Visvesvaraya Technological University

Belagavi, Karnataka, 590 014.

A Mini Project Report on

"SIMULATION OF SATELLITE LAUNCHING USING OPENGL"

Submitted in partial fulfillment of the requirements for the award of

Bachelor of Engineering

in

Computer Science and Engineering

Semester VI

(18CSL67)

Academic Year 2022-23

Submitted By

Mr. KARTIK P HEGADI	**2KE20CSO32**
Ms. KEERTI S PATIL	**2KE20CS035**
Mr. KUNTHUNATH M N	**2KE20CS039**
Ms. MEGHANA TADAS	**2KE20CS046**

Department of Computer Science AND Engineering

K. L. E. SOCIETY'S

K. L. E. INSTITUTE OF TECHNOLOGY,

Opp. Airport, Gokul, Hubballi-580 027

Phone: 0836-2232681 Website: www.kleit.ac.in

K. L. E. SOCIETY'S

K. L. E. INSTITUTE OF TECHNOLOGY,

Opp. Airport, Gokul, Hubballi-580 030

Phone:08362232681 Website: www.kleit.ac.in

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

Certified that the mini project work entitled **"SIMULATION OF SATELLITE LAUNCHING USING OPENGL"** is a bonafide work carried out by **Mr. KARTIK P HEGADI, Ms. KEERTI S PATIL, Mr. KUNTHUNATH M N,** and **Ms. MEGHANA TADAS** bearing USN number **2KE20CS032, 2KE20CS035, 2KE20CS039, 2KE20CS046** in partial fulfilment for the award of degree of **Bachelor of Engineering** in **VI Semester, Computer Science and Engineering** of **Visvesvaraya Technological University**, Belagavi, during the year**2022-23**. It is certified that all corrections/suggestions indicated for internal assessment have beenincorporated in the report deposited in the department library. The mini project report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for the said degree.

Signature of the Guide	**Signature of the HOD**	**Signature of the Principal**
(Dr. Vishwanath K.)	**(Dr. Yerriswamy T.)**	**(Dr. S. G. Joshi)**

Name of the Examiners	**Signature with Date**
1.	
2.	

ACKNOWLEDGEMENT

The mini project report on **"SIMULATION OF SATELLITE LAUNCHING USING OPENGL"** is the outcome of guidance, moral support and devotion bestowed on us throughout my work. For this we acknowledge and express my profound sense of gratitude and thanks to everybody whohave been a source of inspiration during the project work.

First and foremost, we offer our sincere phrases of thanks with innate humility to our **Principal Dr. S. G. Joshi** who has been a constant source of support and encouragement. We would like to thank our **Dean Academics Dr. Manu. T. M.** for his constant support and guidance. We feel deeply indebted to our **H.O.D. Dr. Yerriswamy T.** for the right help provided from the time of inception till date. We would take this opportunity to acknowledge our **Mr. Shridhar Chini,** who not only stood by us as a source of inspiration, but also dedicated her time for me to enable us present the project on time. We would be failing in endeavor, if we do not thank our **Coordinator Mrs. MALATHI Y.** who has helped us in every aspect of our miniproject work.

Last but not the least, we would like to thank our parents, friends & well-wishers who have helped us in this work.

With Regards,

Mr. KARTIK P HEGADI
Ms. KEERTI S PATIL
Mr. KUNTHUNATH M N
Ms. MEGHANA TADAS

ABSTRACT

The main aim of this Mini Project is to illustrate the concepts of simulation of Satellite launching in OpenGL. A Satellite is an object which has been placed into orbit by human endeavor. Such objects are sometimes called artificial satellites to distinguish them from natural satellites such as the Moon Satellites are used for a large number of purposes. Common types include military and civilian Earth observation satellites, communications satellites, navigation satellites, weather satellites, and research satellites. This pushed the entire network into a 'congestion collapse' where most packets were lost and the resultant throughput was negligible. We have used input devices like the mouse and keyboard to interact with the program. We have also used Solid Cube for forming a complete network setup which helps to understand the concept of Congestion Control very well. To differentiate between objects, we have used different colors for different objects.

Two ways of launching satellite to elliptical orbit and circular orbit are presented for the problems of using space tether system launching a small satellite. Through two mobile launching ways, the apogee height and perigee height of the small satellite are calculated both in elliptic orbit and approximate circular orbit. Then, Simulation analysing the influence of the tether length, slip angle and the disturbance of in-plane angle on the apogee height and perigee height of the small satellite. The dynamics equations of the tether system are established in the orbital coordinate system, then, the initial velocity of small satellite is calculated. In order to establish the motion states of small satellite after cutting off the tether, the orbit equations of small satellite are established in the geocentric orbit coordinate system. After the simulation analysis, the optimal end points are obtained, which provide the theoretical basis for selection of end point of space tether system up deployed.

INDEX

Contents	Page no.
Chapter 1: Introduction	
1.1 Project Definition	01
1.2 Literature Survey	02
1.3 Problem statement	03
1.4 Aims and Objectives	03
1.5 Project Overview	04
Chapter 2: About Computer Graphics	
2.1 Computer Graphics	05
2.2 Application of Computer Graphics	05
2.3 Introduction to OpenGL	06
2.4 OpenGL Utility Tool Kit(GLUT)	08
Chapter 3: Requirement Specification	
3.1 General Description	09
3.2 Software Requirements	10
Chapter 4: Software Design	
4.1 System Design	12
4.2 Detailed Design	13
Chapter 5: Functions Used	
5.1 Various functions used	16
Chapter 6: Implementation	
6.1 Snapshots	19
Chapter 7: Conclusion and Future Scope	21
Chapter 8: References	23

CHAPTER 1

INTRODUCTION

This chapter includes some introduction to our project on simulation of satellite launching.

Welcome to our project on the simulation of satellite launching using OpenGL. In this project, we aim to explore the fascinating world of satellites and demonstrate their functionality through the power of computer graphics using the OpenGL library. A satellite simulator is a device that simulates the satellite of an aircraft, as well as its environment and any events that may occur while it is in the air. OpenGL is a widely-used graphics library that allows developers to create interactive 2D and 3D applications. It provides a set of functions for rendering geometric shapes, textures, and animations on a computer screen. By utilizing OpenGL, we can develop a realistic and immersive simulation of satellite behaviour and movements.

Our project will focus on key aspects of satellite working, including:

1.Orbit Simulation: We will simulate the orbital motion of satellites around celestial bodies such as Earth. By implementing the laws of physics governing orbital mechanics, we can visualize how satellites traverse their designated paths.

2.Communication: Satellites serve as vital communication links for transmitting signals across the globe. We will illustrate how satellites establish and maintain connections with ground-based stations, enabling data transfer and communication between different locations.

3.Data Collection: Satellites are equipped with various instruments and sensors to collect valuable data about our planet, weather patterns, and more. We will showcase the process of data acquisition from different sources and how it is transmitted back to Earth for analysis.

Project **4.Visualization:** Using OpenGL's rendering capabilities, we will create visually appealing representations of satellites, celestial bodies, and the surrounding environment. This will provide an immersive experience, allowing users to observe satellite operations from different perspectives.

1.1 Project Definition

Overview: Our project aims to develop a simulation of satellite launching using the OpenGL graphics library. The simulation will provide an interactive and visually immersive experience that allows users to understand the various aspects of satellite operations, including orbit simulation, communication, and data collection. By leveraging the capabilities of OpenGL, we will create a realistic and engaging environment that showcases the intricate workings of satellites.

Key Features and Components:

1.Orbit Simulation:

- Implement the laws of orbital mechanics to simulate satellite trajectories around celestial bodies.
- Display accurate orbital paths, including elliptical, circular, and inclined orbits.
- Provide user controls to adjust orbital parameters such as altitude, eccentricity, and inclination.

2.Communication:

- Model communication links between satellites and ground-based stations.
- Visualize signal transmission and reception between satellites and stations.
- Demonstrate concepts such as line-of-sight, beam steering, and signal strength.

3.Data Collection:

- Simulate data acquisition from different sources, such as weather sensors, imaging devices, and scientific instruments.
- Showcase the process of data collection and storage on the satellite.
- Illustrate data transmission back to Earth for analysis and dissemination.

4.Visualization:

- Render realistic 3D models of satellites, celestial bodies, and the surrounding environment.
- Incorporate textures, lighting, and shading techniques to enhance visual fidelity.
- Provide interactive camera controls to explore the simulation from different viewpoints.

1.2 Literature Survey

Serious gaming applications, also termed as simulators, are games developed for various purposes such as teaching, learning, or training. Serious gaming concepts are more focused in resolving real-world problems than in providing entertainment. In many areas, training is a risky, costly, and time-consuming procedure. Such fields include pilot training, astronaut training, military training, firefighter training, and medical surgery training. Serious gaming is used to reduce the risk of injury and the cost of training. Health, security, inland defence, communication, and education are just a few of the fields where concepts can be applied. Experiential learning is a method of learning that differs from traditional reading and writing instruction. This approach incorporates a healthy balance of learning, training, and other daily activities, as well as knowledge extraction. Acting and experimenting have been proven to be effective teaching techniques. Experiential learning is the best way to start learning. It elevates one’s ability to observe and make judgments.

This research shows a method for 3-D reconstruction and visualisation of geological materials using the IDL programmable data analysis and visualisation environment. Custom IDL programmes for importing, filtering, and visualising data from precision serial lapping are described by the authors. It is capable of sequentially reading and importing categorised raster images into a voxel array. To provide consistent 3-D reconstruction and noise reduction, the voxel array can be processed by median and/or binary filter programmes with variably shaped filter kernels. A voxel-projecting programme and a surface-rendering application can be used to visualise the filtered results. The IDL slicer application is also demonstrated, which is a 3-D analysis and visualisation tool included in the standard IDL edition.

OpenGL (Open Visuals Library) is a language-independent, cross-platform, hardware-accelerated API for creating 3D (and 2D) graphics. Modern computers have a dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering. OpenGL is the software interface to graphics hardware. In other words, the OpenGL graphic rendering requests from your programmes could be sent to and accelerated by the visual hardware. Three sets of libraries are used in our OpenGL apps: Hundreds of instructions beginning with the letter ”gl” make up the core of OpenGL (GL) (e.g., glColor, glVertex, glTranslate, glRotate). Core OpenGL models an object using a set of geometric primitives such as point, line, and polygon. GLU (OpenGL Utility Library) is a library built on top of OpenGL to give useful utilities (such camera view and projection) and more building models (such as quadric surfaces and polygon tessellation). The prefix ”glu” is used in GLU commands (e.g., gluLookAt, gluPerspective). GLUT (OpenGL Utilities Toolkit): OpenGL is meant to work regardless of the windowing system or operating system. GLUT is required to communicate with the operating system, as well as to provide additional building models (such as sphere and torus)

1.3 Problem statement

The goal of this project is to develop a simulation of the launching of a satellite using the OpenGL graphics library. The simulation should provide an interactive and visually immersive experience that allows users to explore and understand various aspects of satellite operations.

1.4 Aims and Objectives

- The aim of this project is to create an interesting and creative simulation of satellite launching using tools like OpenGL and C++ (g++ compiler).
- Apart from the simulation, this project also deals with providing a beautiful graphical interface between the user and the system.

1.5 Project Overview

Our project focuses on developing a simulation of the working of a satellite using the OpenGL graphics library. The simulation aims to provide users with an immersive and interactive experience to understand the intricacies of satellite operations, including orbit simulation, communication, data collection, and visualization.

Key Features and Components:

1.Orbit Simulation: Implement the principles of orbital mechanics to accurately simulate satellite trajectories around celestial bodies. Display different types of orbits, such as circular, elliptical, and inclined orbits. Allow users to adjust orbital parameters, such as altitude, eccentricity, and inclination, to observe their impact on the satellite's motion.

2.Communication modeling: Develop a communication model that demonstrates how satellites establish connections with ground-based stations. Simulate the transmission and reception of signals, considering factors like line-of-sight, beam steering, and signal strength. Visualize the communication links between satellites and stations.

3.Data Collection Simulation: Simulate the process of data collection on the satellite. Showcase the various sensors, instruments, or payloads onboard the satellite that collect valuable data. Illustrate how data is stored on the satellite and transmitted back to Earth for analysis and utilization.

4.Visualization and Rendering: Utilize the capabilities of OpenGL to create visually appealing and realistic 3D models of satellites, celestial bodies, and the surrounding environment. Apply textures, lighting, and shading techniques to enhance the visual fidelity and create an immersive experience for users.

CHAPTER 2

ABOUT COMPUTER GRAPHICS

2.1 Computer Graphics

Computer Graphics concerns the pictorial synthesis of real and imaginary objects from their computer-based models. It provides one of the most natural means of communication with a computer and permits extensive, high bandwidth user computer interaction. Computer graphics is used today in many different areas like user interfaces. Plotting in business, science, office automation, electronic publishing and computer aided drafting and design, simulation and animation for scientific visualization and entertainment, cartography etc.

Graphics are visual presentation on some surface, such as well, canvas, computer screen etc. Examples are photography, line art, graphs, engineering drawings, maps, typography etc. Graphics always combine text, illusion and color. Graphics design may consist of the deliberation selection creation or arrangement of typography alone etc. Clarity or effective communication may be the objective, association with other cultural elements may be sought or merely the creation of a distinctive style.

2.2 Application of Computer Graphics

- Display of information.
- Design for animated games.
- Simulation and animation.
- User interface.
- Display of information in Industries and Business.

Design:

Professional such as engineering and architecture are concerned with design. The use of interactive graphics tools in computer aided design pervades fields. Including as architecture, mechanical engineering, the design of very large-scale integrated circuits and the creation of character for animations.

Simulation and Animation:

Once graphics systems evolved to be capable of generating sophisticated images in real time, engineers and researches began to use them simulators. One of the most important uses has been in the training of pilots. Graphical flights simulation has proved to increase safety and to reduce training expenses.

User interfaces:

Interaction with computers has become dominated by a visual paradigm that includes windows, icons, menus and plotting device such as mouse. User interfaces demonstrate the variety of the available in high level modelling packages and the interactive devices the user can employ in modelling geometric objects.

2.3 Introduction to OpenGL

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer.

These objects are described as sequences of vertices or pixels.

OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

OpenGL Fundamentals

This section explains some of the concepts inherent in OpenGL.

Primitives and Commands

OpenGL draws primitives—points, line segments, or polygons—subject to several selectable modes.

You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set. Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls.

Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The type of clipping depends on which primitive the group of vertices represents.

Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state-querying commands return data that's consistent with complete execution of all previously issued OpenGL commands.

Basic OpenGL Operation

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data.

In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

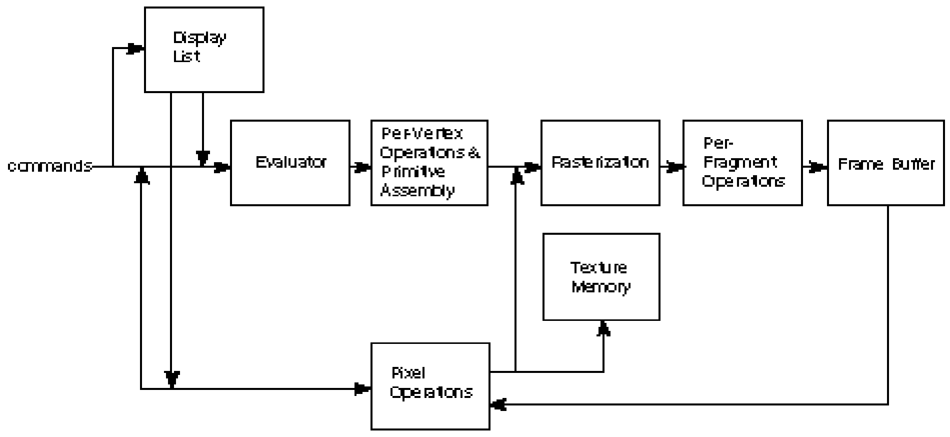


Figure. OpenGL Block Diagram

As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing at a later time.

Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon.

Each fragment so produced is fed into the last stage,

per-fragment operations, which performs the final operations on the data before it's stored as pixels in the frame buffer. These operations include conditional updates to the frame buffer based on incoming and previously stored z-value s (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

OpenGL is competing application programming interfaces (APIs), which can be used in applications to render 2D and 3D computer graphics, taking advantage of hardware acceleration when available. Modern graphics processing unit (GPUs) may implement a particular version of the APIs.

OpenGL is a specification implemented in the C language, though it can use other programming languages. It is built on concept of a state machine, though more recent OpenGL version have transformed it into much more object-based system. As an API, OpenGL depends on no particular language feature, and can be made callable from almost any programming language with the proper bindings. Such bindings exist for Ada, BASIC (Blitz Max (often used to Program Games), Pure Basic, Visual Basic), C#, Delphi, Fortran, Haskell, Java, Lisp, Lua, Pascal, Perl, Python, and Ruby.

Graphics provide one of the most natural means of communicating with a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and efficiently.

2.4 OpenGL Utility Tool Kit (GLUT)

As you know, OpenGL contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse.

OpenGL drawing commands are limited to those that generate simple geometric primitive, GLUT includes several routines that creates more complicated three-dimensional objects such as a sphere, a torus and a teapot.

The OpenGL Utility Library, GLU also has quadrics routines that create some of the same three-dimensional objects as GLUT, such as a sphere, cylinder or cone.

The OpenGL Utility Toolkit (GLUT) is a programming interface with ANSIC and FORTAN bindings for writings window system independent OpenGL programs. The toolkit supports the following functionality:

Multiple windows for OpenGL rendering.

- Call-back driven event processing.
- Sophisticated input devices.
- An "idle" routine and timers.
- A simple, cascading pop-up menu facility.
- Utility routines to generate various solid and wire frame objects.

CHAPTER-3

REQUIREMENT SPECIFICATION

3.1 GENERAL DESCRIPTION

Satellite simulation plays a crucial role in the design, development, and testing of satellite systems. It allows engineers and scientists to model the behavior of satellite under various conditions and simulate their performance before actual implementation. This report provides a general description of satellite simulation, outlining its purpose, components, and key considerations.

Purpose of Satellite Simulation:

Satellite simulation serves multiple purposes throughout the life cycle of a Satellite, including:

- Design Optimization: Simulations aid in optimizing the satellite 's design by predicting its performance characteristics, such as trajectory, stability, and aerodynamics. Engineers can analyze different configurations, materials, and propulsion systems to enhance efficiency and reliability.
- Performance Evaluation: Simulations enable the evaluation of a satellite 's performance under different scenarios, such as varvina pavloads, atmospheric conditions, and mission requirements. This helps assess miss stop generating insumption, and payload capacity.
- Safety and Reliability: Simulations assist in identifying potential failure modes, vulnerabilities, and risks associated with the satellite's components, systems, and operations.
- This helps in improving safety measures and enhancing reliability.
- Training and Education: Satellite simulations provide a valuable tool for training astronauts, mission control teams, and engineers. They allow individuals to familiarize themselves with satellite systems, practice launch sequences, and gain experience in managing complex missions.

Components of Satellite Simulation:

Satellite simulations typically consist of the following key components:

- Mathematical Models: Mathematical models represent the physics and dynamics of the satellite's behavior. These models incorporate equations that govern motion, fluid dynamics, propulsion, and other relevant factors. They may range from simple analytical formulas to complex computational fluid dynamics (CFD) models.
- Software Tools: Specialized software tools are used to implement the mathematical models, simulate satellite behavior, and visualize the results. Examples include systems like MATLAB/Simulink, ANSYS Fluent, OpenSatellite, and STK (Systems Tool Kit). These tools provide a user-friendly interface for confinurina simulation parameters, running simulations, stop generating and analyzing the output.

- Input Parameters: Satellite simulations require various input parameters, such as satellite dimensions, mass properties, propulsion characteristics, atmospheric conditions, and mission profiles. Accurate input values are essential for obtaining reliable simulation results.
- Output Analysis: Simulation outputs include information about satellite performance, such as trajectory, velocity, acceleration, aerodynamic forces, and stability. These outputs are analyzed to assess the satellite's behavior, evaluate its performance metrics, and identify areas for improvement.

Key Considerations in Satellite Simulation:

When conducting satellite simulations, several considerations must be taken into account:

- Validation: It is important to validate the simulation models against real-world data or experimental results to ensure their accuracy and reliability. This may involve comparing simulation outputs with actual satellite or ground tests.
- Uncertainty Analysis: Simulations should account for uncertainties associated with input parameters and model assumptions. Performing sensitivity analyses and Monte Carlo simulations can help quantify the impact of uncertainties on the results.
- Computational Resources: Satellite simulations can be computationally intensive, especially for complex models or high-fidelity simulations. Sufficient computational resources, such as powerful computers or cloud computing platforms, are required to run simulations efficiently.
- Iterative Design Process: Satellite simulation is often an iterative process, with engineers refining the design based on simulation results. Multiple simulations may be conducted to explore different design options, optimize performance, and address potential issues.
- Satellite simulation plays a vital role in the development and evaluation of satellites. It provides a cost-effective and efficient means to predict and analyze satellite behavior, optimize designs, and ensure safety and reliability. By leveraging mathematical models and software tools, engineers can simulate various scenarios, assess performance metrics, and improve the overall effectiveness of satellite systems.

3.2 SOFTWARE REQUIREMENTS

The simulation of satellite typically requires specialized software tools that can accurately model and analyze various aspects of satellite behavior. Here are some key software requirements for satellite simulation:

1. **Numerical Analysis and Modeling Software:**

• MATLAB/Simulink: MATLAB is widely used for numerical analysis and modeling in various engineering disciplines, including satellite simulation. Simulink, a graphical programming environment within MATLAB, is often used for system-level modeling and simulation of satellite.

2. **Computational Fluid Dynamics (CFD) Software:**

• ANSYS Fluent: ANSYS Fluent is a popular CFD software used for simulating fluid flow and heat transfer in satellite engines and aerodynamic analysis. It can model complex flow phenomena, such as combustion, turbulence, and multiphase flow.

3. **Trajectory and Orbit Simulation Software:**

• STK (Systems Tool Kit): STK is a comprehensive software package used for modeling and analyzing satellite and satellite trajectories. It enables the simulation of launch trajectories, orbital mechanics, and mission planning.

4. **Propulsion System Simulation Software:**

• CEARUN (Chemical Equilibrium with Applications): CEARUN is a software package specifically designed for satellite propulsion system analysis. It can calculate equilibrium compositions and thermodynamic properties of satellite propellants, combustion products, and nozzle flows.

5. **Structural Analysis Software:**

Finite Element Analysis (FEA) software: FEA software, such as ANSYS Mechanical or Abaqus, is utilized to perform structural analysis of satellite. It helps predict structural behavior, stress distribution, and failure modes under various loading conditions.

6. **Visualization and Post-processing Software:**

• Tecplot: Tecplot is a visualization and post-processing software used to analyze and present simulation results. It enables the creation of 2D and 3D plots, streamline visualizations, and animations to visualize satellite performance and behavior.

7. **Control System Simulation Software:**

• Simulink Control Design: Simulink Control Design, an extension of MATLAB/Simulink, is used for simulating and analyzing control systems. It allows engineers to model and evaluate the performance of satellite control systems.

8. **Multi-Body Dynamics Simulation Software:**

• Adams: Adams is a multi-body dynamics simulation software commonly used for simulating complex mechanical systems, including satellite. It can model the dynamic interactions between various satellite components, such as stages, thrusters, and payloads.

It's worth noting that specific software requirements may vary depending on the scope, complexity, and specific needs of the satellite simulation project. Engineers and researchers may also develop custom software tools or scripts tailored to their specific simulation requirements.

CHAPTER-4

SOFTWARE DESIGN

4.1 SYSTEM DESIGN

The system design for simulating a satellite involves the overall architecture and components that work together to accurately model and analyze the satellite's behavior. Here is a high-level overview of the system design for a satellite simulation:

1. Mathematical Models:

Satellite Dynamics Model: This model describes the motion and behavior of the satellite during its flight. It incorporates equations of motion, including translational and rotational dynamics, aerodynamic forces, and gravitational effects.

• **Propulsion Model:** The propulsion model simulates the behavior of satellite engines, including thrust generation, fuel consumption, and exhaust characteristics. It may include combustion models, nozzle performance, and specific impulse calculations.

• **Aerodynamics Model:** The aerodynamics model captures the interaction between the satellite and the surrounding air or fluid. It considers factors such as drag, lift, stability, and control surfaces.

• **Environment Model:** The environment model incorporates atmospheric conditions, including pressure, temperature, density, and wind effects, which impact the satellite's flight characteristics.

2. Simulation Software:

• **Numerical Analysis and Simulation Tools:** Software tools like MATLAB/Simulink, Python, or custom programming languages are used to implement the mathematical models and perform numerical simulations. These tools provide a platform to define equations, set initial conditions, and simulate the satellite's behavior over time.

• **Integration and Data Exchange:** Simulation software allows for the integration of various models and data exchange between them. This enables the interaction between the satellite dynamics, propulsion, aerodynamics, and environment models, providing a comprehensive simulation framework.

3. Input Parameters and Configuration:

• **Satellite Design Parameters:** Key design parameters, such as satellite dimensions, mass properties, materials, and payload characteristics, are defined as input parameters for the simulation.

• **Mission Profiles:** Simulation input includes mission-specific parameters, such as desired altitude, velocity, trajectory, and any maneuver requirements.

- **Atmospheric Conditions:** Parameters related to atmospheric conditions, such as air density, temperature, and wind profiles, are provided as input to accurately model the satellite 's interaction with the environment.

4. Simulation Execution:

- **Initialization:** The simulation is initialized by setting the initial conditions, including the satellite 's position, velocity, attitude, and rotational rates.
- **Time Integration:** The simulation progresses by numerically integrating the equations of motion and other models over time. The simulation time step size is determined based on stability and accuracy requirements.
- **Iterative Simulation:** Depending on the complexity of the simulation multiple iterations or time steps may be performed to update the satellite 's position, velocity, and other parameters accurately.

5. Output Analysis and Visualization:

• Simulation Output: The simulation produces various output data, including the satellite’s trajectory, velocity, acceleration, forces, control inputs, and other relevant parameters.
These outputs capture the satellite 's performance throughout the simulation.

- **Post-processing and Visualization:** Post-processing tools like MATLAB, Tecplot, or custom software can be used to analyze and visualize simulation results. Plots, graphs, animations, and 3D visualizations aid in understanding the satellite 's behavior, performance, and characteristios.

• **Model Validation:** Simulation results should be validated against real-world data, experimental results, or previous successful missions to ensure accuracy and reliability.

•**Iterative Improvement:** Simulation results are analyzed to identify areas for improvement in the satellite design, performance, or mission objectives. Iterative design cycles can be initiated to refine the simulation models, input parameters, or system configurations.

The system design of a satellite simulation should be flexible, modular, and scalable to accommodate various satellite configurations, mission profiles, and simulation requirements.

It should allow for easy integration of new models, software tools, or data sources to enhance the accuracy and capabilities of the simulation system.

4.2 DETAILED DESIGN

Designing a detailed simulation of a satellite involves considering various aspects and components that accurately model the satellite's behavior. Here is a more detailed design breakdown for a satellite simulation:

1. Satellite Dynamics Model:

- Translational Dynamics: Implement the equations of motion to simulate the satellite 's translational motion, considering forces such as thrust, drag, gravity, and any additional external forces.
- Rotational Dynamics: Model the satellite 's rotational motion by incorporating equations of angular momentum, torque, and moments of inertia. Consider factors such as aerodynamic forces, control surfaces, and gyroscopic effects.
- Attitude Control: Include control algorithms and actuators to simulate the satellite's attitude control system. This involves modeling the behavior of reaction wheels, thrusters, or other control mechanisms to adjust the satellite 's orientation.

2. Propulsion Model:

- Thrust Generation: Develop a propulsion model that calculates the thrust generated by the satellite engine based on its design, propellant properties, combustion characteristics, and nozzle performance.
- Fuel Consumption: Simulate the fuel consumption rate based on the thrust level, specific impulse, and burn rate of the satellite's propellant.
- Exhaust Plume Effects: Account for the effects of the exhaust plume, including temperature, pressure, and gas dynamics, which may impact the satellite's performance and aerodynamics.

3. Aerodynamics Model:

- Drag and Lift: Incorporate aerodynamic models to calculate the drag and lift forces acting on the satellite. Consider factors such as the satellite's shape, cross-sectional area, air density, and velocity to determine the aerodynamic coefficients.
- Stability Analysis: Analyze the satellite's stability by considering its center of pressure, center of gravity, and moment of inertia. Determine stability margins aild evaluate the satellite's ability to maintain desired flight attitudes.
- Control Surfaces: Model the behavior of control surfaces, such as fins, gimbaled engines, or reaction control systems, to simulate their impact on stability, control, and maneuverability.

4. Environment Model:

- Atmospheric Conditions: Incorporate atmospheric models to simulate air density, temperature, pressure, and wind profiles at different altitudes. Consider variations in atmospheric conditions during the satellite's ascent or descent phases.
- Gravity Model: Account for the gravitational forces acting on the satellite based on the satellite's position, altitude, and Earth's gravitational field.
- Magnetic Field: If necessary, include a model of the Earth's magnetic field to simulate its effects on guidance and navigation systems.

5. Simulation Software and Integration:

- Numerical Methods: Choose appropriate numerical integration methods, such as Runge-Kutta or Euler methods, to solve the differential equations representing) satellite dynamics and other models.
- Software Tools: Utilize simulation software platforms, such as MATLAB/Simulink, Python, or custom programming languages, to implement and integrate the various models and algorithms.
- Data Exchange: Enable data exchange and communication between different models to ensure proper interaction and synchronization during the simulation.

6. Input Parameters and Configuration:

- Satellite Design: Define input parameters related to the satellite's physical characteristics, such as mass, dimensions, moment of inertia, and center of gravity.
- Propulsion Parameters: Specify input parameters related to the satellite engine, such as specific impulse, thrust profile, and propellant properties.
- Mission Profiles: Set input parameters for the mission requirements, including desired trajectory, target altitude, payload weight, and any specific maneuvers.

7. Simulation Execution:

- Initialization: Set initial conditions for the simulation, including the satellite's position, velocity, attitude, and rotational rates.
- Time Integration: Perform numerical integration using the selected integration method to advance the satellite simulation.

CHAPTER 5
FUNCTIONS USED

This program is implemented using various openGL functions which are shown below.

5.1 Various functions used in this program:

- **glutInit()** : interaction between the windowing system and OPENGL is initiated.
- **glutInitDisplayMode()** : used when double buffering is required and depth information is required.
- **glutCreateWindow()** : this opens the OPENGL window and displays the title at top of the window.
- **glutInitWindowSize()** : specifies the size of the window.
- **glutInitWindowPosition()** : specifies the position of the window in screen co-ordinates.
- **glutKeyboardFunc()** : handles normal ascii symbols.
- **glutSpecialFunc()** : handles special keyboard keys.
- **glutReshapeFunc()** : sets up the callback function for reshaping the window.
- **glutIdleFunc()** : this handles the processing of the background.
- **glutDisplayFunc()** : Sets the display callback function, which is responsible for rendering the graphics in the window. This function is called whenever the window needs to be redrawn, such as when it is first created or when
- **glutMainLoop()** : Enters the GLUT event processing loop. This function continuously processes events such as user input, window resizing, and rendering updates. It remains in the loop until the application is terminated.
- **glViewport()** : Sets the viewport, which defines the portion of the window where the graphics will be displayed. It maps the normalized device coordinates to window coordinates.
- **glVertex3fv()** : Specifies a vertex position for rendering geometric primitives in OpenGL. This function takes a pointer to an array of three floating-point values representing the x, y, and z coordinates of the vertex.
- **glColor3fv()** : Specifies the current color for rendering primitives. This function takes a pointer to an array of three floating-point values representing the red, green, and blue components of the colour.
- **glFlush()** : Forces any pending OpenGL commands to be executed immediately. It ensures that the rendering commands are processed and displayed on the screen.

- **glutPostRedisplay()** : Marks the current window as needing to be redisplayed. It requests a call to the display callback function to redraw the window in the next iteration of the event loop.
- **glMatrixMode()** : Sets the current matrix mode, which determines whether subsequent transformations will affect the model view matrix, projection matrix, or other matrices used in OpenGL.
- **glLoadIdentity()** : Replaces the current matrix with the identity matrix. It initializes the current matrix to an identity state, effectively resetting any previous transformations.
- **glTranslatef()** : Applies a translation to the current matrix, which moves the coordinate system in the specified x, y, and z directions.
- **glRotatef()** : used to rotate an object through a specified rotation angle.

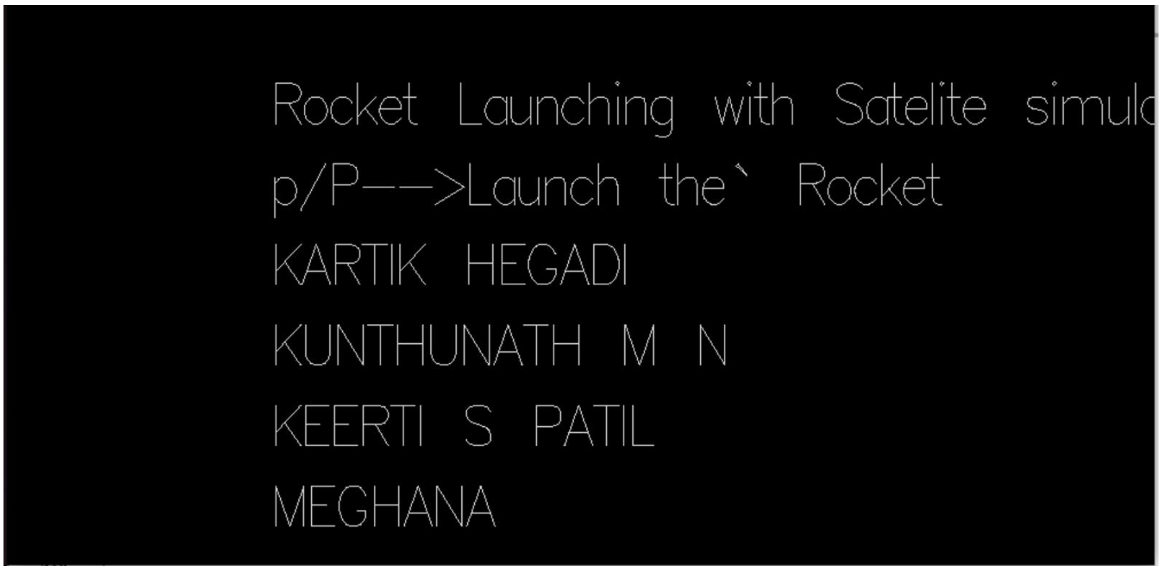
CHAPTER 6

IMPLEMENTATION

The successful implementation of the "Satellite Launch" program utilizing various OpenGL functions demonstrates an effective approach to creating interactive and visually appealing simulations.
To initiate the interaction between the windowing system and OpenGL, the program utilizes the "glutInit()" function. This crucial step establishes the necessary environment and context for OpenGL operations. Additionally, the "glutInitDisplayMode()" function is utilized to specify display modes such as double buffering and depth information, enabling smoother rendering and enhanced 3D depth perception.
Creating the OpenGL window and displaying the title is achieved through the "glutCreateWindow()" function. This function allows users to easily identify the purpose or content of the program. By specifying the desired window size and position using functions like "glutInitWindowSize()" and "glutInitWindowPosition()", the program ensures an optimal viewing experience on different screen resolutions.
User input is efficiently handled through functions like "glutKeyboardFunc()" and "glutSpecialFunc()". These functions enable the program to respond to both normal ASCII symbols and special keyboard keys, allowing for intuitive interaction and control.
To accommodate window resizing, the "glutReshapeFunc()" function sets up a callback function that adjusts the viewport and projection matrix, ensuring that the rendered scene remains properly scaled and aligned. The "glutIdleFunc()" function is employed to handle background processing, facilitating tasks such as animation updates or background computations when the program is idle.
The heart of the rendering process lies in the "glutDisplayFunc()" function, which utilizes a callback function to redraw the window and render the scene. By defining the appropriate viewport using "glViewport()", and specifying points or vertices in three dimensions with "glVertex3fv()", the program accurately represents the satellite launch and satellite deployment.
Various transformation functions such as "glLoadIdentity()", "glTranslatef()", and "glRotatef()" are utilized to manipulate the position and orientation of objects, facilitating smooth animations and realistic movement.
Finally, the program enters the main loop with "glutMainLoop()", continuously processing events, handling user input, and triggering automatic redraws with "glutPostRedisplay()". This ensures a seamless and interactive experience for users, allowing them to engage with the simulation until it is manually terminated.

In conclusion, the effective utilization of the aforementioned OpenGL functions in the "Satellite Launch" program showcases the power and versatility of OpenGL for creating immersive simulations. By carefully employing these functions, the program achieves an engaging and visually captivating representation of a satellite launch and satellite deployment, providing users with an informative and interactive experience.

6.1 SNAPSHOTS



```
Rocket Launching with Satelite simul
p/P-->Launch the` Rocket
KARTIK HEGADI
KUNTHUNATH M N
KEERTI S PATIL
MEGHANA
```

Fig 6.1: Menu for Simulation of Satellite Launching

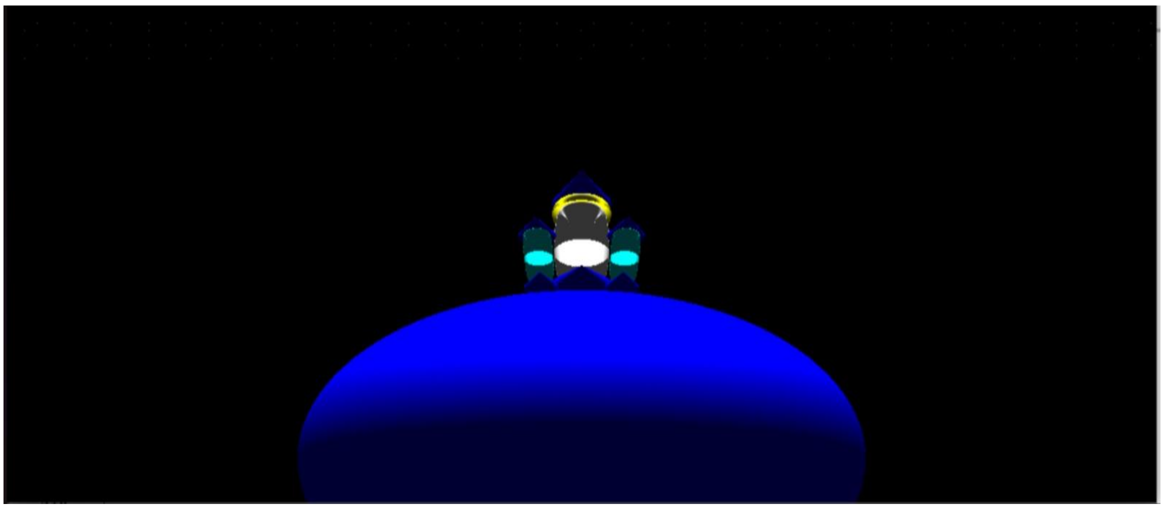


Fig 6.2: Launching of Satellite

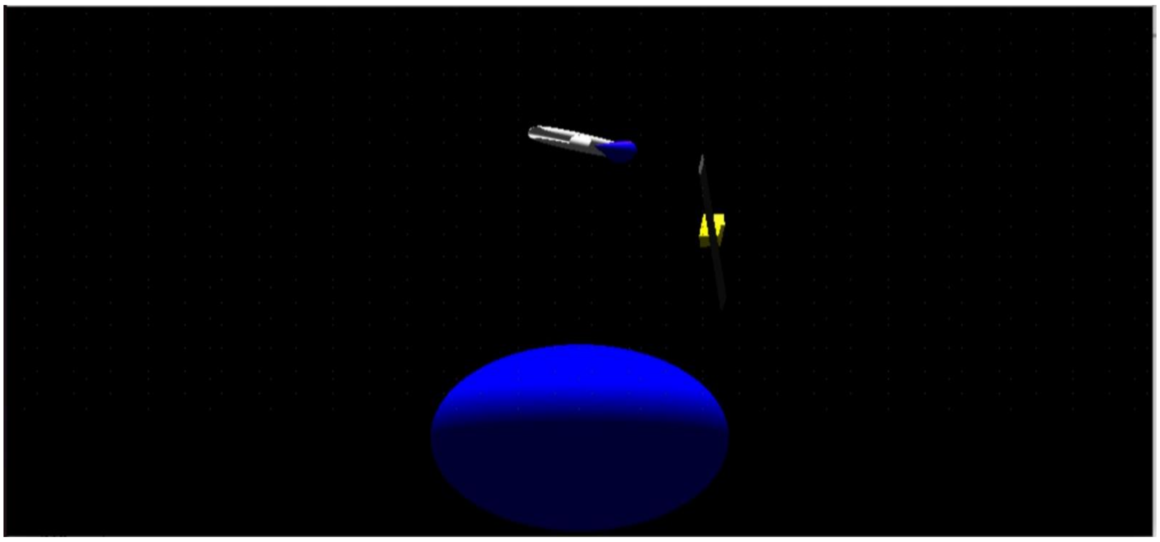


Fig 6.3: Image of Satellite

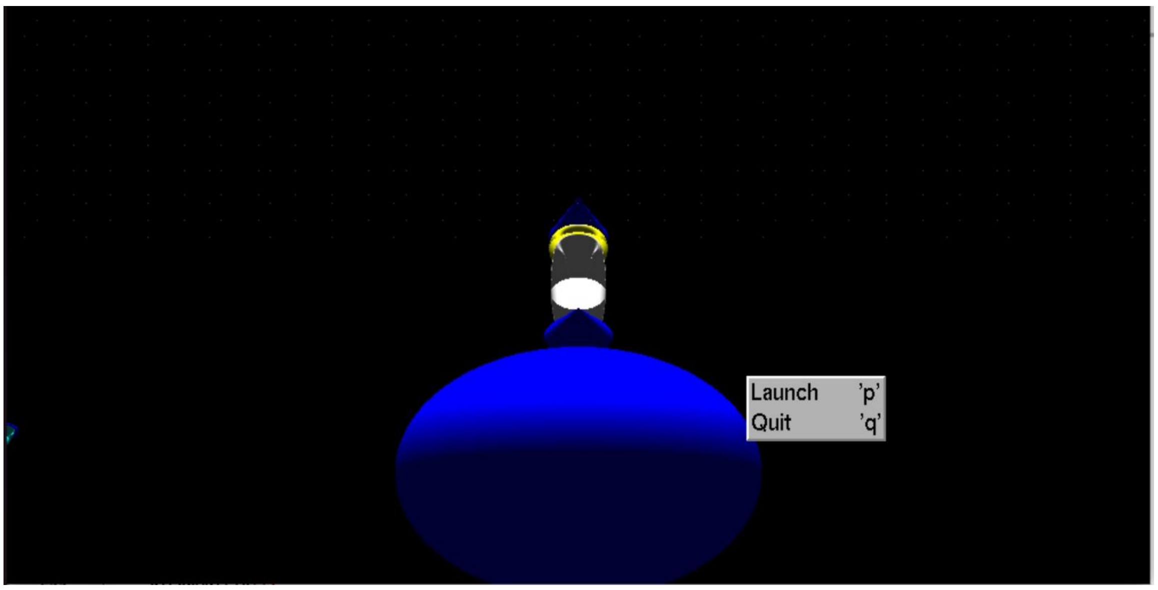


Fig 6.4: Instruction for Satellite Launching

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

The project titled "Satellite Launch" has successfully accomplished the task of simulating a satellite launch and the subsequent deployment of a satellite into orbit. Through the utilization of OpenGL and pre-built objects, this program provides a comprehensive and visually engaging representation of the entire process.

The simulation begins by showcasing the various stages of a satellite launch, including ignition, liftoff, and ascent. Users can witness the realistic depiction of the satellite's propulsion system, with flames and exhaust gases billowing out as it propels itself off the launchpad. The visual effects employed in the simulation create an immersive experience, capturing the excitement and power associated with a real satellite launch.

As the satellite reaches the designated altitude, the simulation transitions smoothly into the satellite deployment phase. Users can observe the careful separation of the satellite from the satellite's upper stage, mimicking the precise manoeuvres required to place the satellite into its intended orbit. The simulation accurately portrays the intricate orbital mechanics involved, showcasing the satellite's trajectory as it gracefully enters its designated orbit around the Earth.

Having successfully achieved the objectives of simulating a satellite launch and satellite deployment, we can confidently conclude that this program utilizing OpenGL and pre-built objects is a resounding success. The attention to detail, realistic visuals, and accurate representation of the entire process make it an effective educational tool for understanding the complexities of space missions.

With its completion, the program is now ready to be demonstrated to a wide audience, including students, space enthusiasts, and professionals in the aerospace industry. It serves as a valuable resource for learning and appreciating the intricacies of satellite launches and satellite operations, providing a captivating experience that bridges the gap between theory and practical application.

Additionally, the "Satellite Launch" project demonstrates the incorporation of key elements that enhance the overall simulation experience. The use of sound effects, such as the roaring sound of satellite engines and the beeping of telemetry data, adds an auditory dimension that immerses users further into the simulation. These realistic audio cues contribute to a more authentic representation of a satellite launch, creating a multisensory experience.

Moreover, the program includes informative visual overlays and annotations that provide real-time data and relevant information throughout the simulation. Users can access details about the satellite's speed, altitude, and fuel levels, as well as information about the satellite's orbit parameters and mission objectives. These overlays serve as educational aids, fostering a deeper understanding of the technical aspects and scientific principles involved in space missions.

Furthermore, the project's successful completion paves the way for future advancements and expansion. There is ample potential for incorporating additional features, such as simulating different types of satellites, exploring various launch sites, and incorporating more complex orbital maneuvers. By continuously refining and expanding the program, it can serve as a platform for experimentation and learning in the field of aerospace engineering and space exploration.

The "Satellite Launch" simulation also holds promise for collaborative learning environments. It can be utilized in classrooms, workshops, and training sessions to engage participants in group activities and discussions. Users can analyze the various stages of the satellite launch together, exchange insights, and explore different scenarios, fostering a collaborative and interactive learning experience.

In conclusion, the successful implementation of the "Satellite Launch" program utilizing OpenGL and pre-built objects provides an impressive simulation of a satellite launch and satellite deployment. With its realistic visuals, sound effects, informative overlays, and potential for future expansion, this program stands as a valuable tool for both educational and entertainment purposes, captivating audiences and fostering a deeper understanding of the fascinating world of space exploration.

CHAPTER 8

REFERENCES

[1]Luís M. B. C. Campos and Paulo J. S. Gil, On Four New Methods of Analytical Calculation of Satellite Trajectories, Aerospace J. 2018, 5, 88, pp. 1-32.

[2] George Buchanan et al., The Development of Satellitery Capability in New Zealand—World Record Satellite and First of Its Kind Satellitery Course, Aerospace J. 2015, 2, pp. 92-117.

[3] Hoani Bryson et al., Vertical Wind Tunnel for Prediction of Satellite Flight Dynamics, Aerospace J. 2016, 3, 10, pp. 1-27.

[4] Stamminger, A. STERN—A satellite programme for German students. In Proceedings of the 21st ESA Symposium on European Satellite and Balloon Programmes and Related Research, Thun, Switzerland, 9–13 June 2013.

[5] Rojas, J.I.; Prats, X.; Montiaur, A.; Garcia-Berro, E. Model satellite workshop: A problem-based learning experience for engineering students. Int. J. Emerg. Technol. Learn. 2008, 3, 70–77.

[6] Heath, M.T.; Dick, W.A. Virtual satellitery: Satellite science meets computer science. Computat. Sci. Eng. IEEE 1998, 5, 16–26.

[7] Cannon, R.L. Model satellitery as a teaching aid in science. Sch. Sci. Math 2010, 74, 471–475.
[8] Jayaram, S.; Boyer, L.; George, J.; Ravindra, K.; Mitchell, K. Project-based introduction to aerospace engineering course: A model satellite. Acta Astronaut. 2010, 66, 1525–1533.

[9] Box, S.; Bishop, C.M.; Hunt, H. Stochastic six-degree-of-freedom flight simulator for passively controlled high-power satellites. J. Aerosp. Eng. 2011, 24, 31–45.