along with the time (measured in Julian Days) at which to calculate the state vectors of the object. This is performed for every particle of every object, at every desired time. These results can then be visualised.

To improve performance, these routines were adapted for parallel computation and performed on the GPU via nVidia's CUDA environment within MATLAB's Parallel Computing Toolbox. This will be discussed further in the GPU Parallelism Section 4.3. Furthermore, due to the large number of calls to these routines, they were optimised slightly by replacing computationally expensive operations such as multiplication and squaring with pre-computed constants where possible.

3.3.4 Visualisation

Using the MATLAB built-in plotting function **scatter3** (the 3D variant of the **scatter** plot function), alongside a textured 3D mesh of the earth, a snapshot of the locations of all the input orbital objects can be visualised. This is achieved with the following psuedocode:

Algorithm 1 Built-in MATLAB visualisation Pseudocode Snippet
Where o is an abritrary orbital object, S is the set of all objects in the input dataset

xArray = xComponent $\forall o \in S$
yArray = yComponent $\forall o \in S$
zArray = zComponent $\forall o \in S$

hold on

scatter3(xArray, yArray, zArray)

eTex = imread('earthmap.jpg')
[x1, y1, z1] = sphere(100)
earth = surf(x1*6000,y1*6000,z1*6000, 'EdgeColor','none')
set(earth,'CData',flipud(eTex),'FaceColor', 'texturemap')

cameraInit(angles)

This static visualisation can then be animated by sequentially changing the supplied position arrays for each frame, at some arbitrary rate of frames per second (frame rate of animation).

This is achieved with the MATLAB built-in **set** which allows a plot to be supplied new data, **drawnow** which updates the rendered plot, and **pause** which pauses for a time measured in seconds.

Algorithm 1 is extended with the following to achieve an animated visualisation:

Algorithm 2 Built-in MATLAB Animation Pseudocode Snippet
Where o is an arbitrary orbital object, S is the set of all objects in the input dataset

```
while Animating do
    tic (begin timing)

    xArray = nextXComponent ∀o ∈ S
    yArray = nextYComponent ∀o ∈ S
    zArray = nextZComponent ∀o ∈ S

    h = reference to scatter plot

    set(h, 'xdata', xArray, 'ydata', yArray, 'zdata', zArray)
    drawnow
    toc (take a measurement of the elapsed time since tic was called)

    pause(1/fps-toc)
end while
```

3.3.5 UDP Communication

Using the built-in **UDPSender** object from the MATLAB Digital Signal Processing Toolkit (installed by default), data can be sent over User Datagram Protocol (or UDP), facilitating real-time communication between programs either locally or over network.

UDP was chosen over TCP/IP to minimise network overhead and because packet loss was desirable over waiting for delayed packets.

The **UDPSender** object was chosen over the alternative MATLAB **udp** object, for it's maximum output buffer size (twice that of **udp**)).

Usage within MATLAB is as follows:

- Initialise destination with **UDPSender** object.
- Explicitly convert data to be sent into a binary format (e.g using **int16** or **int32**).
- Send the data through the UDPSender object using the function **step**.

- Release the UDPSender object using **release** (allowing different format data to be sent).

In pseudocode, this is:

Algorithm 3 MATLAB UDP Psuedocode

Where *objs* is the data object to be sent

```
hudps = dsp.UDPSender(RemoteIPAddress, RemoteIPPort)
data = convert objs to desired binary representation (i.e. int16(objs))
step(hudps,data)
release(hudps)
```

3.4 GPU Parallelism

With the set goal of creating a real time visualisation, the problem of computing computationally intensive routines for a large dataset was a significant technical challenge. Fortunately, the computation of each object is independent and thus can be calculated in parallel (or as close to parallel as the computation hardware would allow).

Thus, using nVidia's CUDA Parallel Programming environment and a low-end nVidia card, these independent operations can be paralellised, contributing to a considerable speedup. Thanks to support for CUDA within MATLAB's Parallel Computing Toolbox, this parallelisation can be accomplished mostly within MATLAB by adapting the existing code.

The primary chunk of code to parallelise was the SGP4 routine for each object. This meant performing both the SGP4 initialisation (sgp4init)and the SGP4 propagation function (sgp4) for each timestep for each object on the GPU.

Thus, the steps needed to parallelise the code were as follows:

- Adapt the SGP4 code to the programming language C. Fortunately, a version of SGP4 already exists written in C, which was used.
- Split the GPU up into blocks upon which the parallel calls can be processed.
- Write the CUDA function to run the sgp4init and sgp4 for each object and timestep in using the above blocks. This CUDA function can be called from within MATLAB.

- Pass the GPU the data structure **satrec** for each object. Since passing data structures between languages is messy, the required contents of the data structures were passed as flat arrays with the index of the array corresponding to an object.
- Retrieve the results from the GPU, stored in a flat array storing the x,y and z components of each object.

The MATLAB code snippets implementing this can be found in Appendix A, Figure A.3, and the CUDA code can be found in Appendix A, Figure A.4.

With the speedups afforded by the above parallelisation(for full speedup data, refer to the Section 4.3), the MATLAB visualisation moved from a strategy of "precompute, then display results sequentially" to a strategy of "compute once every **X** frames, then display those **X** frames", thus allowing an endless, real time visualisation. The strategy of computing **X** frames is to minimise delays due to computation.

3.5 Unity Visualisation

To overcome the constraints of the MATLAB visualisation, a companion visualisation was built in Unity. The allowed greater control over all aspects of the visualisation but primarily relating to the graphics pipeline, camera control and user interactivity.

3.5.1 Receiving Data over UDP

Using the above methods to send data to a remote IP Address and port via UDP, position data for each frame was able to be sent from an instance of MATLAB to an instance of Unity. This data was received and processed in a separate 'slave' thread.

During the initialisation of communications, the total number of objects (inclusive of the number of particles for each object) is sent by from MATLAB and stored by Unity. After this number has been stored, the master thread instantiates that many objects and stores a reference to them.

All future communications are then expected to be frame updates. Each frame update is then decoded and stored in a queue accessible by the main thread. Code snippets of this routine can be found in Appendix Figure A.5.

3.5.2 Animation

Using the built-in Unity function **Update()** which is by default called before every frame rendered in Unity, if the MATLAB frame queue is not empty, the top most entry is popped from the queue and processed.

This processing reads the changes for each object from the entry and updates the corresponding the position vector for each object, using the reference instantiated on initialisation.

3.5.3 Virtual Reality with HTC Vive

Thanks to the design of plugins within Unity, adding support for Virtual Reality within Unity is a trivial task. The primary plugin used was SteamVR which assists in connecting a Unity instance to an local instance of Steam (a software distribution platform), which acts as a hub or 'menu' for the VR device (HTC Vive in this case).

3.5.3.1 SteamVR

Using the aforementioned SteamVR plugin, synchronising the visualisation camera and the VR headset is as simple as adding a prefab asset (a predefined entity) into the Unity environment. This asset also gives access to controller inputs which can be coded to provide interaction. In this project, controller interaction was limited to teleportation movement.

3.5.3.2 Teleportation

Using a modified open source framework, the HTC Vive controllers were configured to allow the user to teleport around the 3D environment. This was an extension of the real movement of the user in the real room, which is a bounded area. With the teleportation implementation, the user is able to move where the bounded area is in the digital space. This has become somewhat of a widely adopted method of locomotion in VR simulations, as automatic or guided locomotion can induce motion-sickness in users.

Working in tandem with this script was an invisible 2D plane running through with centre of the earth and covering the entire environment, which acts as the collision plane (the area to which the player may teleport).

PLEASE NOTE: the implementation of this script is not the author's work and full credit for this open source framework is given to the rightful creator [14].

With the above implementations, the Unity visualisation was enhanced by the perception of real movement within the digital environment and stereoscopic 3D.

3.6 Close Approach Analyser

To avoid hindering the performance of either the MATLAB or Unity visualisations, it was decided to build the analysis tool separately, as a companion tool to be used post-simulation. This goal of this tool was to use the propagated position information from the MATLAB simulation to perform calculations as to the chance of any collisions between every unique pairing of objects (and particles not of the same object).

Due to the author's comfort with the Java language, it was chosen to quickly build the analysis tool. This brough the total programming language count used in this project up to four.

Using the output files created by the MATLAB simulation (containing all propagated particles and objects at all timesteps the simulation calculated) as the input for this tool, each time-step was checked for close approaches by comparing comparing the close approach threshold radius to the distance calculated using the Euclidean distance formula:

For two points $P_1 = (x_1, y_1, z_1), P_2 = (x_2, y_2, z_2)$,

$$D(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{3.3}$$

Initially, the close approach threshold radius was an estimate of the average size of a satellite, but this was recognised as a naive approach, as the fidelity required for this calculation to be valid was completely unfeasbile. Thus the close approach threshold radius was set between 15 and 30km. This value was chosen based on the age of the TLE file used. The age of a TLE file can be found by taking the difference between the simulation time and the time since the epoch (reference date) of the TLE.

The close approach threshold range of 15-30km was reasoned as appropriate, given the estimated accuracy of an average 3 day old TLE set is 30km [15]. It is worth

noting that accuracy of TLE sets have been found to vary wildly from source to source, with some sources having accuracy down to 1km at epoch and as high as 300km at epoch. Historically, TLE's have been shown to occasionally exceed the characteristicly estimated accuracy by a factor of twenty when compared to positionally well-known satellites (e.g. tracked by GPS). [16].

Chapter 4

Results

Links to videos of both the MALTAB and Unity visualisations as well as source code can be found in Appendix B.

4.1 MATLAB Visualisation

A number of visualisations of different catalogues with various configurations of particles will be shown. The various catalogues correspond to different groupings of objects, grouped by orbit classifcation. These orbits are introduced without particles showing error, and then with particles to demonstrate the uncertainty in each catalogue. Note the along-track variance (variance in distance along the direction of the velocity vector).

4.1.1 Catalogue Visualisations without Particle Generation

The following figures show snapshots of various TLE catalogues visualised without particle clouds representing uncertainty volumes. Figure 4.1 seen below shows the current complete catalogue of 15500 objects being tracked.

Figure 4.1: MATLAB Visualisation of 'Complete' TLE Catalogue

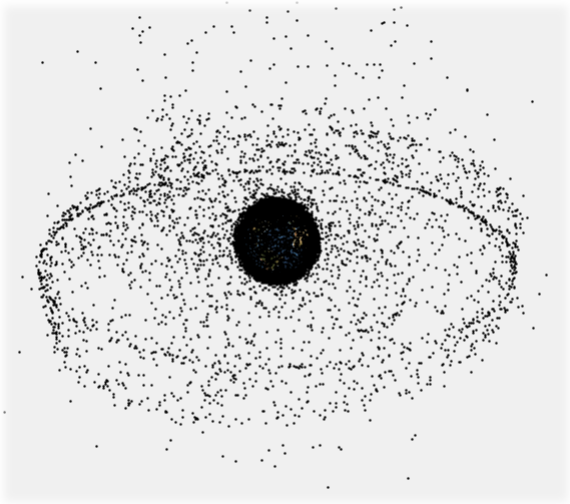


Figure 4.2: MATLAB Visualisation of Low Earth Orbit Catalogue

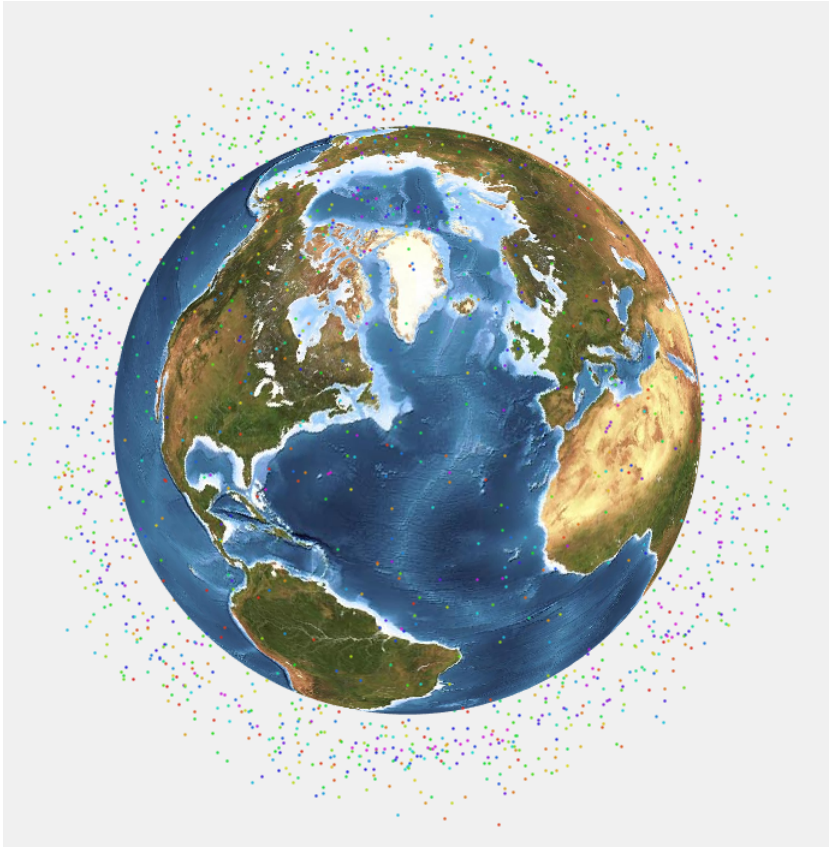


Figure 4.3: MATLAB Visualisation of Highly Elliptical Orbit Catalogue

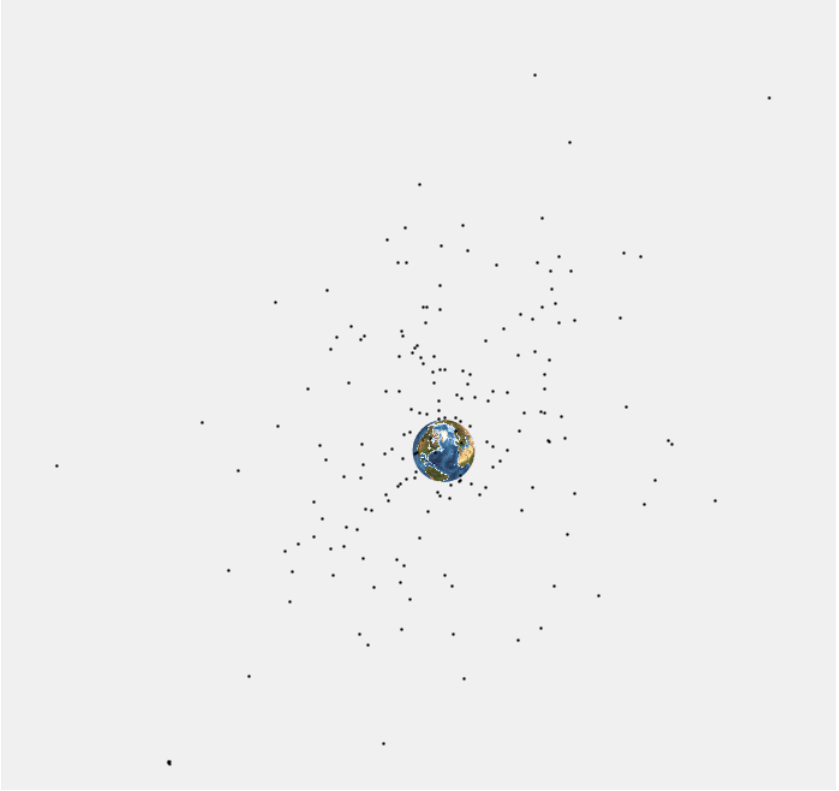


Figure 4.4: MATLAB Visualisation of Medium Earth Orbit Catalogue

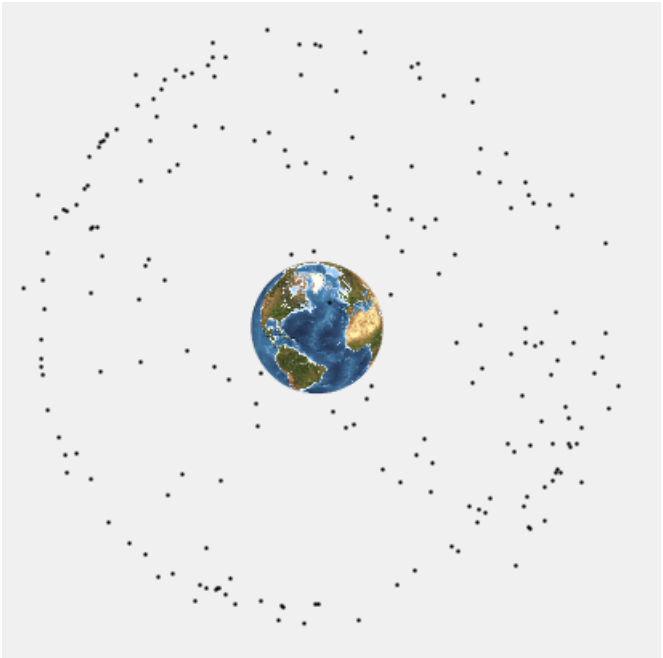
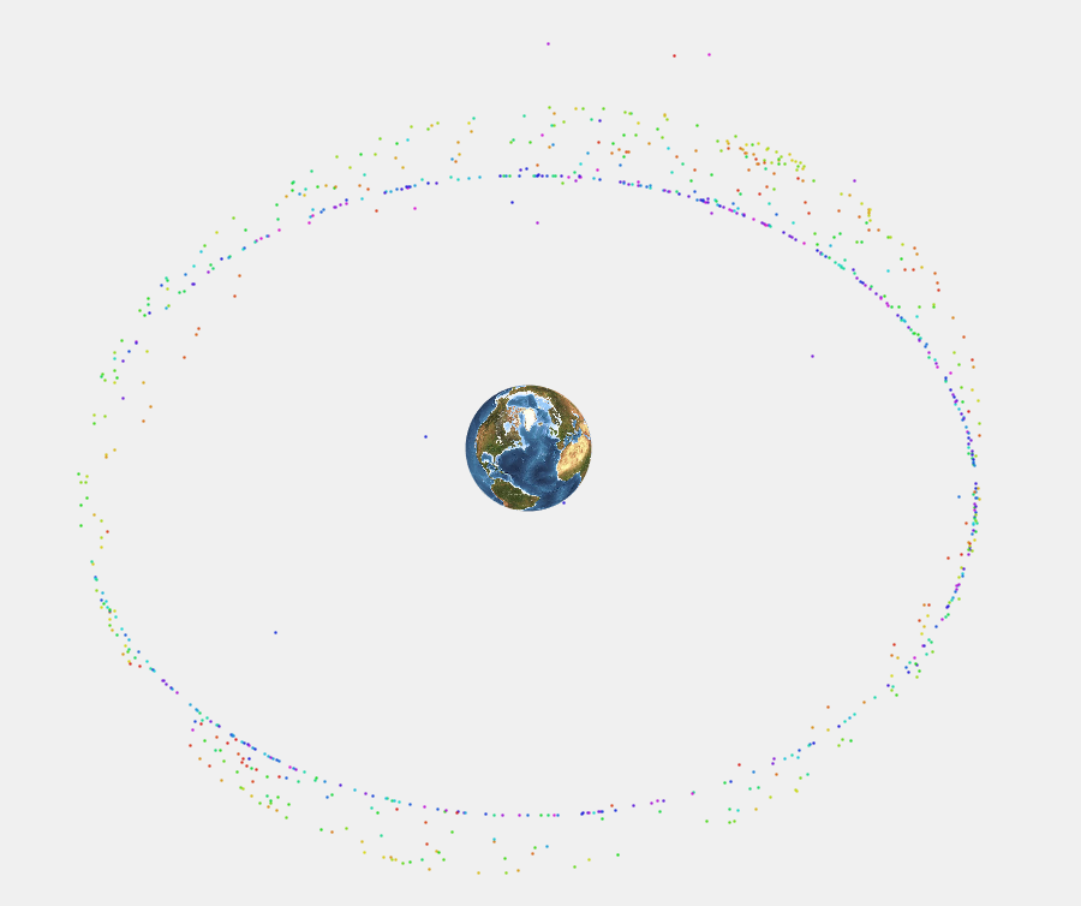


Figure 4.5: MATLAB Visualisation of Geosynchronous Earth Orbit Catalogue



4.1.2 Particle Generation

The following visualisations show snapshots of various TLE catalogues visualised with particle clouds of size 100. It is important to note a higher particle count is often required to accurately represent error for analysis purposes, but for visualisation purposes, 100 is sufficient. Unless stated otherwise, all visualisations used the most recent possible TLE set at the time of visualisation, sourced from Space-Track.

Each particle cloud for each object is coloured the same, although due to the large number of objects, colours had to be re-used.

Note that despite using the most recent TLE sets (less than a day old) the particle clouds had substantial along-track uncertainty (on average ±250km for Low Earth Orbit).

Table 4.1: Average along track uncertainty

Catalogue	Uncertainty
LEO	$\pm 250km$
MEO	$\pm 1200km$
HEO	$\pm 1900km$
GEO	$\pm 2400km$

Figure 4.6: MATLAB Visualisation of Low Earth Orbit Catalogue with 100 particles/object

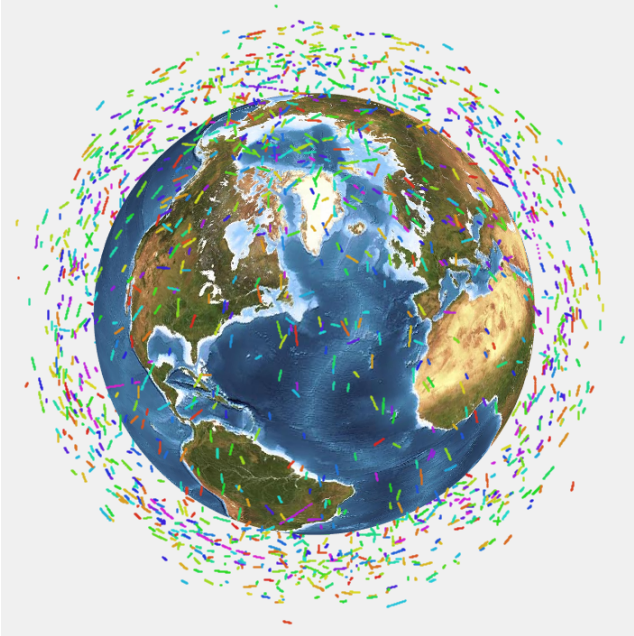


Figure 4.7: MATLAB Visualisation of Medium Earth Orbit Catalogue with 100 particles/object

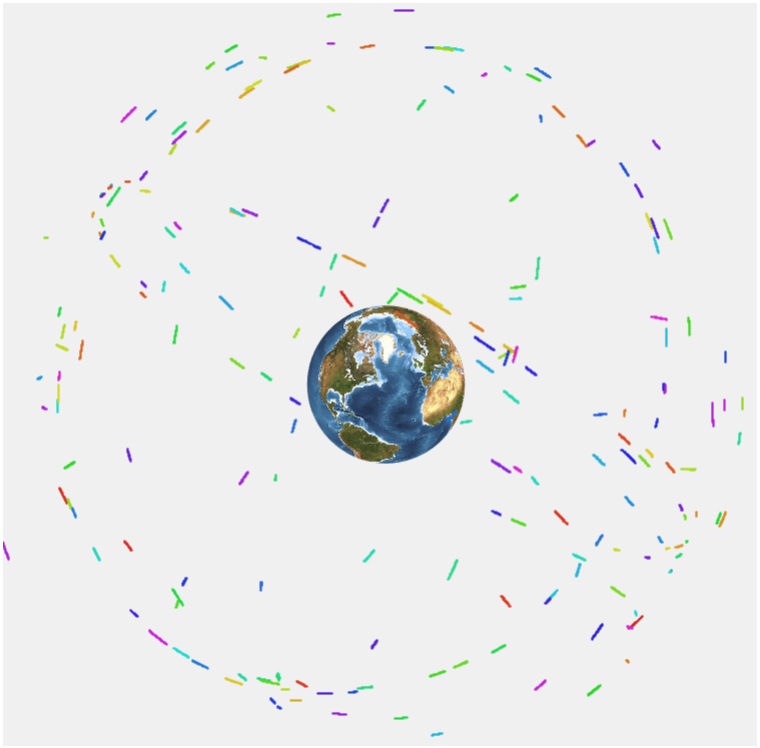


Figure 4.8: MATLAB Visualisation of Highly Elliptical Orbit Catalogue with 100 particles/object

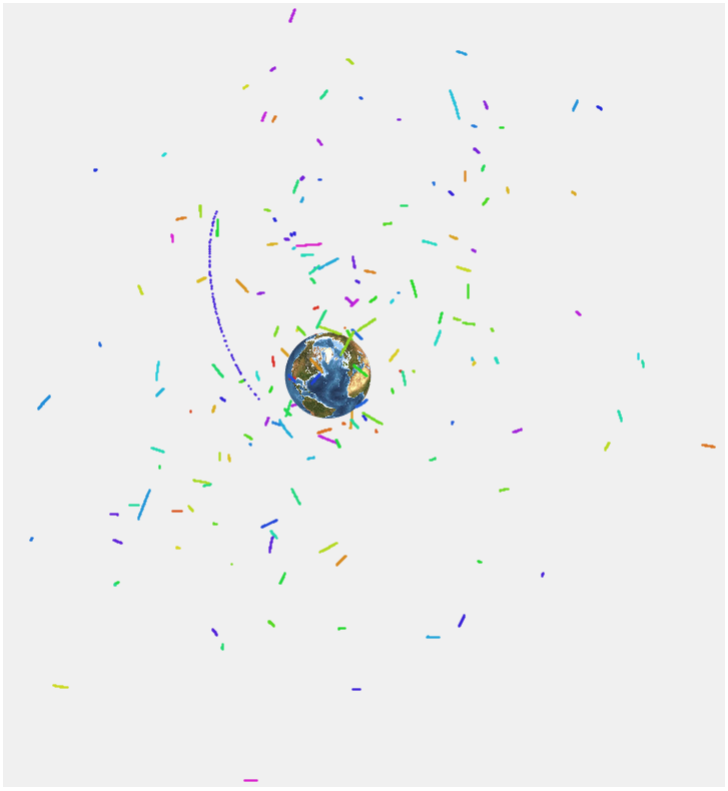


Figure 4.9: MATLAB Visualisation of Geosynchronous Earth Orbit Catalogue with 100 particles/object

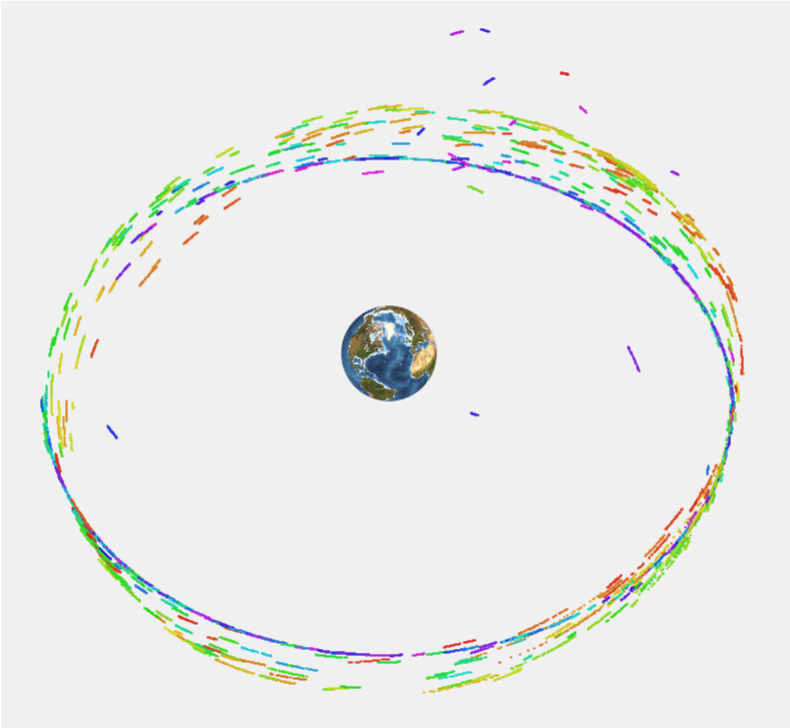
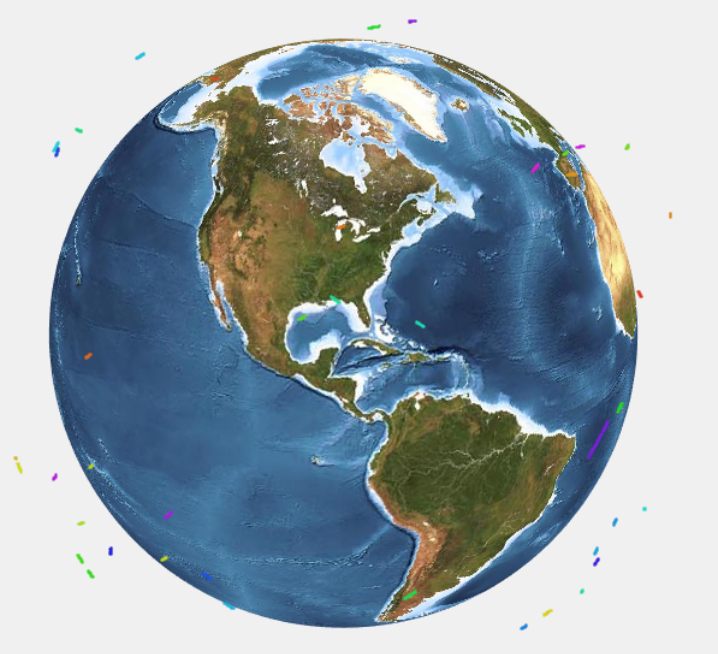


Figure 4.10: MATLAB Visualisation of Orbital Communications Catalogue with 100 particles/object



4.2 Unity Visualisation

The following figures show Unity visualisations of various catalogues.

Figure 4.11: Unity Visualisation of Low Earth Orbit Catalogue (verifcation)

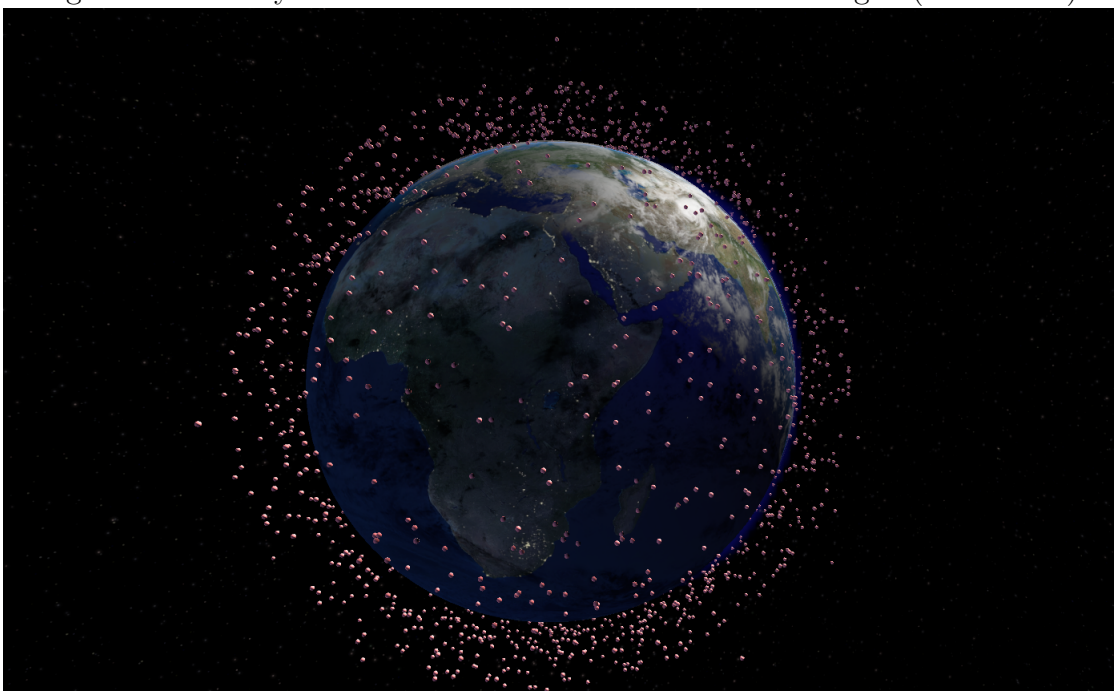


Figure 4.12: Unity Visualisation of two week old TLE set with 400 particles/object, showing TLE accuracy decay

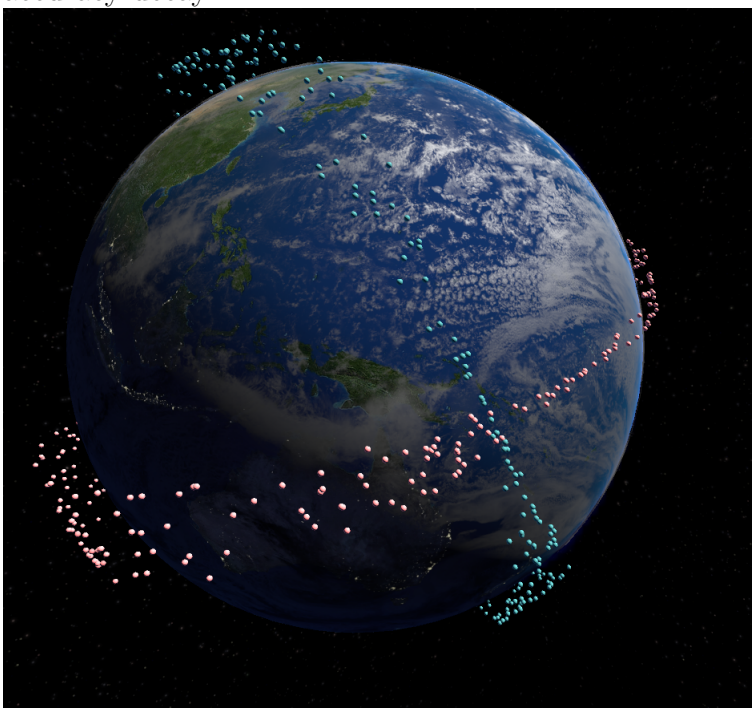


Figure 4.13: Unity Visualisation of Highly Elliptical Orbit Catalogue with 25 particles/object

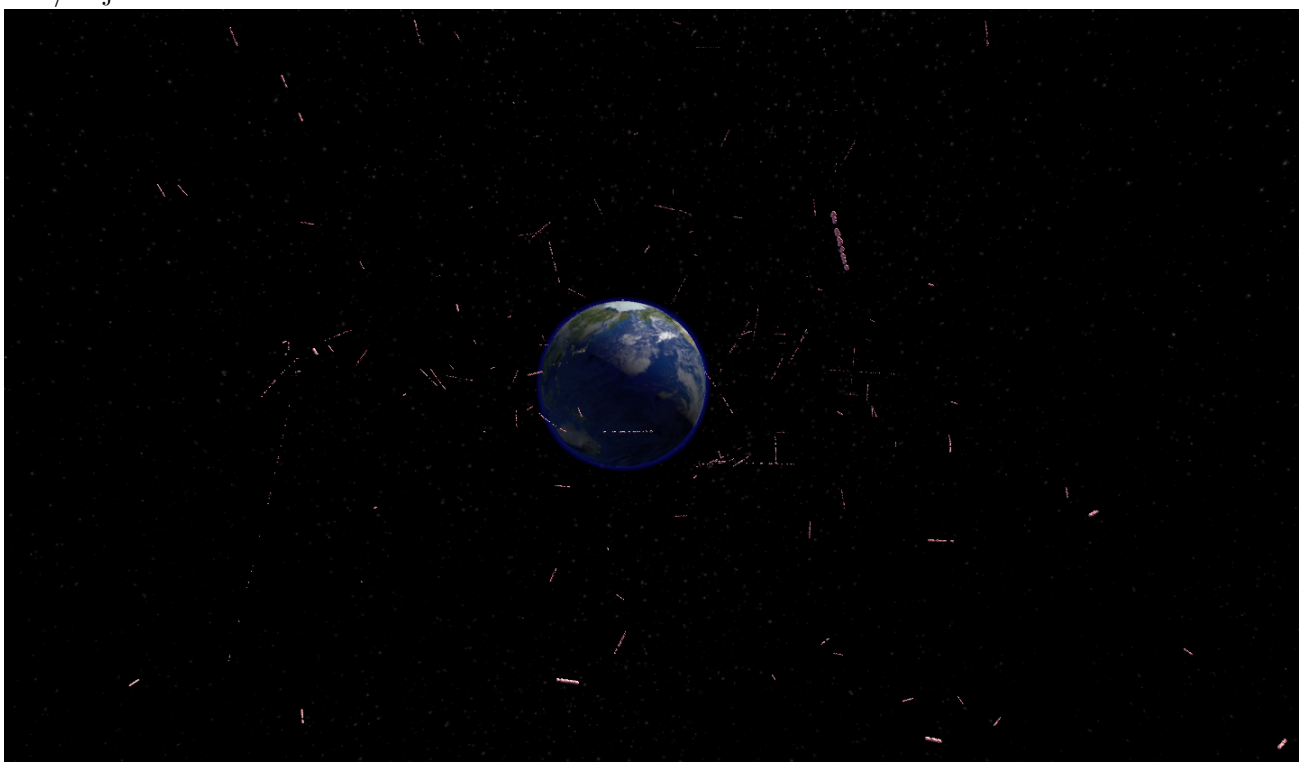


Figure 4.14: Unity Visualisation of Medium Earth Orbit Catalogue with 25 particles/object

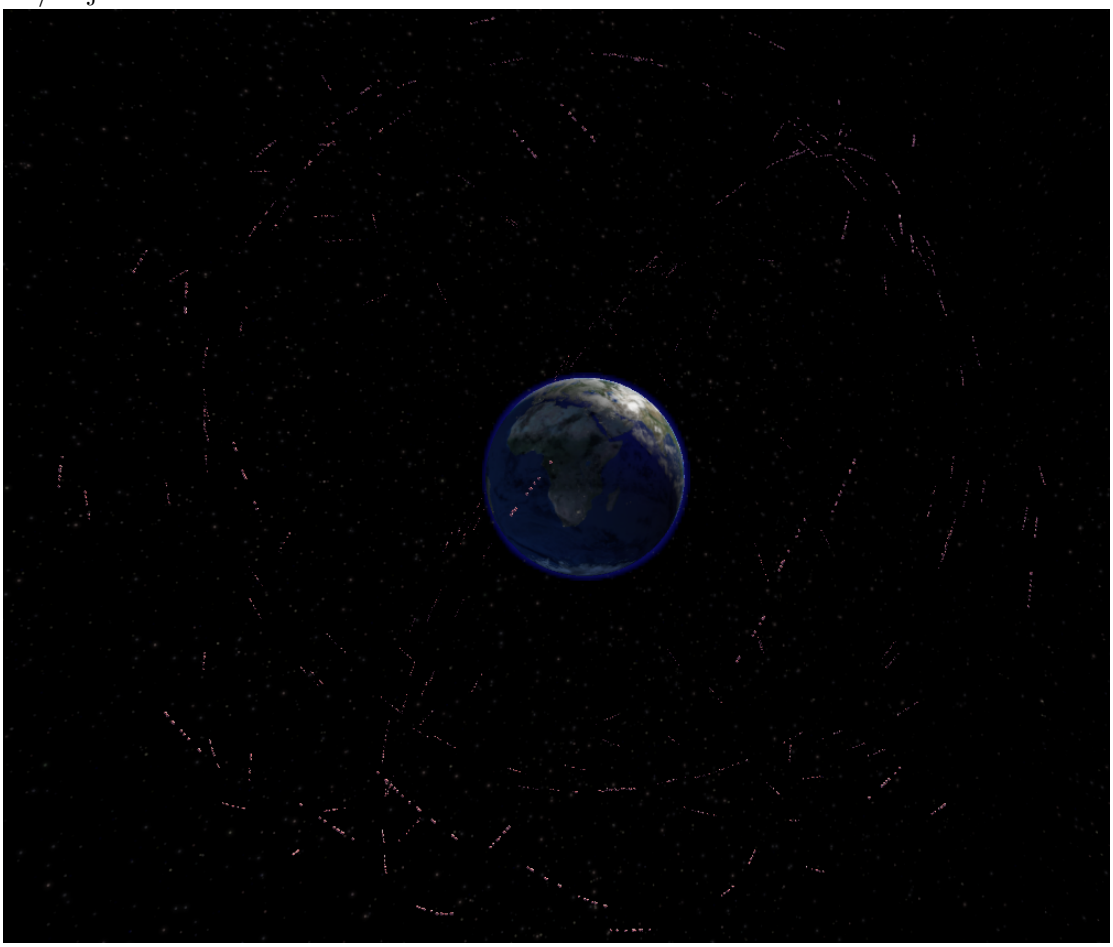
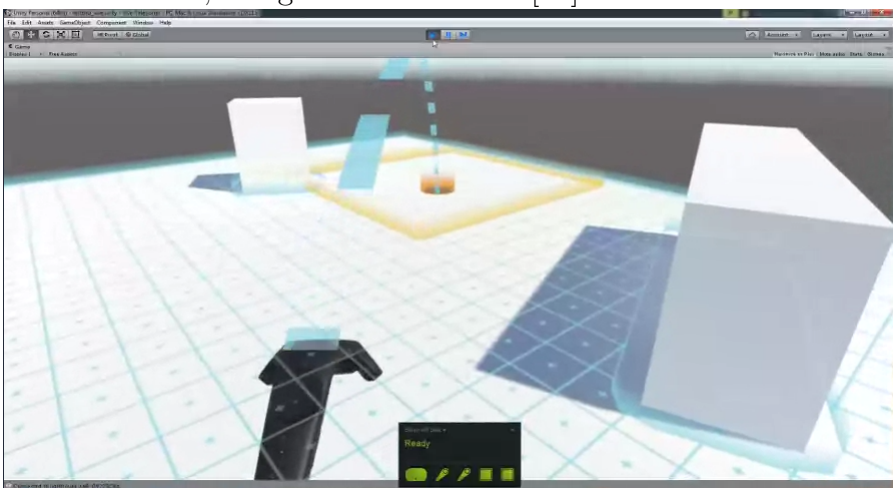


Figure 4.15 shows the teleportation movement indicator, demonstrating where the user will teleport to when they release the trigger input.

Figure 4.15: Unity Teleportation shown in text environment, allowing movement outside of room bounds, using framework from [14]



4.3 GPU Parallelism Performance

After parallelising the propagation code, the computation time for the parallel implementation was compared with the serial implementation for various numbers of objects and frames to compute. The results are shown below.

Table 4.2: Average computation time of serial and parallel implementation for 1 frame for varying numbers of objects

	Average Run Time	
Num. Objects	Serial	Parallel
15500	102s	13.1s
2200	4.30s	1.70s
333	0.69s	0.29s
24	0.13s	0.05s

Table 4.3: Average computation time of serial and parallel implementation for 100 frames for varying numbers of objects

	Average Run Time	
Num. Objects	Serial	Parallel
15500	145s	16.0s
2200	32.0s	1.70s
333	0.96s	0.30s
24	0.15s	0.05s

Table 4.4: Average computation time of serial and parallel implementation for 1000 frames for varying numbers of objects

	Average Run Time	
Num. Objects	Serial	Parallel
15500	1422s	N/A (Out of VRAM)
2200	176s	3.00s
333	26.8s	0.41s
24	1.99s	0.06s

Figure 4.16: Average Compute Time vs Number of Objects for 100 Frames

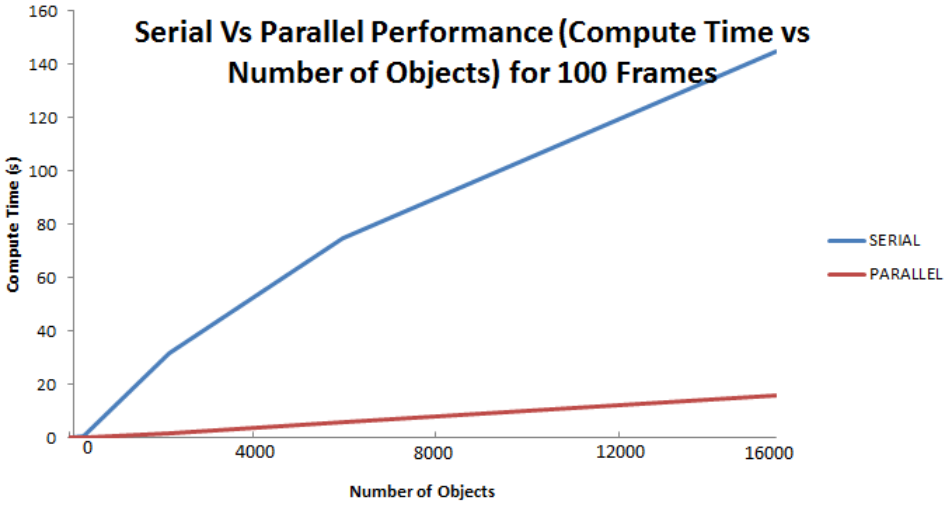


Figure 4.17: Speedup Factor from Parallel to Serial Implementation for 100 frames

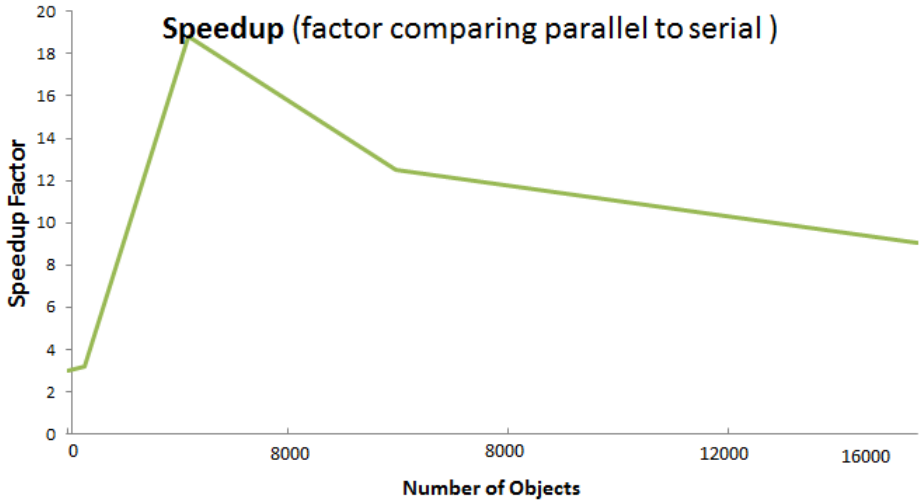


Figure 4.18: Average Compute Time vs Number of Objects for 1000 Frames

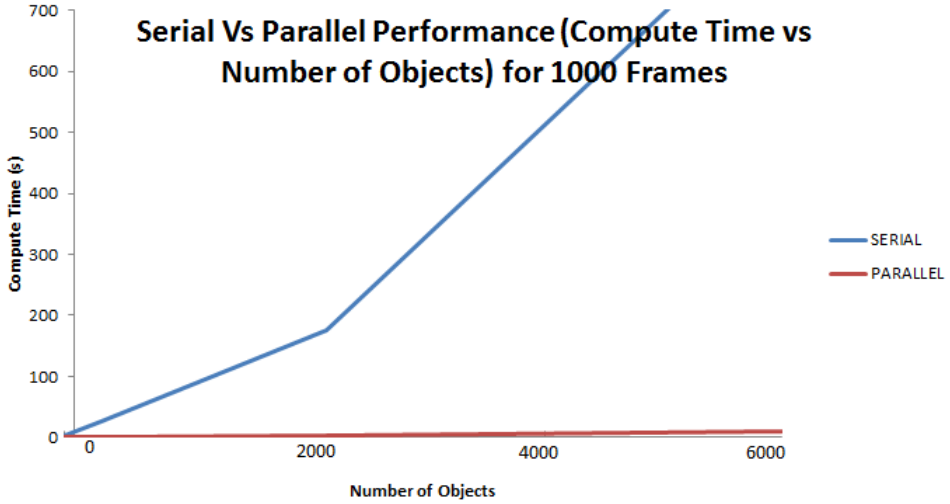
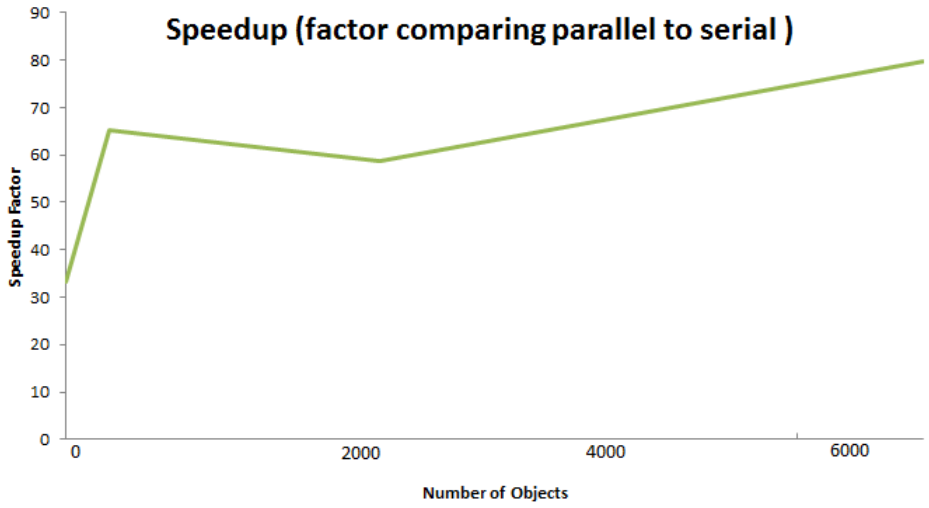


Figure 4.19: Speedup Factor from Parallel to Serial Implementation for 1000 frames



4.4 Close Approach Analyser

To test the close approach analyser, the analysis tool was run using a TLE set with a known collision. The chosen collision was the Iridium 33 and Kosmos 2251 collision on February 10, 2009, at 16:56 UTC, with a TLE containing those two satellites with an epoch at 16:50 UTC.

This TLE is then run through the MATLAB Simulation with 20,000 particles, until the collision is visually confirmed. The simulation can then exit, and the output file created will contain all of the positions of all 20,000 particles for both satellites, at all timesteps (from 16:50 UTC to approx 16:57 UTC in 0.1 minute timesteps).

This output file is then input into the analysis tool which will give the probability of a close approach (which is the fraction of all particles that experienced a close approach).

This analysis was performed at varying close approach thresholds, and the results can be seen below in Table 4.5.

Table 4.5: Probability of close approach for varying threshold

Threshold	Probability
15km	0.15
20km	0.512
25km	0.831

Chapter 5

Discussion

5.1 Particles

The most significant result of the investigation with respect to the use of particles instead of Gaussian volumes, was the significant along track uncertainty observed, even close to epoch. The comparatively minimal across track uncertainty and curvature of the observed particle cloud further supports the theory of significant discrepancies between the true distribution of error and that represented with an ellipsoid volume. This supports the use of particle representations of uncertainty over Gaussian volumes with TLE analysis.

Further investigation directly comparing the impact of inaccuracy in Gaussian error volumes with Monte-Carlo particle representations in the context of low fidelity threat detection such as those described in this document is needed to quantify this relationship.

5.2 Parallelisation

Due to the independent nature of each object within a large set of objects, the problem was highly parallelisable, with speedups of up to 80 times observed, using a low-end graphics card. This significant speedup was a result of unnecessarily sequential operations being parallelised.

Use of a graphics card with more VRAM would allow larger problems to be simulated and also allow greater parallelisation, assuming transfer speeds to and from the GPU did not quickly become a bottleneck for performance.

This intrinsic affinity for parallelisation supports the effectiveness of supercomput-

er/cluster approaches to simulating large catalogues of objects with massive numbers of particles.

5.3 UDP Bottleneck

Communications between MATLAB and Unity over UDP was a significant bottleneck due to the large amounts of data being sent. If all objects were sent as an individual message, performance on the receiving end in Unity was poor. Conversely, sending all objects as one message improved performance, but imposed a cap on the number of objects that could be sent, due to the native implementation of UDP communications in MATLAB having an output buffer size of 8192 bytes. Fortunately, the number of objects required to hit this cap was similar to the number of objects required to hit a performance cap with significant slowdowns.

Thus, if the Unity visualisation was to be optimised further, the following areas would need to be addressed:

- MATLAB output buffer size problem. This could be bypassed by sending chunked data when this output buffer size is reached, or by writing a new UDP implementation with a larger output buffer.
- Further optimisation of object location updating for large numbers of objects. This might be achieved by using less position updates and interpolating between points. This would also lighten the load on the UDP bottleneck.
- Threading or GPU parallelisation of the aforementioned object location updating would contribute to a significant speedup, as the visualisation was also bottlenecked by CPU performance for large numbers of objects.

One of the concessions made to maximise performance was the use of 16 bit integers, which meant upper orbits such as geosynchronous orbits were not able to be sent to Unity, as the positions required greater width than 16 bit integer representations supported. However, the performance trade-off was that the throughput was effectively doubled.

5.4 Suitability of Virtual Reality to Visualisation

Whilst a wildy different medium to the norm, the use of Virtual Reality was extremely effective at conveying scale, depth, and angle, which can be difficult to

visualise with conventional visualisations, due in part to the characteristic small field of view of a scientific visualisation.

With additional environmental interactivity such as a point-and-shoot selection, the use of VR would have further utility, especially as a presentation tool. Compared to a conventional simulations or videos, an interactive environment may offer far more depth.

Further study is needed to establish the anecdotal benefits to user engagement and learning offered by Virtual Reality seen in this study. This would include quantitative investigation of the aforementioned metrics.

5.5 Close Approach Analysis

Looking at the results for the Iridium 33 / Kosmos 2251 collision, the fidelity of the TLE set was surprisingly low. With a close approach threshold of 15km and a particle count of 20,000, a known collision only had a probability of close approach of 0.15. Stepping this threshold to 25, gave a probability of 0.831 for a close approach.

Given the reported accuracy for average TLE sets being considerably higher (with high quality sets having $\pm 1km$ error at epoch, and average quality sets having approximately $\pm 30km$ three days after epoch [15]), this result was surprising, although several sources discuss how TLE sets can have wildly varying accuracy [16]. It is also possible that this project's implementation of SGP4 propagation is contributing to a loss in fidelity.
Whilst the fidelity of the TLE limits the ability to reliably predict collisions or approaches below 15km, the fidelity afforded does allow the prediction of close approaches with a larger close approach threshold (such as 25 or 30km). Thus, the method discussed in this project could be used as a initial processing step to identify pairings of interest, with the results being further validated by a higher fidelity model before a SSA recommendation is offered.

Performance needs to be optimised significantly to be useful -¿ perform full catalog analysis with large particle number -¿ supercomputer cluster parallelises well

5.6 Critical Performance Review

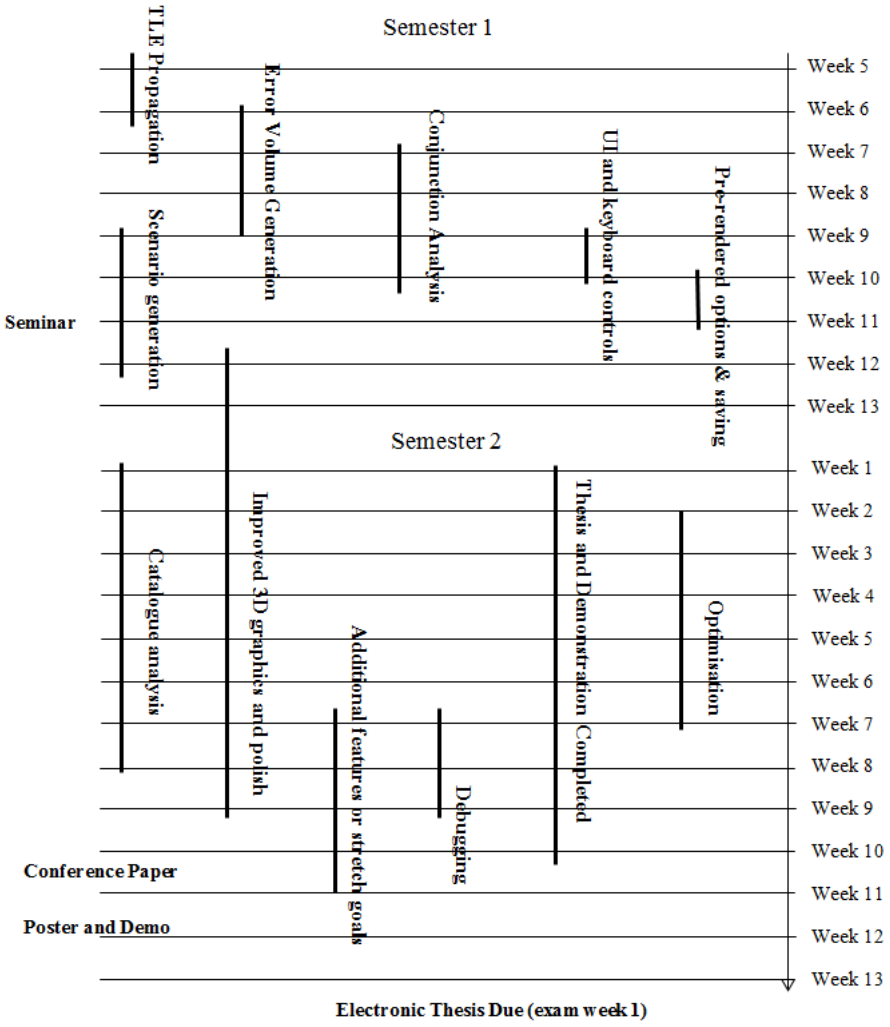
Relating to the goals set for the project, this project performed well at achieving the goals on paper. Where the project performed was the disjointed connection of the

individual goals in their final state. Ideally, the various components of this thesis would have meshed well as a whole, but feasibility and time related complications hindered this area greatly.

Foremost of these problems was the lack of interactivity with objects in the Unity VR simulation. This was one of the stretch goals intended for this project: being able to select a pair or group of objects in the visualisation and calculate a probability of a close approach. Due to performance problems and a desire to have support sizable simulations, this stretch goal was deemed less important. In retrospect, this may have been to the detriment of the project taken as a whole.
Comparing the actual development of the project against the initial development schedule, set near the beginning of the project and seen overleaf in Figure 5.1, the project was mostly a success. The first semester goals were universally met, but the loftier second semester goals were more difficult to achieve, in part because of assumptions regarding the feasibility of certain tasks such as full catalogue analysis and the stretch goals.

Otherwise, all of the major deadlines were met, with some of the minor deadlines being completed late. Primarily the complications with the development schedule stemmed from discrepancies between the initial estimate of a task's difficulty and the actual difficulty of a task. Further time related complications included grouping of the author's other assessment with certain tasks in the schedule.

Figure 5.1: Development Schedule from Research Proposal in April



5.7 Possible future work

There are many areas of possible future work related to this project. Future work directly relating to the implementations in this project would include:

- Propagation on a GPU with more memory, allowing for larger real time visualisations
- Further optimisations to GPU implementation
- UDP implementation with larger buffer size
- Parallelise and optimise Unity visualisation to support larger simulations.
- Integration of close approach analysis with Unity

- Domain decomposition pre-filtering methods to optimise close approach analysis
- Further interaction options in Unity visualisation such as selecting objects to retrieve data or performing analysis on selected object.

Areas of further research include: improvements to Two-line element fidelity such as correction methods, further investigation into the utility of Virtual Reality as an investigation/presentation medium (this would include quantifying engagement or learning metrics) and directly comparing the impact of inaccuracy in Gaussian error volumes with Monte-Carlo particle representations in the context of low fidelity threat detection.

Chapter 6

Conclusions

6.1 Summary and conclusions

This project created two visualisation tools and a companion close approach analysis tool using the common TLE/SGP4 approach. It investigated the use of generative Monte-Carlo particles methods and found them to be a useful alternative to the typical Gaussian error volume, particularly when TLE data has aged.

This investigation further highlights the quick degradation of TLE sets and the need for TLE correction methods, a current area of study (see [18, 19]). This investigation found the fidelity afforded by TLE is insufficient to reliably detect close approaches within a threshold of 15km, even close to epoch. A threshold of 25-30km was found to be significantly more reliable.

Finally, this project found Virtual Reality to be an effective and engaging presentation tool, conferring depth and scale far greater than is possible in other visualisations. Further investigation is needed to quantify these benefits.

Appendix A

Code Snippets

A.1 TLE File Handling

Figure A.1: TLE File Handling Code Snippet

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Read in input file
[FileName,PathName,FilterIndex] = uigetfile('*.txt');
if FileName ~= 0
    filePath = strcat(PathName,FileName);
    fid = fopen(filePath);
    tline = fgetl(fid);
    f = 1;
    %If first char is not number -> named TLE file
    if ~isstrprop(tline(1), 'digit')
        %Strip every first line in 3
        f = 2/3;
    end
    %Get number of rows in file
    rows = numel(textread(filePath,'%1c%*[^\n]'));
    Nrows = rows*f;

    %Init c - cell array storing TLE elements
    c = cell([Nrows, 1]);
    count = 1;
    while ischar(tline)
        %%if first char not number
        if isstrprop(tline(1), 'digit')
            c(floor(count*f),1) = cellstr(tline);
        end
        count = count + 1;
        tline = fgetl(fid);
    end
else
    fprintf('User cancelled file select.\n');
    return
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure A.2: Particle Generation Code Snippet

```
%Init gpu arrays
satnum = zeros(1, objs);
jdsatepoch = zeros(1, objs);
tsepoch = zeros(1, objs);
bstar = zeros(1, objs);
ecco = zeros(1, objs);
argpo = zeros(1, objs);
inclo = zeros(1, objs);
no = zeros(1, objs);
mo = zeros(1, objs);
nodeo = zeros(1, objs);

for i = 1:2:Nrows
    %Process input strings into two lines
    line1 = c{i,1};
    line2 = c{i+1,1};
    j = floor(i/2);

    %return satrec for each object in TLE
    [satrec] = tleToSatrec0(7210, line1, line2, 'm', 'j', startmfe, stopmfe, deltamin);

    %Populate GPUArrays with satrec struct values
    for x = 1:(partNum)

        satnum(partNum*j+x) = satrec.satnum;
        tsepoch(partNum*j+x) = julian_time - satrec.jdsatepoch;
        jdsatepoch(partNum*j+x) = satrec.jdsatepoch;
        bstar(partNum*j+x) = satrec.bstar;
        ecco(partNum*j+x) = satrec.ecco + rEcco*(rand*2-1);
        argpo(partNum*j+x) = satrec.argpo + rArgpo*(rand*2-1);
        inclo(partNum*j+x) = satrec.inclo + rInclo*(rand*2-1);
        mo(partNum*j+x) = satrec.mo + rMo*(rand*2-1);
        no(partNum*j+x) = satrec.no + rNo*(rand*2-1);
        nodeo(partNum*j+x) = satrec.nodeo + rNodeo*(rand*2-1);
    end
end
```

Figure A.3: MATLAB GPU Code Snippet

```
%Send data to GPU -> the follwing arrays have been initialised above

%The results array
d_Result = gpuArray(res);

%The contents of each satrec stored in corresponding arrays
d_satnum = gpuArray(satnum);
d_jdsatepoch = gpuArray(jdsatepoch);
d_bstar = gpuArray(bstar);
d_ecco = gpuArray(ecco);
d_argpo = gpuArray(argpo);
d_inclo = gpuArray(inclo);
d_mo = gpuArray(mo);
d_no = gpuArray(no);
d_nodeo = gpuArray(nodeo);
d_tsepoch = gpuArray(tsepoch);

fprintf('... ');

%Initialise GPU Kernel
kernel = parallel.gpu.CUDAKernel('Prop_demo.ptx' , 'Prop_demo.cu' );
kernel.ThreadBlockSize = [sqrt(kernel.MaxThreadsPerBlock) , sqrt(kernel.MaxThreadsPerBlock),1];
kernel.GridSize = [ceil( cols / kernel.ThreadBlockSize(1)) , ceil( rows / kernel.ThreadBlockSize(2) )];

fprintf('... ');

%Run Parallel Code starting at time 0
t = 0;
d_Result = feval(kernel, d_Result, rows, cols, t, deltamin, d_jdsatepoch, d_bstar, d_ecco, d_argpo, d_inclo,
```

Figure A.4: nVidia CUDA Code Snippet

```
//Compile method within MATLAB -> | nvcc -ptx -arch=sm_21 Prop_demo.cu'

#include "sgp4unit.cu"

//This function
__global__ void PartPd_Kernel(double* ResultOut, int rows, int cols, int t,
        double del, double* jdsatepoch, double* bstar,
        double* ecco, double* argpo, double* inclo, double* mo, double* no,
        double* nodeo, double* tse)
    {
    //Determine Thread Position in Grid
    double rowD = blockIdx.y * blockDim.y + threadIdx.y;
    double colD = blockIdx.x * blockDim.x + threadIdx.x;
    int row = (int) rowD;
    int col = (int) colD;
    gravconsttype whichconst = wgs72old;
    elsetrec satrec;



    //Ensure Excess Threads are not Evaluated
    if ((row < rows)&&(col < cols)){
        sgp4init(whichconst, 'i',  1,  jdsatepoch[col], bstar[col],
                  ecco[col],  argpo[col],  inclo[col],  mo[col],  no[col],
                  nodeo[col], satrec);
        double time = (tse[col]*1440) + ((t + rowD)*del);
        double ro[3];
        double vo[3];

        sgp4(whichconst, satrec, time, ro, vo);

        ResultOut[row + rows*(col+cols*0)] = ro[0];
        ResultOut[row + rows*(col+cols*1)] = ro[1];
        ResultOut[row + rows*(col+cols*2)] = ro[2];
    }

}
```

Figure A.5: Unity UDP Decoding Code Snippet

```
// receive thread
private void ReceiveData()
{
    client = new UdpClient(port);
    client.Client.ReceiveBufferSize = 65536;

    while (exitFlag == false)
    {
        try
        {
            IPEndPoint anyIP = new IPEndPoint(IPAddress.Any, 0);
            byte[] data = client.Receive(ref anyIP);

            //If init flag set up object array with recieved number of objects
            if (initFlag) {
                num = BitConverter.ToInt32(data, 0);
                gameObjArr = new Transform[num];
                pastPosArr = new Vector3[num];
                print("num is" + num);
            } else {
                int p;
                for (p=0; p < num; p++)
                {
                    int[] currObj = new int[4];
                    int position = p;
                    currObj[0] = position;

                    currObj[1] = BitConverter.ToInt16(data, (p*2));
                    currObj[2] = BitConverter.ToInt16(data, num * 2+(p*2));
                    currObj[3] = BitConverter.ToInt16(data,  num * 4 + (p * 2));

                    if (position > 0 && position <= num)
                    {
                        if (currObj.Length != 0)
                        {
                            queue.Enqueue(currObj);
                        } else
                        {
                            print("Outputting temp  " + currObj[0] + " " + currObj[1] + " " + currObj[2]
                        }
                    }
```

Appendix B

Companion Files

Companion files are provided at the following location (hosted on Dropbox):

`https://www.dropbox.com/sh/8jtsg8todazhc81/AACrgWpOaQxdjOygA2Lj2fYLa`

This includes videos of both the MATLAB and Unity Visualisations, all the source MATLAB and Java code and the modified Vallado SGP4 code.

The MATLAB code is compatible with MATLAB R2015b.

Bibliography

[1] Allianz Global Corporate and Speciality, Space Risks: A new generation of challenge, (2012)

[2] U.S. Congress, Office of Technology Assessment, Orbiting Debris: A Space Environmental Problem-Background Paper, (September 1990), OTA-BP-ISC-72, Washington, DC: U.S. Government Printing Office

[3] Space Environment Research Centre, SERC Partners, (2014). Retrieved April 1, 2016, URL: http://www.serc.org.au/research/program-3/

[4] T.S. Kelso, Analysis of the Iridium 33-Cosmos 2251 Collision, Centre for Space Standards and Innovation, (2009)

[5] F. R. Hoots, SpaceTrack Report No.3, Department of Defense, 1980, Retrieved October 1, 2016, URL:https://celestrak.com/NORAD/documentation/spacetrk.pdf

[6] Science Applications International Corporation, SpaceTrack, Retrieved October 15, 2016, URL: https://www.space-track.org/

[7] D. A. Vallado, Revisiting Spacetrack Report No.3, Centre for Space Standards and Innovation, Colorado Springs, 2006, Retrieved October 1, 2016, URL: https://celestrak.com/publications/AIAA/2006-6753/AIAA-2006-6753.pdf

[8] Hobson, T., Sensor Management for Enhanced Catalogue Management of Resident Space Objects, 2014, University of Queensland, p146.

[9] Kim Dismukes, Two-line Element Definition, 2011, NASA, Retrieved October 1, 2016, URL: http://spaceflight.nasa.gov/realdata/sightings/SSapplications/Post/JavaSSOP/SSOP_Help/tle_def.html

[10] R.W. Ghrist ,D. Plakalovic, Impact of Non-Gaussian Error Volumes on Conjunction Assessment Risk Analysis, (2012), a.i. Solutions, Inc., Colorado Springs, CO 80915

[11] Y Jianjun, Z Jianqiu, Z Zesen, Gaussian Sum PHD Filtering Algorithm for Nonlinear Non-Gaussian Model, (2008), Department of Electronic Engineering, Fudan University, Shanghai 200433, China

[12] G. Terejanu, P. Singlay, T. Singhz, P.D. Scottx, Uncertainty Propagation for Nonlinear Dynamical Systems using Gaussian Mixture Models, (2008), American Institute of Aeronautics and Astronautics

[13] A. A. Surez, "Virtual Reality: A tool for treating phobias of heights", 2013, University of Turabo, Eleventh LACCEI Latin American and Caribbean Conference for Engineering and Technology

[14] A. Biagioli, HTC Vive Teleportation System with Parabolic Pointer, 2016, Retrieved October 1, 2016, URL: `https://github.com/Flafla2/Vive-Teleporter`

[15] AGI, Conjunction Analysis Toolkit, 2016, `http://www.agi.com/resources/help/online/stk/11.0/Content/cat/Cat02-04.htm`

[16] D. L. Oltrogge, J. Ramrath, Parametric Characterization of SGP4 Theory and TLE Positional Accuracy, AGI, 2014, Retrieved October 5, 2016, URL: `https://www.agi.com/downloads/resources/white-papers/20140911_EGP_Fit_Capability_AMOS_3317135_v04.docx.pdf`

[17] J. Mason, Development of a MATLAB/STK TLE Accuracy Assessment Tool, International Space University, 2009, Retrieved October 15, 2016, URL: `https://arxiv.org/ftp/arxiv/papers/1304/1304.0842.pdf`

[18] J. C. Bennett, Improving low-Earth orbit predictions using two-line element data with bias correction, The Satellite Positioning for Atmosphere, Climate and Environment (SPACE) Research Centre, School of Mathematical and Geospatial Sciences, 2012, Retrieved October 15, 2016, URL: `http://www.amostech.com/technicalpapers/2012/astrodynamics/bennett.pdf`

[19] W. Lui. ,R. Wang ,R. Yan ,J. Gong, Improving LEO prediction precision with TLEs, Center for Space Science and Applied Research, 2013, Retrieved October 15, 2016, URL: `http://aero.tamu.edu/sites/default/files/faculty/alfriend/CTI2P/CT3%20S2.3%20Liu.pdf`