A SOFTWARE PACKAGE TO SUPPORT MISSION ANALYSIS AND ORBITAL MECHANICS CALCULATIONS

Jorge Tiago Melo Barbosa da Silva e Vila

Instituto Superior Técnico, Av. Rovisco Pais, 1, 1049-001 Lisboa, Portugal, Email: jorge.vila@tecnico.ulisboa.pt

ABSTRACT

The objective of this work was to develop a free and open-source software to support the tasks of space mission analysis and orbital mechanics calculations. It is comprised of two modules – a trajectory propagator and a three-dimensional Solar System simulator. The software was developed for Windows 7, and programmed in C++11. The distribution is made through a website, under the EUPL v.1.1.

The propagator uses the initial conditions provided by the user either through the graphical interface, or XML input files, and propagates the trajectory aided by a Runge-Kutta-Fehlberg 7(8) numerical integrator. The propagator uses a list of force systems, based on several central bodies, and selects the most adequate during the computation. The force model includes the central body's gravity field and the perturbation introduced by its J_2 coefficient, third bodies' gravity fields, and the solar radiation pressure. The propagator generates plain-text and NASA SPK output files.

The simulator is a three-dimensional rendering of the Solar system and allows the user to visualize the trajectory and attitude of celestial bodies and spacecraft. The 3D models of the Sun and planets, with day and night textures, atmosphere, clouds, and planetary rings, are procedurally generated. It is also possible to load 3D models for more complex geometries. A manual camera controlled by the keyboard and mouse was implemented, along with a system of automatic cameras parametrized by the user. The simulator is also capable of exporting movies.

1. INTRODUCTION

The simulation and analysis of trajectories is required throughout the entire lifecycle of every space mission. It requires precise ephemerides, reliable physical models, and robust propagation algorithms. Computers have aided this process since the beginning of the space era and their increasing capabilities now allow ordinary household computers to be used for space mission analysis and design. The birth of three-dimensional computer graphics and its ensuing staggering improvement, largely driven by the videogame industry, contributed to the creation of brand new visualization tools for all areas of science. Combining accurate trajectory propagation and visualization capabilities within the same software suite provides mission designers a fully-featured solution, greatly supporting their work.

The objective of this work was to develop a free and open-source (FOSS) software package, comprising a trajectory propagator module for data generation and a three-dimensional simulator module for enhanced data analysis The trajectory propagator module is able to accurately calculate the trajectory of a spacecraft within the Solar System, whereas the simulation module displays it in a three-dimensional visually realistic environment. Despite being integrated, the modules retain the ability to be used on their own and in conjunction with existing industry tools. Additionally, one of the main design goals was the make the software intuitive and usable by someone with the minimal amount of training.

2. SOFTWARE DESIGN

The software was named SimOrbit, since it is, in its essence, an orbital simulator, and SimOrbit is not a registered trademark.

The chosen programming language was C++ as defined in its most recent international standard ISO/IEC 14882:2011 [1]. This choice took into account several factors, including the stability and maturity of the programming language, familiarity with the standard, and availability of third party components.

Several third party components were integrated in the software in order to increase its functionality and reduce the development time. Microsoft Direct3D was the selected graphics application programming interface (API). It is responsible for managing the graphics hardware, and rendering the three-dimensional (3D) environment, and the graphical user interface (GUI). NASA's Navigation and Ancillary Information Facility (NAIF) SPICE Toolkit [2] is the source of spacecraft and planetary constants, ephemerides, and orientation data. The MPFR [3] and the Multiple Precision Integers and Rationals (MPIR) [4] libraries are used for arbitrary precision support and also to decrease the computation time by using CPU-specific optimizations. The Open Asset Import Library (Assimp) [5] powers the 3D model loader which enables the simulator to use a wide variety to 3D model formats.

2.1. Architecture

The software is comprised of a main application, SimOrbit, and two dynamic link libraries (DLL), SimOrbit Library and SPICE C++, which interact as shown in Figure 2.1.

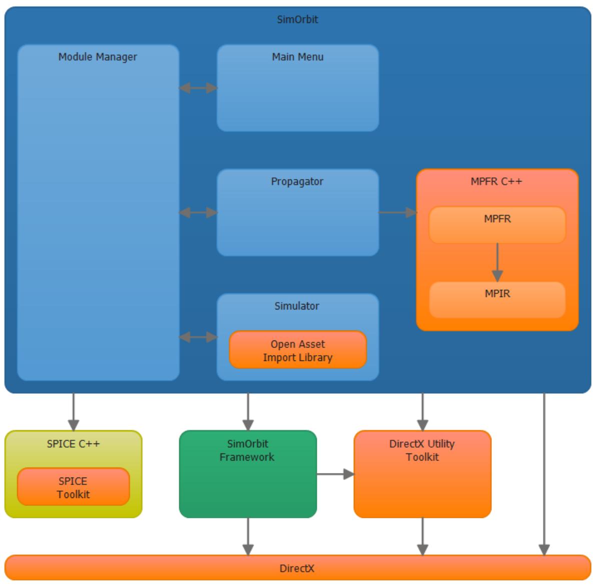


Figure 2.1 – Very high level layer diagram of the software architecture showing the relations between components (third party components displayed in orange).

SimOrbit is a modular application, and includes three modules – Main Menu, Propagator, and Simulator. The SimOrbit Library is part of a framework which harmonizes and simplifies the development of modules. The framework comprises a set of headers and the library itself. The SPICE C++ is a thread-safe C++ wrapper for the SPICE Toolkit which takes advantage of C++ features, such as function overloading, exceptions, vectors and strings to interface with the SPICE Toolkit routines

2.2. Documentation

The C++ code was documented using XML documentation comments. These comments, present in file, class, and method headers detail their purpose, input and output, possible exceptions thrown, and academic source, where applicable. Additional comments are present throughout key algorithms explaining their flow. The XML documentation comments served as input in the creation of Portable Document Format (PDF) and HyperText Markup Language (HTML) reference manuals. Additionally, a user manual was written, describing the general operation of the software. The user manual also includes the definitions used throughout the software, such as frames of reference and units of measurement.

2.3. Licensing and Distribution

Since SimOrbit is a free and open-source software it was necessary to select a license that ensured the original authors retained full ownership, and that any derivative works would be distributed in a similar fashion. The chosen license was the European Union Public Licence (EUPL) v.1.1 [6]. The software and source code will be distributed via a website under the domain “simorbit.eu”. The website also details the software’s features, along with screenshots, and provides a means to contact the authors.

3. PROPAGATOR MODULE

The first implemented component was the trajectory propagator. This component, which has the sole purpose of data generation, is be able to calculate the trajectory of a single spacecraft within the Solar System. Trajectory propagation is done by integrating the equations that describe the motion of the spacecraft, starting from a set initial conditions. This required the definition of a force model, and the implementation of a numerical integrator.

3.1. Force Model

The force model takes into account the fundamental forces that influence the trajectory of a spacecraft travelling through the Solar system [7], and the effect of any engine the spacecraft may be equipped with.

A brief study of space mission scenarios (i.e. use cases) was conducted in order to determine which fundamental forces and engine types needed to be included for accurate propagation of a trajectory. Following the study’s conclusions, the central body gravity field, and its J_2 perturbation, the gravity fields of third bodies, and the solar radiation pressure were the implemented fundamental forces. Additionally, the study also highlighted the necessity of defining a collection of different force systems, each valid within a region of space, and having the propagator switch between them automatically. Furthermore, an engine system was developed which allows the user to use instantaneous delta-V or constant acceleration engine inputs.

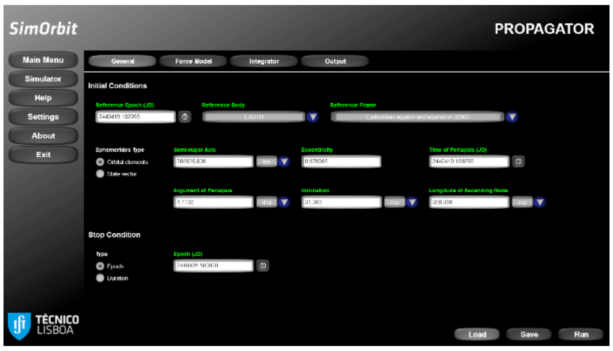
3.2. Numerical Integrator

To propagate the initial state a numerical integrator with adequate precision was required. The chosen numerical method is the Runge-Kutta-Fehlberg 7(8) [3] since it is mature, widely used, fast, and sufficiently accurate for the task. Furthermore it has local truncation error evaluation and self-adjusts the step sized based on it. There are, however, instances where it is desirable to have added control of the step size, so the algorithm was modified to include minimum, maximum, and fixed step size constraints.

The integrator makes use of the MPIR and MPFR libraries to both take advantage of CPU specific instruction sets leading to performance increases but also to give the user a fine-control of the bit-precision. This grants flexibility to the integrator, allowing for its use in both time-critical and precision-critical environments.

3.3. Input and Output

The trajectory propagation requires the user to provide a set of initial conditions, and configure the force model and numerical integrator. To facilitate the task a tabbed GUI where the user can directly input values and set options was implemented, as shown in Figure 3.1.



Initial Conditions

Reference Epoch (JD): 2440419.182095
Reference Body: EARTH
Reference Frame: Earth mean equator and equinox of J2000

Ephemerides Type: [✓ Orbital elements] State vector
Semi-major Axis: 286535.830 km
Eccentricity: 0.978085
Time of Periapsis (JD): 2440419.180255
Argument of Periapsis: 1.4102 deg
Inclination: 31.383 deg
Longitude of Ascending Node: 358.380 deg

Stop Condition

Type: [✓ Epoch] Duration
Epoch (JD): 2440426.500000

Figure 3.1 – Screenshot of the propagator module general tab filled with valid values.

The GUI controls allow the user to input data in several different formats and units of the International System of Units (SI) and the IAU (2009) System of Astronomical Constants, where applicable. This prevents the user from having to perform unit conversions outside of the software, and possibly making a mistake. The propagator GUI validates all input values and flags errors as soon as the affected control loses focus. Hovering the control with the mouse displays a tooltip indicating what the error is.

Since a propagator setup can be modified as the mission definition changes, or may need to be shared between several people working on the project, the setup can be stored and loaded using an Extensible Markup Language (XML) document. Additionally, for Earth satellites it is also possible to load Two-Line Element (TLE) sets that describe the initial conditions.

To ensure maximum compatibility with third party tools, the propagator is able to simultaneously generate two output files – a binary NASA SPK, which is the de facto standard for planetary and spacecraft ephemerides, and a plain-text file.

3.4. Operation

The trajectory propagator is the only multi-threaded component in the SimOrbit project. The worker thread features a controller that manages the output file writers, numerical integrator, and force model, and outputs progress information to the GUI, as shown in Figure 3.2.

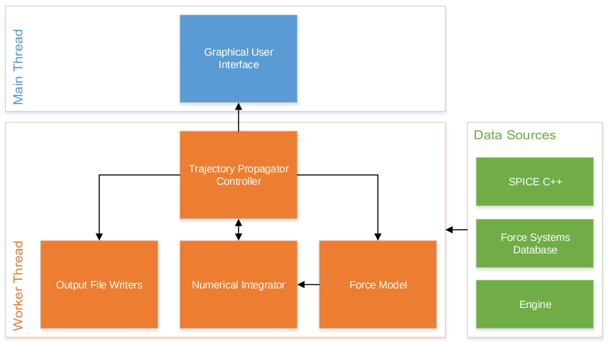


Figure 3.2 – Thread communication and information flow diagram during the operation of the propagator module.

4. SIMULATOR MODULE

The second implemented component, the simulator module, is a three-dimensional realistic rendering of the Solar System. This component's primary purpose is spacecraft trajectory, position, and attitude visualization.

Most mission analysis and design tools that include 3D visualization features only provide very basic renderings of the Solar System bodies and spacecraft of interest. Although this is often enough for a preliminary trajectory analysis, it is insufficient to fully understand the scene. To ensure the success of a space mission it is necessary to know if a spacecraft's solar panels are illuminated, if its instruments and antennas are pointed correctly, if a feature being photographed is illuminated, if a lander is in view, if there are objects in its path that were not taken into account while propagating the trajectory, etc… For all these purposes it is necessary to have a rendering of the 3D environment that strives to match the reality.

The main development goal for the simulator module was to maximize visual realism. In order to achieve this goal, it is able to:

- Generate or load the geometry for simple objects (i.e. ellipsoidal celestial bodies)
- Load 3D models for more complex objects, such as spacecraft, asteroids, and comets;
- Place the models on their correct positions and attitudes;
- Render the scene using physically-based shading techniques.

4.1. Generation and Rendering of Celestial Bodies

To generate the geometry for ellipsoidal celestial bodies a procedural generator was developed. This generator is able to generate the ellipsoidal shape of the body, including features such as atmosphere, clouds, and planetary rings, using a combination of information from the NAIF kernels and an XML configuration file. Figure 4.1 shows planet Earth generated and rendered by SimOrbit.

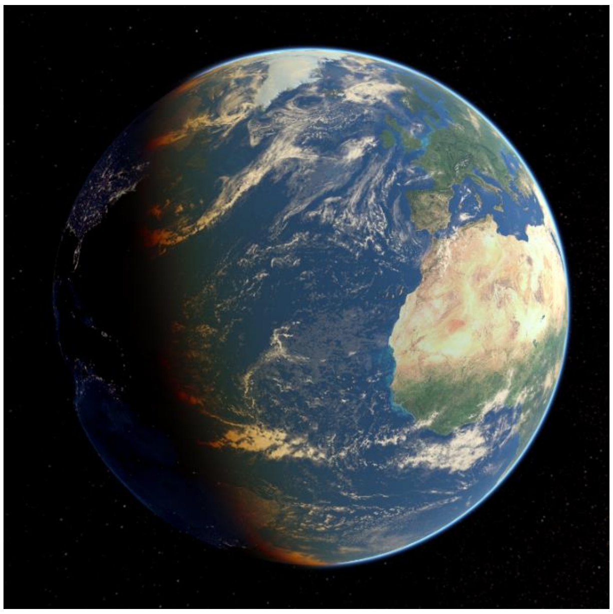


Figure 4.1 – Screenshot of the Earth as rendered by SimOrbit.

The planetary shader uses physically-based shading techniques, which are algorithms that take into account the physical properties of the materials and the lighting in order to render images that resemble reality.

The lighting of a planet is based on the Kelemen-Szirmay-Kalos bidirectional reflectance distribution function (BRDF) model [8]. This model closely approximates the Cook-Torrance model [9] but is significantly cheaper to evaluate. The classic Blinn-Phong model [10] was also tested, but proved too glossy and unrealistic.

The colour of the planet's surface is a combination of diffuse and emissive colours and textures. The diffuse texture stores the daylight representation of the planet's surface, whereas the emissive texture provides the night-time colours, such as city lights or auroras. The user can also provide a specular map, which controls the specular strength (i.e. the reflectivity of the surface), and a tangent space normal map which stores the per-pixel normal in order to give the illusion of terrain height.

When light travels through the atmosphere of a planet it is scattered by its particles, a phenomenon known as atmospheric scattering. It is of an extreme importance towards a photorealistic rendering of a planet. The planetary shader uses the physically based atmospheric scattering algorithm proposed by Sean O'Neal [11] to calculate the Rayleigh and Mie scattering components and render the atmosphere. Figure 4.2 shows Earth and Mars atmospheres as rendered by SimOrbit.

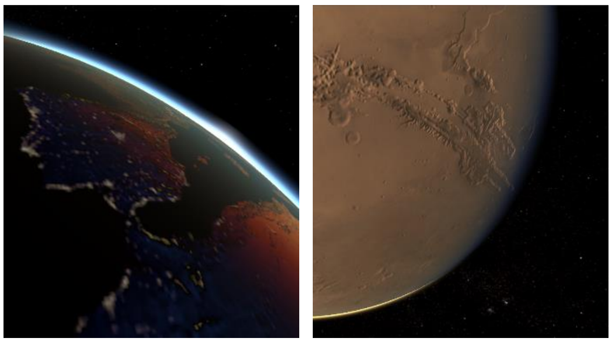


Figure 4.2 – Atmospheric scattering examples. Sunrise over the Iberian Peninsula on Earth (left); sunset over Valles Marineris on Mars (right).

The software also supports rendering clouds. Cloud height information and a texture map containing their transparency can be provided. The shader then combines this information with the atmospheric scattering algorithm and renders photorealistic clouds. However, it does not render the cloud's surface shadow, which was found to be of no consequence.

Finally, the procedural planet generator and shader also support planetary rings. These are a flat dual-sided anullus centred on the planet, laying on its equatorial plane. The inner and outer ring radius are provided by the user, along with a texture map storing both their colour and transparency. The rings' illumination presents a very complex problem, since these are illuminated by backscattered from the planet and the neighbouring ring particles, and forward scattered light. A simple model was devised attempting to match Saturn's rings renderings to the photos from the Voyager and Cassini missions. The model is not physically accurate but provides acceptable results.

4.2. Loading and Rendering of 3D Models

For more complex geometries, such as spacecraft and some asteroids and comets, it is necessary to load 3D models containing information regarding their geometry and materials. Since 3D models come in a variety of formats and it is not trivial to convert between them a model loader supporting most common interchange formats was included. The loader is powered by the Assimp library.

Two ways were implemented for associating a 3D model with an object – static and dynamic. The static associations are listed in an XML file, alternatively the user can also load a 3D model dynamically via the GUI. Additionally, since there is no guarantee the loaded models are correctly scaled, the software requires a scaling factor to be entered by the user (e.g. on an asteroid model units may represent kilometres, whereas on a spacecraft model they can represent inches). This factor is defined as the constant that converts model units to kilometres, and by omission it has the value of 1.

The models shader is much simpler than the planetary shader, and is based on the Blin-Phong model [10], which produces fairly accurate results, as shown in Figure 4.3 for NASA's Cassini spacecraft model [12], despite not being physically-based.

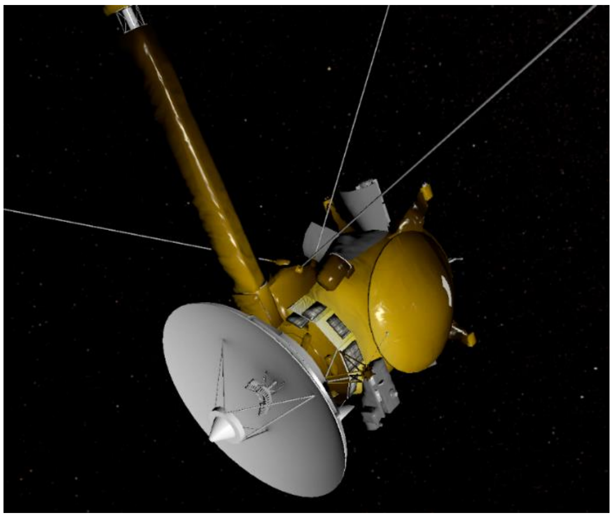


Figure 4.3 – Screenshot of the Cassini spacecraft model as rendered by SimOrbit.

4.3. Auxiliary Features

Besides rendering a realistic scene, it is also necessary to draw auxiliary features for scientific purposes. Firstly, to analyse the evolution of the position of a spacecraft as it travels through the Solar System it is necessary to draw its trajectory. Secondly, to monitor its attitude it is necessary to render local frames of reference. Lastly, to know the position of distant objects whose apparent size is too small to discern, markers need to be placed, along with labels with the object's name.

The most useful auxiliary feature that the module renders is a trajectory. It is also the most complex, and several problems arise while trying to generate and render them. Firstly, the graphics API can only draw line segments, meaning all trajectories must be reduced to a set of points. These are retrieved using an iterative method that divides each coverage window (i.e. contiguous time span for which ephemeris are known) in segments until a tolerance condition is met, or the maximum number of trajectory points is reached. Since this is a costly process, involving tens of thousands of ephemeris queries, it is not possible to do it in real time while rendering. Furthermore, the loaded trajectory cannot be adjusted when changing reference body, meaning it is always drawn with reference to the Sun.

The frames of reference (Figure 4.4) are invaluable to assess the orientation of an object, for example to determine its relative orientation to another, or to have a general idea of its flight path and lighting angles. They are composed by three perpendicular vectors, labelled X, Y, and Z, according the orientation of the primary axis of the selected frame.

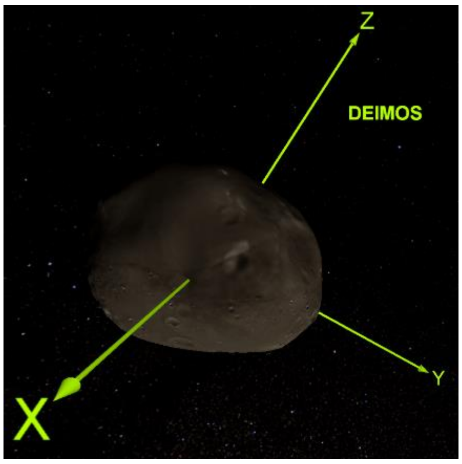


Figure 4.4 – Deimos BODY frame of reference and label.

Since the size of the bodies in the Solar System is extremely small when compared to the distance between bodies, it becomes necessary to place markers in the position of distant bodies in order to have an idea of their position. The software accomplishes this by placing four-point star-shaped markers with the name of their associated body in the positions of distant bodies, as shown in Figure 4.5.

VENUS
MERCURY
SUN

Figure 4.5 – Sun, Mercury, and Venus markers and labels.

All the auxiliary features are controllable on an object basis and can be turned on and off via GUI. Additionally, they are rendered with the light positioned at the camera, thus ensuring they are always lit, and in chartreuse colour since it is widely regarded as the most visible colour.

4.4. Cameras

Controlling the point-of-view of the simulation is essential to evaluate all aspects of a scene. Often it is necessary to have a camera near a spacecraft, in other occasions it is ideal to have it on the surface of a planet, or even the top or an orbital plane. To allow the user full control over the camera, two control modes are implemented – manual and automatic.

The manual control, provides a camera with five degrees of freedom (three-dimensional translation, pitch, and yaw) allowing the user to freely explore the scene. This camera is controlled in real time by using the keyboard for translation and the mouse for rotation as shown in Figure 4.6. The roll movement was restricted since it would require either a third degree of freedom on the mouse, or adding rotation controls to the keyboard. The camera vertical direction was defined as the vertical direction of the J2000 frame of reference.

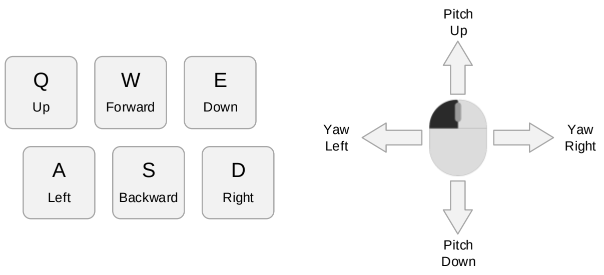


Figure 4.6 – Manual camera keyboard and mouse controls.

The automatic mode is a "must have" for presentations and directing high-quality movies. In this mode the point-of-view is controlled automatically throughout the simulation, based on a list of parametrized cameras. The cameras can be added, edited, and deleted via the GUI or loaded from one or more XML files. An automatic camera is defined by:

- Time span – The time interval in which the camera should be active, defined by a start epoch and optionally a stop epoch or duration;
- Time scale – The time that passes within the simulation for every second;
- Eye position – The position of the camera;
- Look at position – The position the camera is looking at;
- Up direction – Optional, the direction of the camera vertical.

To facilitate the parametrization of the camera the positions and direction are defined by an offset or direction from a specific reference body in one of the implemented frames of reference.

The software does not require the cameras time span to be contiguous or non-intersecting. Instead, it selects the camera based on which cameras are valid for the current epoch and gives precedence to the one that has a later start epoch. If no camera is defined it stops updating the camera parameters until another valid camera is found.

4.5. Movie Making

Since the simulator module only renderers the scene on the screen, the only possible way of saving its output is capturing a movie. Hence, a Windows Media Video (WMV) movie maker was implemented. This format was selected since it is guaranteed to be compatible with the operating systems the software is compatible with. The movie maker captures the entire window contents, with the exception of the GUI as it serves no purpose in a movie.

In order to capture a movie the user first needs to configure it, by selecting the output filename, the resolution, and the quality level. The resolution can be set to one of three options – Native (the current window resolution), HD720 (1280x720), or HD1080 (1920x1080). The quality level is a discrete value 1 to 100, where 1 represents the lowest possible quality and 100 the highest. After the output is configured, capturing can be started and stopped by pressing the appropriate button on the GUI, or the C key on the keyboard.

4.6. GUI and Simulation Control

To enable the user to control the simulation a GUI was included. The interface was designed to be as unobtrusive as possible, and consists of a semi-transparent, auto-hiding dialog on the bottom of the window, with tabs to control the scene, camera, and movie-making.

The scene tab allows the user to control all the objects being rendered. The user can load kernels containing ephemerides and orientation information for objects of interest, and load 3D models as desired. Additionally, for each object, it is possible to configure and toggle the display of auxiliary features. Time control features, namely the ability to jump to a specific date, and to slow down or fast forward the simulation were also included.

The camera tab is used to toggle between the manual and automatic control modes. It also lists the loaded automatic cameras, and provides means to add, edit, and delete them. Additionally, it shows the diagram of the manual camera keyboard and mouse controls.

Finally, the movie-making tab allows the user to configure the movie writer output filename, quality, and resolution and initiate or finish the capture of a movie.

5. TRAJECTORY PROPAGATOR VALIDATION

In order to be able to use the trajectory propagator in an operational environment it is necessary to perform an extensive validation process to ensure there are no errors which can compromise a space mission. The validation presented in this section is preliminary, and its purpose is only to confirm that the force model is capable of accurately calculating simple trajectories for which there are analytical solutions.

5.1. Keplerian Orbit

The most basic test of the trajectory propagator consists in verifying the orbital stability of a Keplerian orbit. To achieve this all perturbations were disabled, leaving only the spherical gravity field of the central body. A set of orbital elements, describing an Earth orbit, was chosen, and the trajectory was propagated for a duration of 100 days with a fixed time step of 10 minutes. The orbital elements residuals were calculated and plotted for each data point. Their mean and standard deviation were calculated (Table 5.1), and their normal probability plots were generated. The analysis reveals minor, well-behaved, normally distributed residuals for all elements with the exception of the mean anomaly at epoch (M_0). This suggests the orbit is stable with near-constant orbital parameters, as expected, but the mean motion of the spacecraft is higher than that of the analytical solution. Nevertheless, the accumulated error of M_0 over the course of the 100 days is of only $9.2606 \times 10^{-3\circ}$, well within an acceptable tolerance.

Table 5.1 – Statistical analysis of the residuals of the classical orbital elements during a 100 days propagation of a Keplerian orbit.

Element	**Mean (μ)**	**Standard Deviation (σ)**
Semi-major axis	-3.0337e-03	7.2842e-03
Eccentricity	-4.5622e-08	1.4529e-07
Inclination	2.5361e-07	1.3024e-06
Longitude of ascending node	2.2247e-06	4.7369e-06
Argument of periapsis	9.1730e-07	1.6502e-04
Mean anomaly at epoch	4.6052e-03	2.6692e-03

5.2. J_2 Perturbation

For a satellite orbiting a central body, the J_2 perturbation causes a variation of the longitude of ascending node (Ω) and the argument of periapsis (ω). Two types of orbits were used to validate the J_2 contribution – a sun-synchronous orbit, and a Molniya orbit. Sun-synchronous satellites orbit the Earth in a way that they appear to always orbit the same position from the perspective of the Sun, i.e. the orbital plane rotates at the rate of $0.9856° \cdot \mathrm{day}^{-1}$. The Molniya satellites are placed in highly elliptical orbits with an inclination near $63.4°$. At this inclination the argument of periapsis is not perturbed by the J_2 coefficient of the Earth's gravitational field, thus remaining constant.

The sun-synchronous satellite AQUA was selected to test the variation of the longitude of ascending node. The trajectory was propagated for a duration of 100 days with a fixed time step of 10 minutes, and the force system was configured to only include the Earth's gravity field. The obtained solution diverges from the analytical one as it integrates errors over time, however these were considered to be within an acceptable tolerance. The numerical solution has $\Delta\Omega = 0.9681° day^{-1}$, whereas the analytical solution has a $\Delta\Omega = 0.9915° day^{-1}$. The coefficient of correlation between the solutions is $R = 1$.

The MOLNIYA 1-81 satellite was selected to test the variation of the argument of periapsis, since from all the operational Molniya satellites, its inclination was the one closest to $63.4°$. The trajectory was propagated exactly as before. Here the error increases linearly over time, and the numerical solution reveals has $\Delta\omega = 0.0007° day^{-1}$, whereas the analytical solution has a $\Delta\omega = 0.0013° day^{-1}$. The coefficient of correlation between the two solutions is $R = 0.9487$.

Taking into account the results obtained for both the AQUA and the MOLNIYA 1-81 satellites, the J_2 perturbation implementation was considered valid.

5.3. Third Bodies and Solar Radiation Pressure

Testing the trajectory perturbations introduced by third bodies and the solar radiation pressure is a more complex problem, for which no analytical solution exists. Ideally, the software should be tested against previously validated software, unfortunately due to time constraints that was not possible. Nevertheless, it was necessary to verify that at least the influence of third-bodies was being taken into account and reasonably calculated.

To verify the influence of third-bodies a trans-lunar trajectory was propagated. This is a type of free return trajectory which takes a spacecraft from a circular parking orbit around the Earth, around the far side of the Moon, and back to Earth without requiring any propulsion, except that required to initially set the trajectory. The Apollo 11 mission trajectory, for which an abundance of information is available, was ideal for testing.

The spacecraft was considered to have a mass of $m = 43866kg$, corresponding to the mass of the Command and Service Module (CSM) and the mass of the Lunar Module (LM) combined, a projected area of $A = 27m^2$ and a reflectivity of $\varepsilon = 0.5$. The numerical integrator was set up with an initial time step of 1 minute with lower and upper bounds of 1 second and 1 hour, respectively, and truncation error tolerances of $tol_{pos} = 1 \times 10^{-6} m$ and $tol_{vel} = 1 \times 10^{-6} m \cdot s^{-1}$. The trajectory was propagated for a duration of duration of 5.8 days, with all Earth's force system perturbations enabled.

The propagated trajectory, shown in Figure 5.1, is as expected. The spacecraft leaves Earth, performs a swing-by at the Moon, and returns. Despite the lack of numerical validation of the trajectory, it is enough to verify that the third-body perturbation due to the presence of the Moon is being accurately taken into account.

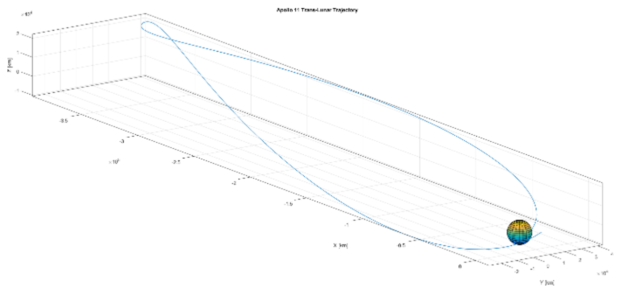


Figure 5.1 – Apollo 11 trans-lunar trajectory as propagated by SimOrbit.

5.4. Engine (Delta-V)

The engine delta-V inputs were tested by propagating a Hohmann transfer, which is a manoeuvre that takes a spacecraft from one circular orbit to another, on the same plane, using two delta-V impulses.

It was decided to use the engine to transfer a satellite from Low Earth Orbit (LEO) to a Geosynchronous Equatorial Orbit (GEO), which has an altitude of 35,786km above the equator. The engine was configured with the two delta-V inputs, obtained analytically, and the trajectory was propagated for the duration of 3 days, starting 1 day before the first delta-V. The time step was fixed to 10 minutes. The resulting trajectory is represented on Figure 5.2, and is exactly what was expected.

The semi-major axis error is $\varepsilon_a = 1 \times 10^{-4} km$, and the eccentricity error is $\varepsilon_e = -1.4330 \times 10^{-7}$, both considered to be well within an acceptable tolerance, thus validating the engine's delta-V implementation.

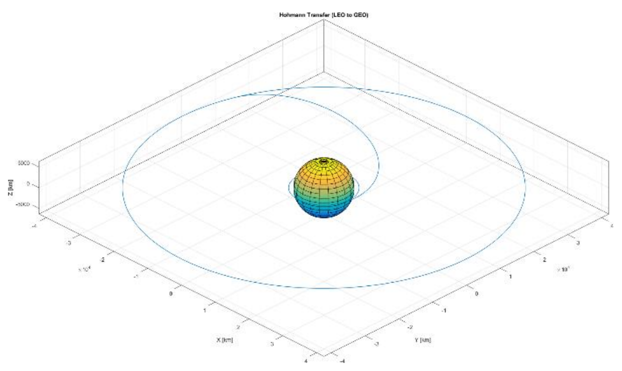


Figure 5.2 – Hohmann transfer orbit from Low Earth Orbit (LEO) to Geosynchronous Equatorial Orbit (GEO).

6. PERFORMANCE

The SimOrbit performance was assessed on an Intel Core i7-5600U CPU @ 2.60GHz, with 8GB RAM, and an integrated Intel HD Graphics 5500 graphics processor, running the Windows 7 Professional 64-bit operating system. The display resolution was 2560x1440 pixel. The executable used for testing was built with full program optimization and for a 64-bit system.

6.1. Propagator

To assess the propagator module performance, a trajectory propagation of an Earth satellite, with a duration of one sidereal year, was used. The force system was configured to account for all perturbations (i.e. Earth J_2 coefficient, Moon and Sun as third bodies, and solar radiation pressure). Additionally, a very low-thrust constant acceleration engine input was added to be able to measure the performance of the engine. The initial time step was set to 10 minutes and was left unbounded.

Firstly, the results of a CPU sampling were analysed. One "hot path" was found. The propagator worker thread function was responsible for 96.9% of the CPU usage, with 73.0% corresponding to the RKF7(8) integrator and 22.6% to the dynamic force system change algorithm. Looking inside the integrator it was found that 64.3% of the total CPU usage came from that force model implementation.

To gain a better understanding of what force model components were more expensive to compute, several performance counters, based on the High Precision Event Timer (HPET), were added to the force model code. The implemented counters measured how long it took to execute the code that calculates each component of the acceleration. Analysing the performance counter results, shown in Table 6.1, it is obvious the most time consuming computations are the central body J_2 and solar radiation pressure perturbations.

Table 6.1 – Force model performance counter results.

Performance Counter	Computation Time [s]
Central body GM	4.5101
Central body J_2	38.6601
Third bodies GM	14.5795
Sun GM	14.1066
Solar radiation pressure	52.8765
Engine	6.7374
Total	**133.7283**

6.2. Simulator

The simulator module testing focused on measuring the frames per second (FPS) under different conditions to try and assess the individual performance of the shaders, and the influence movie capturing had on graphics performance. The Direct3D 10 mode was used in the tests, since it was supported by the graphics processor, and the FPS was measured using the Visual Studio Graphics Analyzer.

The first test was run on windowed mode at 1600x900 pixel. A sequence of six automated cameras that focused on objects being rendered by different types of shaders were defined. Each camera was set to be active for a total of 30 seconds. The sequence was ran twice, with native resolution movie capture off and on. Table 6.2 lists the measured average FPS values and the influence of movie capturing.

Table 6.2 – Average FPS measured for different types of shaders on a 1600x900 pixel window, with native resolution movie capturing off and on.

Shader	Rendering [FPS]	Capturing [FPS]
None	319.4	125.4
Sun	97.7	**59.6**
Planet	247.7	95.7
Planet with atmosphere	109.4	**55.8**
Rings	226.4	106.7
Model	265.7	100.4

The worst performance was found on the "Sun" (i.e. the Perlin noise implementation) and the "Planet with atmosphere" shaders, which is not surprising since both are computationally expensive algorithms. Nevertheless, the performance of all shaders is very good. Capturing a movie at the native resolution caused an average increase of 6.9ms in the frame rendering time.

Subsequently, full screen mode was tested at the 2560x1440 pixel, commonly known as Quad HD (QHD). This test was meant to put as much strain as possible on the graphics processor and determine the lower FPS limits at which a simulation would run. The camera sequence and methodology from the previous test were used, and the results are shown in Table 6.3. As expected, due to the 2.56 times higher resolution, the FPS are lower than those of the previous test, although the relative shader performance remains constant. Movie capture at this resolution was found to reduce the FPS to very low values, where camera movement causes noticeable flicker.

Table 6.3 – Average FPS measured for different types of shaders on a 2560x1440 pixel screen, with native resolution movie capturing off and on.

Shader	Rendering [FPS]	Capturing [FPS]
None	208.7	**39.3**
Sun	**56.2**	**25.4**
Planet	167.9	**36.9**
Planet with atmosphere	60.4	**31.1**
Rings	153.4	**35.6**
Model	175.6	**37.2**

7. CONCLUSIONS

This work was aimed at developing a free and open-source software package to support mission analysis and orbital mechanics calculations. In order to provide an integrated solution, enabling a user to generate and analyse data, two components were developed – a trajectory propagator for data generation, and a 3D visualization component. The trajectory propagator is able to generate accurate spacecraft trajectory data based on a set of initial conditions describing its state (i.e. position and velocity) at a reference epoch. It generates a binary SPK and plain text output which can be easily used by other tools in the mission design and analysis process. The 3D visualization component produces a visually realistic rendering of the Solar System and spacecraft of interest. It also displays auxiliary features to aid mission analysis and is able to export movies.

Although the developed software conforms to the specifications, performs adequately, and has been preliminary validated, it was found to have some flaws. On the propagator module, a very short constant acceleration engine input (i.e. with a time span with an order of magnitude similar or lower than that of the time step) can be leaped over and ignored by the numerical integrator. In order to mitigate this problem the software resets the initial time step at the start of the engine input, but this was found to be inadequate to solve the issue. A recommendation was included in the user manual to use delta-V inputs, instead of constant acceleration, in these cases. The propagator should also be modified to allow for the propagation of spacecraft attitude and mass in addition to the position and velocity. This would greatly improve the accuracy of the solar radiation pressure component, and allow for propellant calculations. On the simulator module, loading a spacecraft trajectory is a time consuming operation, and for very long or complex trajectories, limited by the maximum number of points, it does not always produce an accurate representation of the reality. This issue needs further studying, possibly creating an algorithm that refines the trajectory depending on the camera position. Furthermore, also on the propagator module, the Sun's corona shader has the lowest performance of all implemented shaders and is not physically-based. A study of the corona's lighting properties needs to be performed in order to create a fast physically-based algorithm that improves both the visual realism and performance of this shader.

The design and development of the software was a challenging endeavour, as it draws knowledge from several distinct areas – orbital mechanics, programming, computer graphics, optics, and media generation. The work also greatly contributed towards understanding the difficulties associated with large software projects, and their planning and time management requirements. Extensive computer programming knowledge, namely in the C++11 and High-Level Shading Language (HLSL) programming languages, was also acquired. Furthermore, the importance of defining requirements, establishing standards, and producing appropriate documentation was learnt.

Overall, the software is considered to successfully fulfil the proposed objective, allowing a user to accurately propagate and visualize the trajectory of spacecraft in a visually realistic environment.

8. REFERENCES

[1] "Information technology – Programming languages – C++," ISO/IEC Standard 14882:2011, 2011. [Online]. http://www.iso.org/iso/catalogue_detail.htm?csnumber=50372

[2] Charles H. Acton, "Ancillary data services of NASA's Navigation and Ancillary Information Facility," *Planetary and Space Science*, vol. 44, no. 1, pp. 65-70, January 1996. [Online]. http://dx.doi.org/10.1016/0032-0633(95)00107-7

[3] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding," *ACM Trans. Math. Softw.*, vol. 33, no. 2, p. Article 13, June 2007. [Online]. http://dx.doi.org/10.1145/1236463.1236468

[4] MPIR Team. (2014, July) MPIR. [Online]. http://www.mpir.org

[5] Assimp Development Team. (2013) Open Asset Import Library. [Online]. http://assimp.sourceforge.net

[6] European Commission. (2009, November) European Union Public Licence - EUPL v.1.1. [Online]. http://ec.europa.eu/idabc/eupl.html

[7] Oliver Montenbruck and Eberhard Gill, *Satellite Orbits - Models, Methods, and Applications*, 1st ed. Heidelberg, Germany: Springer-Verlag Berlin, 2000.

[8] Csaba Kelemen and László Szirmay-Kalos, "A Microfacet Based Coupled Specular-Matte BRDF Model with Importance Sampling," in *EUROGRAPHICS 2001*, 2001. [Online]. http://www.fsz.bme.hu/~szirmay/scook.pdf

[9] Robert L. Cook and Kenneth E. Torrance, "A Reflectance Model for Computer Graphics," *ACM Transactions on Graphics*, vol. 1, no. 1, pp. 7-24, January 1982. [Online]. http:/dx.doi.org/10.1145/357290.357293

[10] James F. Blinn, "Models of light reflection for computer synthesized pictures," *SIGGRAPH Comput. Graph.*, vol. 11, no. 2, pp. 192-198, July 1977. [Online]. http://dx.doi.org/10.1145/965141.563893

[11] NVIDIA Corporation, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Matt Pharr and Randima Fernando, Eds. Boston, MA, USA: Addison-Wesley Professional, 2005. [Online]. http://http.developer.nvidia.com/GPUGems2/gpugems2_frontmatter.html

[12] National Aeronautics and Space Administration. (2015, May) NASA 3D Resources. [Online]. http://nasa3d.arc.nasa.gov