Two-Line Element Decoder and Visualizer

UNM ECE 535 Final Project: Software Type

Sonny Ji
Department of Electrical and Computer Engineering
The University of New Mexico
Albuquerque, United States
velten@unm.edu

***Abstract*—This document details the conceptualization, design, implementation, and finalization of a software tool for visualizing the orbits of satellites. The goal of this tool is to aid in the visualization of satellites and their orbits, along with a presentation that promotes a human-digestible format for better intuition and understanding of satellite orbital mechanics and satellite communication ranges.**

***Index Terms*—satellite communications, TLE, orbital mechanics, 3D-simulation, software**

I. Introduction

This is a project for the ECE 535 Satellite Communications class at The University of New Mexico's Department of Electrical and Computer Engineering. The goal of this project is to develop and build a desktop application for decoding two-line element sets and representing the results in an interactive 3D graphic that tracks the satellite. Additionally, this tool will attempt to further append satellite information through pinging web-based APIs for further satellite information and data. This would include items (when applicable) such as: orbit lines, satellite use and type, cone of coverage, trajectory, Earth for scale, and visualizations of all 6 classical orbital mechanics parameters. The deliverable for this project is an application / software package for taking in two-line element sets as an input and presenting 3D models of the previously described as an output, along with a brief guide on how to use and interpret the tool.

Information on the tool and a guide on how to use it are included at the end of this document under the *Usage Information and Notes* section.

II. Useful Information and Definitions

A. Abbreviations and Acronyms

- TLE: Two-Line Element
- AN: Ascending Node
- DN: Descending Node
- RAAN: Right Ascension of the Ascending Node

B. Variables and Symbols

- a: Semi-major axis of elliptical orbit
- b: Semi-minor axis of elliptical orbit
- e: Eccentricity of elliptical orbit
- i: Inclination of elliptical orbit
- Ω: RAAN, longitude of the ascending node
- ω: Argument of periapsis of elliptical orbit
- ν: True anomaly of elliptical orbit
- Υ: First point of Aries, vernal equinox
- $\mathfrak{Q}$: AN
- $\mathfrak{V}$: DN
- M: Mean anomaly of elliptical orbit
- E: Elliptical anomaly of elliptical orbit
- n_0: Mean motion of elliptical orbit
- R: Radius of celestial body
- r: Radius of orbit with respect to central orbiting body

C. Subscripts

- $\cdot_a$: Relating to the apoapsis of an orbit
- $\cdot_p$: Relating to the periapsis of an orbit
- $\cdot_\oplus$: Relating to the Earth
- $\cdot_0$: Relating to epoch

D. Units

- All distances (i.e. a, r_p, $R_\oplus$, etc.) are presented in kilometers
- All velocities (i.e. v_p, v_a, $\vec{\mathbf{v}}$, etc.) are presented in kilometers per second
- All angles (i.e. ν_0, E_0, M_0, etc.) are presented in radians
- All angular velocities (i.e. $\omega_\oplus$, n, etc.) are presented in radians per second
- All durations (i.e. Δt, etc.) are presented in seconds

E. Constants

- $R_\oplus = 6378\,\mathrm{km}$
- $\omega_\oplus = 7.292\,115\,9 \cdot 10^{-5}\,\mathrm{rad/s}$
- $\mu_\oplus = 3.986 \cdot 10^5\,\mathrm{km^3/s^2}$

III. Motivations and Methodologies

A. Motivations

As someone with a background in mechanical engineering, I found the orbital mechanics section of ECE 535 incredibly interesting. However, I wanted something to help me, and others with the same problems, visualize satellites and their orbits. Since the position of a satellite in a 3D orbit is somewhat difficult to grasp if it's your first exposure to the concept, a tool that helps bridge the gap between just understanding the math and numerical values behind the satellite, and gaining an intuition for how those numbers would actually

affect a satellite and its orbit without having to do multiple orbital mechanics calculations, would be incredibly useful for students and anyone else just starting out in orbital mechanics. So, for my project, I decided to create a tool to visualize satellites and their orbits.

Additionally, this program also helps to provide a tool to decode and interpret TLEs. Since TLEs contain all of the important information relevant to positioning a satellite in an orbit around the Earth, they serve as a great tool for user data input for selecting a satellite to visualize. Thus, I also decided to include a TLE "translator" in this project as well. Overall, the reason this project is important is that it will be a very useful tool for students learning about orbits and satellite communications to help give a more graphical and intuitive understanding of how satellites are positioned where they are, and how that affects the world.

B. Methodologies

While there are many orbit visualizers and simulators out there, a game called Kerbal Space Program was incredibly helpful for me in understanding orbits and satellites through its representation of orbits in a 3D interactive model, and definitely helped me gain a somewhat-intuitive understanding of orbital mechanics and the philosophies and principles behind the design, construction, and maintenance of communication satellites and satellite networks. Thus, I decided to expand on Kerbal Space Program and make my own visualization method specifically for ECE 535 roughly based on Kerbal Space Program's graphical representations.

Furthermore, I decided to use Unity as the central graphics engine for this project. This is because Unity is a widely-used 3D game development platform and rendering engine, with a very wide compatibility for what machines it could compile and build for. Additionally, due to its large amount of user-generated help and support forums, along with its documentation for how to use and program for the Unity ecosystem, it seemed like a good way to both learn the fundamentals of Unity while also producing a helpful tool for the final project of ECE 535.

IV. Implementation Philosophy and Ontology

The high-level goal of this project is to create a tool that visualizes the six classical orbital elements: a, e, i, Ω, ω, and ν. The tool must also represent these quantities graphically, in a way that demonstrates exactly what the variable is measuring. Additionally, the tool must visualize celestial elements that help a student's understanding and intuition: Υ, Ω, $\mho$. Finally, the tool must take a full, correctly formatted TLE as an input.

V. Governing Physics and Equations

A. Background for General Orbital Mechanics

A satellite's orbit around the Earth can be fully described through the six orbital elements.

The first orbital element, a, is the semi-major axis of the orbit. This is useful in determining approximately how large an orbit is. Additionally, it is also the semi-major axis of the ellipse as seen from directly above the orbital plane, or in the negative $\hat{\mathbf{W}}$ direction in the perifocal coordinate system, as demonstrated in Fig. 1.

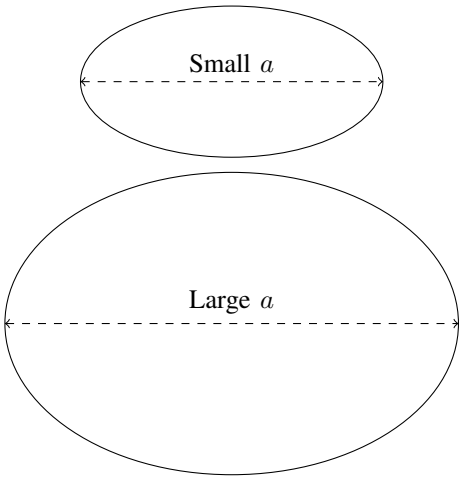

Fig. 1. Small and large semi-major axes.

The second orbital element, e, is the eccentricity of the orbit. This is useful in determining the aspect ratio of the orbit, or how "squished" an elliptical orbit is. Additionally, it is also the eccentricity of the ellipse created by the orbit as demonstrated in Fig. 2.

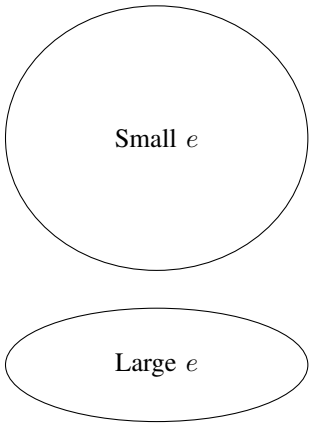

Fig. 2. Small and large eccentricities.

The third orbital element, i, is the inclination of the orbit. This is useful in determining the tilt of the orbit from the celestial equatorial plane as demonstrated in Fig. 3.

The fourth orbital element, Ω, is the RAAN, or the longitude of the AN. It is the angle between Υ and Ω measured as a positive rotation on the celestial equator, or eastward from Υ. This is useful in determining how rotated an orbit is on Earth's equator as demonstrated in Fig. 4. Because the stars and constellations generally don't move appreciably on a human timescale, the standard coordinate system used is based on the position of the stars and is called the celestial coordinate system. The first two planes of importance for

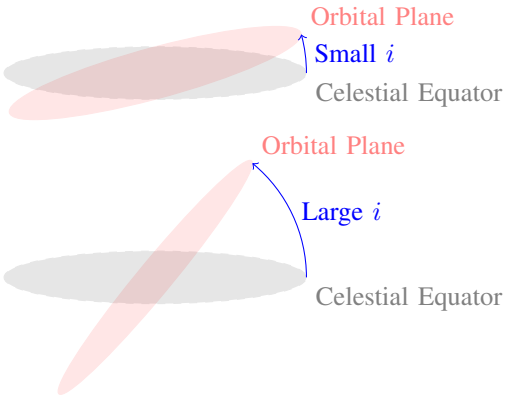

Fig. 3. Small and large inclinations.

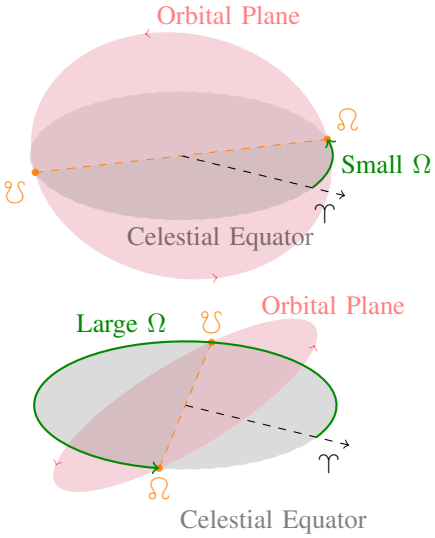

Fig. 4. Small and large RAANs.

celestial mechanics are the celestial equatorial plane, or the plane created by the Earth's equator, and the ecliptic plane, or the plane created by the Earth's orbit around the Sun. For Earth-orbiting satellites, the celestial equatorial plane is always used as the reference plane that the orbital elements are measured off of. Additionally, this system has a reference direction, designated $\hat{\mathbf{I}}$, in the direction of the vernal equinox or the First Point of Aries, ♈. This direction is determined when the line created by the intersection of the celestial equatorial plane and the ecliptic plane perfectly points toward the gravitational center of the Sun, meaning the Sun is coplanar with both the equatorial plane and the ecliptic plane. Specifically, the ♈ vector points towards the Sun when the Sun appears to move from the Southern Hemisphere towards the Northern Hemisphere. Finally, the last plane of interest for orbital mechanics is the orbital plane, or the plane created by a satellite's orbit around the Earth. The intersection of the orbital plane and the equatorial plane creates a line called the line of nodes. The points at which this line intersects the satellite's orbit are labeled the AN, or ☊, and DN, or ☋. Specifically, the AN is the point when the satellite crosses from below the equatorial plane to above it, and the DN is the point when the satellite crosses from above the equatorial plane to below it.

The fifth orbital element, ω, is the argument of periapsis. It is the angle between ☊ and the periapsis of the orbit, measured in the direction of the path of the orbiting body. This is useful in determining how rotated an orbit is in its orbital plane, with respect to the celestial equator, or the line of nodes, as demonstrated in Fig. 5.

The sixth and final orbital element, ν, is the true anomaly. It is the angular position of the orbiting body in its orbit measured from the point about which the body orbits, or the focus of the ellipse of the orbit, in the direction of the path of the orbiting body, with respect to its periapsis. This is useful in determining exactly where the orbiting body is in its orbiting path as demonstrated in Fig. 6.

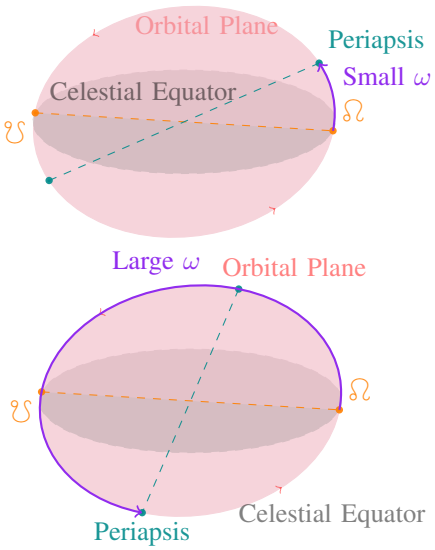

Fig. 5. Small and large arguments of periapsis.

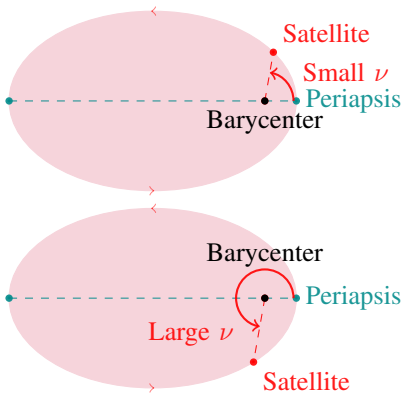

Fig. 6. Small and large true anomalies.

B. Classical Orbital Elements

Putting together all six of these classical orbital elements, any orbiting body can be fully described and positioned, as demonstrated in Fig. 7.

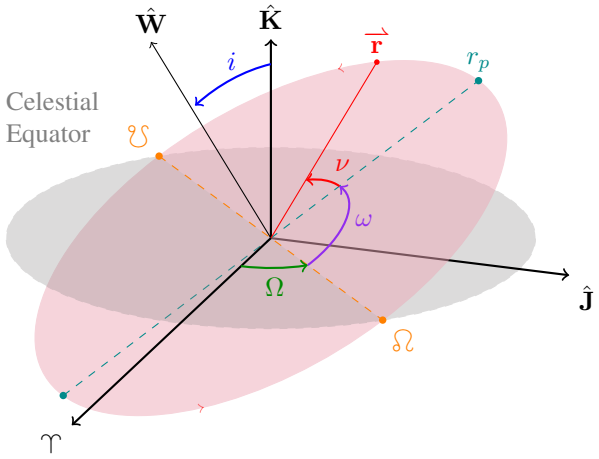

Fig. 7. All classical orbital elements.

VI. 3D Development in Unity

This project is broken up into three major sections for ease of compartmentalization and organization: camera control, TLE translation, and orbit visualization. The camera control section is important for providing an intuitive and friendly user experience to best aid the tool in its main goals. The TLE translation is important for providing the base functionality for input management of the tool. Finally, the orbit visualization is important as that is the main goal of this project. The project is broken up into this order because it was thought of as the most efficient way to slowly integrate myself into the Unity ecosystem and learn how to use Unity before getting into the much more difficult aspects of creating an orbit in 3D. The camera control section is first to learn how to translate and rotate objects, in this case, the camera and player, in a 3-dimensional space using Unity's standards, functions, and constraints. Next, the TLE translation section is to learn how to enable the user to enter text inputs and how to parse and handle strings within Unity. Finally, the orbit visualization section combines everything learned in the two previous sections and renders an orbiting satellite, along with the satellite variables and the orbit path, in a 3D environment that the player can explore. In order to assist the development and derivation of the equations of motion for the satellite's orbit, the online graphing calculator Desmos is used to test out the equations.

A. Player and Camera Control

The camera is controlled by the classical `WASD` keyboard keys and the mouse, which have become a de facto standard for all PC games. However, due to this project essentially being a "space game", the camera movement should fit the theme as well. Thus, a "smooth" camera was the desired product; instead of the jerky, instant movement that FPS games are known for, the camera should smoothly track the mouse and keyboard movements, with a gradual slow-down instead of an instant stop. This effect is achieved through the custom C# script `cameraMovement.cs`.

For the camera rotation, the raw mouse `x` and `y` positions are taken as an input and parsed into a linear interpolation function, `Mathf.Lerp()`, along with a time delta so that the `x` and `y` values move towards the actual raw mouse positions with a set amount of lag. This also has the additional effect of slowing the rate-of-change of the coordinates down as they approach the true raw values of the mouse. The `x` coordinate is then clamped between $-90°$ and $+90°$ so that the user cannot look up or down further than the vertical, which helps significantly in not disorienting the user or causing motion sickness. Finally, these coordinates are interpreted as Euler angles as a 3-tuple, using the datatype `Vector3`, and the camera rotation is transformed based on these angles.

For the camera position translation, the keyboard keys are internally mapped to various axes internally by Unity. This results in the following axes and their respective keys for a positive and negative input to the axis, respectively:

- `Axis : + | -`
- `Horizontal: D | A`
- `Vertical : W | S`
- `Jump : Space | L Shift`

From this, the "strength" of each axis is parsed into a `Vector3` position translation vector. This vector is then interpreted as a force applied to the camera in order to move it. Since the movement is treated as a force, this allows for the movement to be affected, and subsequently slowed down by an opposing drag force, creating a motion that feels like the camera is sliding around on a smooth surface, which also invokes the feelings of zero-gravity and outer space.

B. Parsing TLEs

The user inputs TLEs by pasting a correctly-formatted string of text into an input box. Once the input textbox is no longer selected, or the user has pressed the `Enter` key, the function for parsing in TLEs, `readStringInput()`, inside of the script `readTLE.cs`, is called and begins its calculations. From this, everything in the TLE textbox is loaded into the local variable `TLE`. Since TLEs have a very strict formatting constraint, substrings for each orbital element can be extracted by simply calling the string method `<String>.Substring()`. In order to make the index math simpler, the second line of the TLE is further extracted and placed into the variable `line2`. All relevant sections are then extracted into their respective variables as follows:

- i: `i = line2.Substring(8, 8);`
- Ω: `Omega = line2.Substring(17, 8);`
- e: `e = line2.Substring(26, 7);`
- ω: `omega = line2.Substring(34, 8);`
- n_0: `n_0 = line2.Substring(52, 11);`

For almost all of these variables, the only calculations required are converting the value from degrees to radians. However, in

order to calculate the semi-major axis, the following calculation must be performed on the mean motion:

$$a = \sqrt[3]{\frac{\mu_\oplus}{n_0^2}} \tag{1}$$

Equation (1) is derived from the following Keplerian relationship, or more specifically, Kepler's third law:

$$a^3 = \frac{\mu_\oplus}{n_0^2} \tag{2}$$

Thus, all of the relevant orbital elements can be retrieved from this script by calling their respective methods from the textbox object.

C. Initial Desmos Development

Given Desmos' very user-friendly UX, easily editable graphical expressions, and LaTeX support for equation and expression formatting, it was used as an initial base platform to develop all of the equations required to display the orbit of the satellite in Unity. More specifically, Desmos' 3D graphing calculator was used to create the coordinate axis rotations. The end result of this development can be viewed at this link to the final iteration of the calculator: https://www.desmos.com/3d/8e16126c8f.

Starting off, the actual orbit itself is simply an ellipse, so that was used as an initial point for development. The equation for graphing an ellipse is as follows:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{3}$$

Equation (3) creates an ellipse with a semi-major axis of a and a semi-minor axis of b. However, since Unity uses vector coordinates to transform and render objects, converting (3) into a parametric representation would make it much easier to integrate into Unity. Since an ellipse is essentially an elongated circle, the same expressions for the coordinates of a circle can be used, but with a scalar modification for each axis. Thus, parameterizing (3), the following is derived:

$$(x, y) = (a\cos(E), b\cos(E)),\ E \in [0, 2\pi) \tag{4}$$

In (4), E is the eccentric anomaly of the ellipse, which can be thought of as θ for polar coordinates. The eccentric anomaly is important because it is used to calculate the motion of orbiting bodies. Furthermore, since orbiting objects actually orbit around the barycenter of the 2-body system instead of around the center of the orbital ellipse, the ellipse had to be shifted so that one of the foci of the ellipse was located at the origin of the graph. In order to do this, the distance from the center of the ellipse to one of its foci must be calculated. Due to the properties of an ellipse, it is known that the distance from one focus to one of the ends of the semi-minor axis is equivalent to the length of the semi-major axis, as demonstrated in Fig. 8. From this, using the Pythagorean theorem, the following equation is derived:

$$c = \sqrt{a^2 - b^2} \tag{5}$$

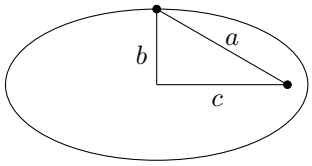

Fig. 8. Distance from center to focus for an ellipse.

Applying this offset to the parametric ellipse, the following is derived:

$$(x, y) = (a\cos(E) - c, b\sin(E)) \tag{6}$$

Equation (6) shifts the ellipse to the left, in the negative-x direction, so that the periapsis of the orbit is at $E = 0\,\text{rad}$. Finally, since TLEs do not contain an expression for the length of the semi-minor axis, b must be derived from a and e as follows:

$$e^2 = 1 - \frac{b^2}{a^2} \tag{7}$$

$$\Rightarrow e^2a^2 = a^2 - b^2 \tag{8}$$

$$\Rightarrow b = \sqrt{a^2 - e^2a^2} \tag{9}$$

Then, using (5), the following is derived:

$$c = ea \tag{10}$$

$$\therefore (x, y) = \left(a\cos(E) - ea, \sqrt{a^2 - e^2a^2}\sin(E)\right) \tag{11}$$

From this, various 3D rotations are performed in on the ellipse around the origin in order to achieve the final orbit ellipse.

Out of the classical orbital elements, i, Ω, and ω are the rotational elements: i rotates the orbit around the line of nodes; Ω rotates the orbit around the barycenter in the celestial equatorial plane; and ω rotates the orbit around the barycenter in the orbital plane. Thus, the simplest order of rotations would be to first rotate the ellipse in the orbital plane, so that it is a simple 2-dimensional rotation, then rotate the entire orbital plane around the line of nodes, which would be colinear with the $\hat{\mathbf{I}}$ axis at this point, and then finally rotate the line of nodes and the orbital plane around the origin, which would essentially be another 2-dimensional rotation, keeping all of the $\hat{\mathbf{K}}$ components static. From Fig. 9 and the following

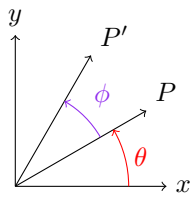

Fig. 9. 2D rotation.

equations, it is demonstrated that, given an initial point of

a graph P, and a desired rotation ϕ, a desired final point P' can be achieved:

$$P' = \begin{cases} x' = r\cos(\theta)\cos(\phi) - r\sin(\theta)\sin(\phi) \\ y' = r\cos(\theta)\sin(\phi) + r\sin(\theta)\cos(\phi) \end{cases} \tag{12}$$

$$P = \begin{cases} x = r\cos(\theta) \\ y = r\sin(\theta) \end{cases} \tag{13}$$

$$\therefore P' = \begin{cases} x' = x\cos(\phi) - y\sin(\phi) \\ y' = x\sin(\phi) + y\cos(\phi) \end{cases} \tag{14}$$

Applying (14) to the base orbital ellipse as a rotation of ω, a rotation matrix $[\boldsymbol{R_\omega}]$ can be derived:

$$[\boldsymbol{R_\omega}] = \begin{bmatrix} \cos(\omega) & -\sin(\omega) \\ \sin(\omega) & \cos(\omega) \end{bmatrix} \tag{15}$$

Equation (15) rotates the orbital ellipse, and by extension the periapsis, around the barycenter, so that the periapsis is ω away from ☊ in the direction of the orbit, thus achieving the desired effect of the orbital element ω, as demonstrated in Fig. 10 and Fig. 11.

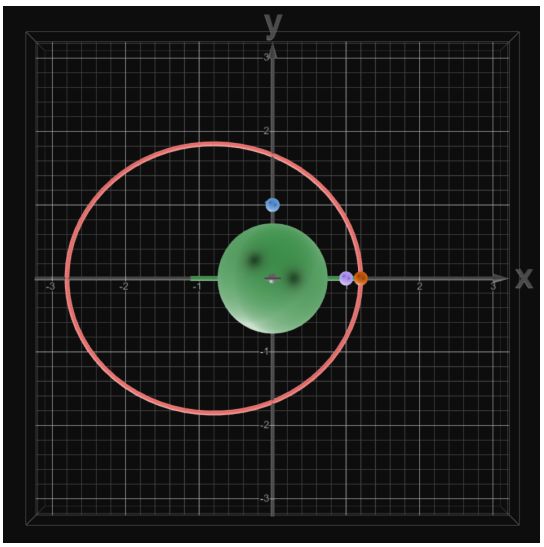

Fig. 10. Orbital ellipse before $[\boldsymbol{R_\omega}]$.

Next, the entire orbital plane must be rotated around the line of nodes. In its current state, the line of nodes is colinear with the x-axis, meaning rotating the graph's yz-plane around the origin would achieve the desired effect. Thus, using the output 2-dimensional vector of the last matrix as the input for the next matrix, a rotation matrix $[\boldsymbol{R_i}]$ can be derived. Given that the z-component of the input vector is zero, since the base orbital plane is only in the xy-plane, the third column of $[\boldsymbol{R_i}]$ can be omitted:

$$[\boldsymbol{R_i}] = \begin{bmatrix} 1 & 0 \\ 0 & \cos(i) \\ 0 & \sin(i) \end{bmatrix} \tag{16}$$

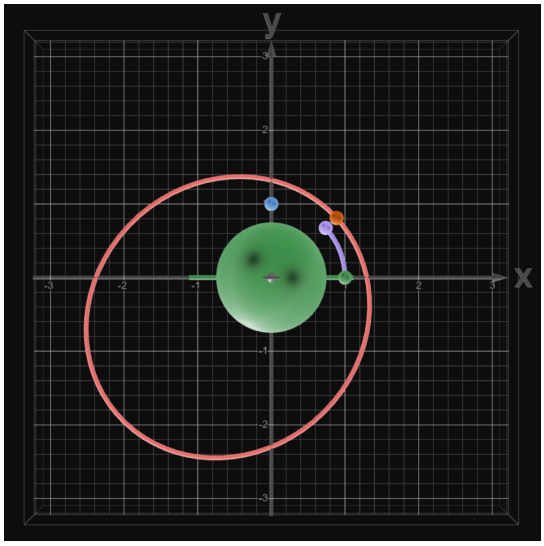

Fig. 11. Orbital ellipse after $[\boldsymbol{R_\omega}]$.

Equation (16) rotates the orbital plane around the line of nodes, so that the orbital plane is i away from the celestial equatorial plane, achieving the desired result, as demonstrated in Fig. 12 and Fig. 13.

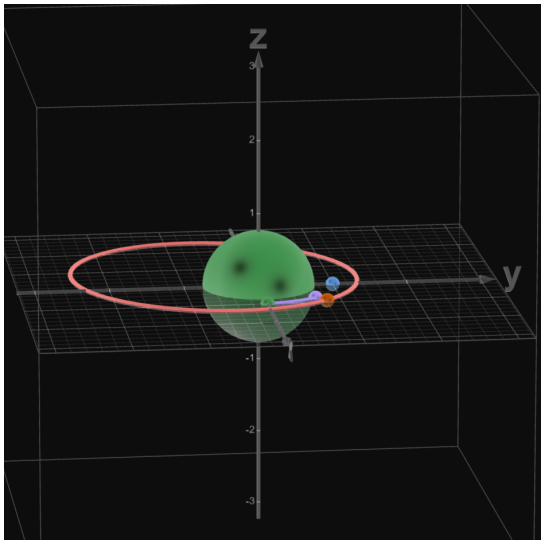

Fig. 12. Orbital ellipse before $[\boldsymbol{R_i}]$.

Finally, the result of (16) must be rotated around the z-axis. Since this is simply another rotation of the xy-plane without affecting the z-component of any coordinate, a rotation matrix similar to that of (15), $[\boldsymbol{R_\Omega}]$, can be employed:

$$[\boldsymbol{R_\Omega}] = \begin{bmatrix} \cos(\Omega) & -\sin(\Omega) & 0 \\ \sin(\Omega) & \cos(\Omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{17}$$

Equation (17) rotates the entire orbit around the celestial pole, so that the line of nodes is Ω away from ♈, achieving the desired result, as demonstrated in Fig. 14 and Fig. 15.

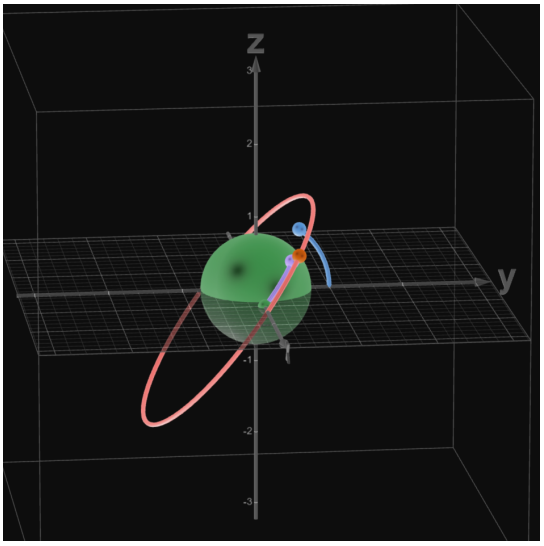

Fig. 13. Orbital ellipse after $[\boldsymbol{R_i}]$.

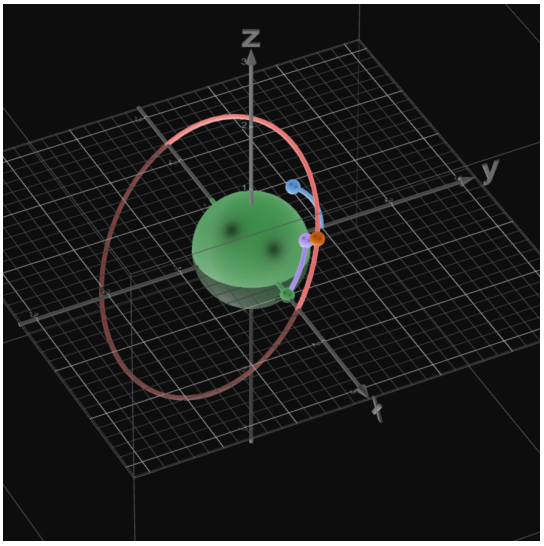

Fig. 14. Orbital ellipse before $[\boldsymbol{R_\Omega}]$.

Putting all of these rotation matrices together, the following equation describes the entire translation from a simple ellipse drawn from a and e to a 3-dimensional orbit of a satellite.

$$\begin{Bmatrix} I \\ J \\ K \end{Bmatrix} = [\boldsymbol{R_\Omega}]\,[\boldsymbol{R_i}]\,[\boldsymbol{R_\omega}] \begin{Bmatrix} a\cos(E) - c \\ b\sin(E) \end{Bmatrix} \tag{18}$$

D. Implementation in Unity

The resulting matrices and equations from Desmos are replicated using the `Mathf` class in the main satellite script, `orbit.cs`, within Unity. The `cameraMovement.cs` script is attached to the main camera object; the `readTLE.cs` script is attached to the textbox object; and the `orbit.cs` script is attached to the satellite object. This then concludes the main effort of the project. Aside from the main deliverable, additional features were also added, such as a rotating Earth, a start button, an exit button, and a textual display for all six classical orbital elements.

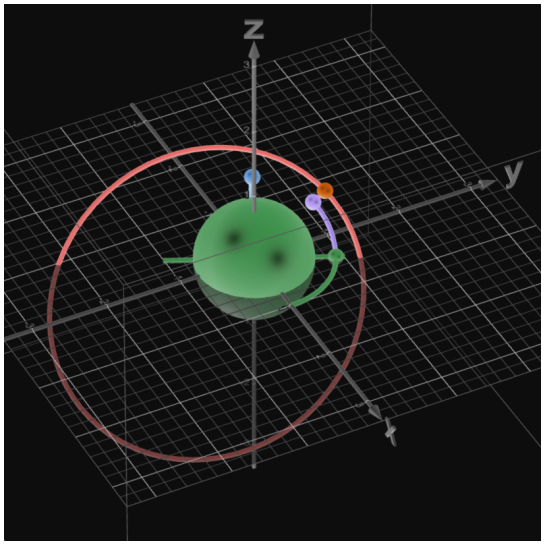

Fig. 15. Orbital ellipse after $[\boldsymbol{R_\Omega}]$.

The rotation of the Earth is set based on the framerate of the program, so that it also matches the rotation of the satellite. In testing, using the International Space Station's TLE, through one full rotation of the Earth, the satellite object completed approximately 15.5 orbits, which is consistent with the actual International Space Station, confirming the periodic accuracy of the program.

VII. Conclusion

A. Project Deliverables

The final deliverable for the project is a Windows software tool, built in Unity, that takes in a TLE as an input and creates a 3D model of the orbit of the satellite that the TLE corresponds to, in an interactive 3D environment. A brief guide on how to use the tool is also included as a deliverable at the end of this document.

B. Challenges and Difficulties

Many challenges and difficulties were faced throughout the development of this project. Chiefly among those were learning Unity for the first time and understanding all of the relationships and quirks in the Unity object management system, deconstructing all six of the classical orbital elements into something comprehensible, and developing a coherent set of rotation matrices to represent those orbital elements.

Additionally, the major time constraint of needing to finish such a large project in less than half a semester, while also spending time on other courses, proved to be incredibly challenging.

C. Thoughts on Project and Future Development

For additional development in the future, I would like to add in all of the visual elements that I did not have time to add into this initial release of the project, such as all of the visualizations of the orbital elements as seen in the Desmos figures, as well as a transparent cone to demonstrate where geographically a satellite would be able to communicate with.

USAGE INFORMATION AND NOTES

The program is started by running the `Satellite Visualizer.exe`.

A TLE can be entered or pasted into the textbox labeled `ENTER TLE:`.

The camera is moved via `WASD, Space, L Shift` and the mouse.

Clicking the green button labeled `START` after having input a valid TLE will begin all rotations and orbits.

Clicking the red button labeled `CLOSE PROGRAM` will close out of the program.

The source code for my final project can be found in the GitHub repository for this project: https://github.com/Veloraxzia-Tenebris/Satellite-Visualizer.git. The playable release can be found here: https://github.com/Veloraxzia-Tenebris/Satellite-Visualizer/releases/tag/v1.0.0