INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Eduardo Moura Zindani

EDUCATIONAL ORBIT SIMULATION WITH GENERATIVE AI AGENTIC WORKFLOW AND VIRTUAL REALITY VISUALISATION

Final Paper
2025

Course of Aerospace Engineering

CDU ???.??

Eduardo Moura Zindani

EDUCATIONAL ORBIT SIMULATION WITH GENERATIVE AI AGENTIC WORKFLOW AND VIRTUAL REALITY VISUALISATION

Advisor

Prof. Dr. Christopher Shneider Cerqueira (ITA)

AEROSPACE ENGINEERING

São José dos Campos
Instituto Tecnológico de Aeronáutica

2025

Cataloging-in Publication Data
Documentation and Information Division

Zindani, Eduardo Moura
Educational Orbit Simulation with Generative AI Agentic Workflow and Virtual Reality Visualisation / Eduardo Moura Zindani.
São José dos Campos, 2025.
68p.

Final paper (Undergraduation study) – Course of Aerospace Engineering– Instituto Tecnológico de Aeronáutica, 2025. Advisor: Prof. Dr. Christopher Shneider Cerqueira.

1. AI. 2. VR. 3. Orbit. I. Instituto Tecnológico de Aeronáutica. II. Educational Orbit Simulation with Generative AI Agentic Workflow and Virtual Reality Visualisation.

BIBLIOGRAPHIC REFERENCE

ZINDANI, Eduardo Moura. **Educational Orbit Simulation with Generative AI Agentic Workflow and Virtual Reality Visualisation**. 2025. 68p. Final paper (Undergraduation study) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

Eduardo Moura Zindani
Rua H8B, Ap. 235
12.228-461 – São José dos Campos–SP

EDUCATIONAL ORBIT SIMULATION WITH GENERATIVE AI AGENTIC WORKFLOW AND VIRTUAL REALITY VISUALISATION

This publication was accepted like Final Work of Undergraduation Study

---

Eduardo Moura Zindani

Author

Christopher Shneider Cerqueira (ITA)

Advisor

Prof. Dra. Maísa de Oliveria Terra
Course Coordinator of Aerospace Engineering

São José dos Campos: June 20, 2025.

...

Acknowledgments

...

"If I have seen farther than others,
it is because I stood on the shoulders of giants."
— Sir Isaac Newton

Resumo

Métodos educacionais tradicionais frequentemente encontram dificuldades para transmitir conceitos complexos, espaciais e dinâmicos como os da mecânica orbital. A recente convergência entre a Realidade Mista (RM) de consumo e os sofisticados agentes de Inteligência Artificial (IA) Generativa apresenta uma oportunidade para criar um novo paradigma de interfaces de aprendizagem intuitivas e experienciais. Este trabalho detalha o projeto, desenvolvimento e demonstração de uma plataforma educacional interativa para a exploração dos princípios da mecânica orbital. O objetivo principal do sistema é conectar a teoria de leis físicas abstratas à compreensão intuitiva, permitindo que os usuários aprendam por meio da interação corporificada. A metodologia é centrada em uma arquitetura modular que integra dois componentes principais: (1) um "cérebro" agente generativo, impulsionado por Modelos de Linguagem Abrangentes, que interpreta comandos em linguagem natural e atua como um guia educacional especializado; e (2) um "mundo" de simulação em tempo real e visualização imersiva em realidade mista, construído no motor Unity para o Meta Quest 3, que renderiza trajetórias orbitais fisicamente precisas em um espaço tridimensional onde os usuários vivenciam a mecânica orbital de dentro. A plataforma facilita um ciclo de interação multimodal contínuo, onde os comandos de voz do usuário são capturados, processados pelo agente para alterar os parâmetros da simulação e refletidos na visualização imersiva em RV com feedback auditivo conversacional. Este trabalho entrega um protótipo funcional que demonstra uma nova abordagem para a educação científica, transformando dados abstratos em uma experiência manipulável e conversacional para promover uma aprendizagem exploratória e profundamente engajadora. A plataforma é disponibilizada como software de código aberto para permitir validação, adaptação e extensão pela comunidade para diversos contextos educacionais.

Abstract

Traditional educational methods often struggle to convey complex, spatial, and dynamic concepts such as those found in orbital mechanics. The recent convergence of consumer-grade Mixed Reality (MR) and sophisticated Generative AI agents presents an opportunity to create a new paradigm for intuitive and experiential learning interfaces. This paper details the design, development, and demonstration of an interactive educational platform for exploring the principles of orbital mechanics. The system's primary objective is to bridge the gap between abstract physical laws and intuitive comprehension by enabling users to learn through embodied interaction. The methodology is centered on a modular architecture that integrates two core components: (1) a generative agent "brain," powered by Large Language Models, which interprets natural language commands and acts as an expert educational guide; and (2) a real-time simulation and immersive mixed reality visualization "world," built in the Unity engine for the Meta Quest 3, which renders physically accurate orbital trajectories in a three-dimensional space where users experience orbital mechanics from within. The platform facilitates a seamless multimodal interaction loop where a user's voice commands are captured, processed by the agent to alter simulation parameters, and reflected in the immersive VR visualization with conversational auditory feedback. This work delivers a functional prototype that demonstrates a novel approach to science education, transforming abstract data into a manipulable, conversational experience to foster exploratory and deeply engaging learning. The platform is released as open-source software to enable community validation, adaptation, and extension for diverse educational contexts.

List of Figures

List of Tables

Contents

1 Introduction

1.1 Organisation

This work is organised into three main chapters, each addressing a distinct aspect of the project. The breakdown is as follows:

- **Chapter 1: Introduction.** This chapter sets the stage for the research and development.
  - *Motivation* (§1.2): Presents the core argument that the convergence of immersive reality technologies—particularly Virtual Reality—and generative AI enables a new, more intuitive paradigm for educational interfaces.
  - *Objectives* (§1.3): Defines the project's specific, actionable goals, centered on the development and demonstration of an interactive, agent-guided simulation platform.
- **Chapter 2: Literature Review.** This chapter provides the theoretical and technical foundation for the work by reviewing three key domains.
  - *Augmented and Virtual Reality* (§2.1): Reviews the evolution of immersive hardware and software ecosystems and establishes their pedagogical value for spatial learning.
  - *Generative Agents* (§2.2): Defines the architecture of modern LLM-powered agents, detailing their ability to use planning, memory, and external tools to reason through and execute complex tasks.
  - *Orbital Mechanics* (§2.3): Outlines the fundamental physics of celestial motion, including the two-body problem, classical orbital elements, and impulsive maneuvers, which form the mathematical basis for the simulation.
- **Chapter 3: Methodology.** This chapter details the practical design, implementation, and demonstration strategy of the project.

- *System Architecture and Data Flow* (§3.2): Describes the end-to-end fluxogram of the system, illustrating how user voice input is captured, processed by the agent, and rendered in the virtual reality simulation in a continuous loop.
- *Core Component Implementation* (§3.4): Details the specific development plan and tools for the two primary modules: the Generative Agent ("Brain") and the Simulation and AR Visualisation ("World").
- *Demonstration and Release Strategy* (§3.6): Outlines the approach for showcasing the platform's capabilities through a demonstration video and releasing it as open-source software to enable community validation, adaptation, and extension.

1.2 Motivation

For decades, popular media and speculative fiction have envisioned futuristic interfaces for exploration and control, from holographic command centers to immersive planetary navigation tools. Films such as *Minority Report* (2002) and *Iron Man* (2008) popularized visions of humans interacting with vast information systems through gestures, speech, and spatial manipulation. These visions were once confined to science fiction, but today, the convergence of immersive reality technologies—particularly Virtual Reality (VR)—and Artificial Intelligence (AI) is bringing such interfaces into the realm of technological feasibility.

In particular, the past few years have seen rapid advances in consumer-grade immersive hardware, particularly in virtual reality. Devices like the Meta Quest and Apple Vision Pro represent significant milestones in accessibility and visual fidelity, enabling immersive environments that are no longer confined to laboratory research or elite applications. The implications for interface design, interaction paradigms, and knowledge acquisition are profound. VR is no longer a speculative technology; it is present, evolving, and increasingly democratized.

Concurrently, the emergence of generative AI and language-based agents has introduced a paradigm shift in how humans interact with complex systems. Large Language Models (LLMs), such as those powering conversational agents, can now interpret natural language, generate multimodal content, and coordinate sequences of actions across software environments. This represents a departure from deterministic, rule-based systems toward stochastic and adaptive workflows, where agents interpret intention, negotiate uncertainty, and build dynamically responsive experiences.

When these technologies—immersive VR and generative agents—are combined, they form the foundation for a new kind of interface: one that is spatial, conversational, and

adaptive. Such interfaces do not rely on code or static menus; they respond to voice, gesture, and embodied input. They transform abstract data into manipulable space, and procedural complexity into natural dialogue.

This is particularly relevant in the domain of education. Traditional educational systems remain bound to text, diagrams, and symbolic representation. While these tools are powerful, they often fall short when applied to fields that are inherently spatial, dynamic, or non-intuitive. Orbital mechanics, for example, involves motion through three-dimensional space governed by non-linear physical laws. Launch trajectories, gravitational slingshots, inclination changes, these are difficult to visualize and even harder to intuit.

In this context, immersive simulation becomes more than a visual aid: it becomes a cognitive bridge. A learner can speak a question and witness a launch trajectory materialize in their physical space. They can observe orbits evolve in real time, ask about inclinations or transfer windows, and receive explanations grounded in physics. Education becomes experiential, a process of exploration rather than instruction.

Moreover, generative agents provide a layer of accessibility that is historically absent in technical domains. They can guide the learner, interpret vague queries, correct misconceptions, and explain phenomena in adaptive ways. They act as intelligent mediators between curiosity and formal knowledge.

Given these technological conditions—the maturity of consumer VR, the rise of stochastic AI agents, and the persistent limitations of traditional educational media—this project is motivated by a clear opportunity: to construct a new type of educational experience. One that is not constrained by interface conventions, disciplinary jargon, or static presentation. One that invites the user to learn by seeing, asking, moving, and listening.

The convergence of embodied interaction and generative intelligence allows for a simulation system that is not only technically rigorous, but experientially meaningful. It enables a form of learning in which the abstract becomes tangible, the distant becomes near, and the user is placed at the center of the scientific process. This project emerges from the belief that space education, and scientific education more broadly, can and must evolve to meet the possibilities of our time.

1.3 Objectives

General Objective

To develop an interactive, agent-guided simulation platform that enables users to explore and understand orbital mechanics through embodied interaction, combining natural

language dialogue and real-time virtual reality visualizations.

Specific Objectives

1. Design and implement a simulation environment capable of rendering orbital trajectories in real time, grounded in physically accurate models.

2. Integrate a generative agent capable of interpreting natural language input, translating it into simulation parameters, and guiding the user through explanations and interactions.

3. Enable multimodal interaction by combining voice commands, spatial presence, and mixed reality visual feedback (VR immersion and AR passthrough capabilities) to create a seamless and intuitive user experience optimized for learning orbital mechanics concepts.

4. Ensure that all components of the system, simulation and agent, function coherently and communicate reliably in real time.

5. Create a system architecture that is modular and extensible, allowing for future expansion to other celestial bodies, educational modules, or mission types.

6. Demonstrate and deliver an open-source platform that showcases the potential for agent-guided virtual reality learning in orbital mechanics, enabling community validation, adaptation, and extension for diverse educational contexts.

2 Literature Review

2.1 Augmented and Virtual Reality in Immersive Educational Simulation Systems

Augmented Reality (AR) and Virtual Reality (VR) are complementary immersive technologies that enrich or replace a user's perception of the world. AR overlays digital content onto the real environment in real-time, allowing virtual objects to coexist with physical surroundings (Billinghurst; Clark; Lee, 2015). In contrast, VR completely immerses the user in a fully synthetic, computer-generated environment, blocking out the physical world. Milgram's classic "Reality-Virtuality" continuum illustrates these as endpoints: AR lies near the real-world end (mixing virtual content with reality), whereas VR occupies the extreme virtual end with an entirely simulated world (Milgram; Kishino, 1994). In essence, AR adds to the user's real-world experience, while VR transposes the user into an interactive virtual scene. Both technologies share common roots in decades of research and development. The term augmented reality was first coined by Caudell and Mizell (1992) in the context of assisting Boeing manufacturing with see-through displays (Caudell; Mizell, 1992). A few years later, Azuma's influential survey defined AR by three key characteristics: combining real and virtual content, interactive operation in real time, and accurate 3D registration of virtual objects in the physical world (Azuma, 1997; Billinghurst; Clark; Lee, 2015). VR, meanwhile, has been long conceptualized as achieving presence – the feeling of "being there" in a virtual environment – by engaging multiple senses with responsive 3D graphics and audio (Johnson-Glenberg, 2018). Modern definitions emphasize that VR provides immersive first-person experiences where users can interact with simulated worlds as if they were real, inducing a strong sense of presence and agency within the virtual scene.

2.1.1 Hardware Evolution:

AR and VR technologies have evolved rapidly, enabling consumer-grade devices that support realistic immersive experiences. While early head-mounted displays date back

to the 1960s (e.g., Sutherland's Sword of Damocles), the 2010s marked a turning point with modern devices. On the VR front, the Oculus Rift prototype (2010) by Palmer Luckey re-ignited interest with a wide field of view and affordable design. Crowdfunded in 2012 and acquired by Facebook in 2014, Oculus released its first consumer headset in 2016, alongside HTC's Vive, which introduced room-scale tracking. These devices brought high-fidelity visuals and motion tracking to mainstream audiences.

The next major step came with standalone VR headsets. The Oculus/Meta Quest series, starting in 2019, integrated processing and inside-out tracking directly into the headset. Quest 2 (2020) and Quest 3 (2023) improved resolution, optics, and added passthrough AR capabilities (Ruth, 2024). In parallel, PC-based headsets like the Valve Index and Varjo pushed the fidelity frontier for gaming and enterprise simulation.

AR hardware followed a distinct trajectory. Initial systems used handheld or laptop setups, but the release of Microsoft's HoloLens in 2016 marked the arrival of self-contained AR headsets with spatial mapping and inside-out tracking. Magic Leap One (2018) added novel display technologies (Billinghurst; Clark; Lee, 2015), while consumer experiments like Google Glass (2013) explored heads-up interfaces before being discontinued in 2023 (Ruth, 2024).

Smartphones played a critical role in scaling AR adoption. Apps like Pokémon GO (2016) introduced mainstream users to AR through camera overlays. ARKit (Apple) and ARCore (Google), launched in 2017, enabled mobile AR with motion and depth tracking (Vieyra; Vieyra, 2018).

Most recently, the line between AR and VR is blurring. Apple's Vision Pro (announced 2023) merges high-resolution VR with passthrough AR, positioning itself as a "spatial computer." With features like dual 4K displays and hand/eye tracking, it may represent a watershed moment for XR despite its premium price (Ruth, 2024).

As of 2025, the hardware ecosystem spans from mobile-based AR apps to advanced mixed reality headsets, forming a robust toolbox for immersive educational simulations.

2.1.2 Software Ecosystems and Frameworks:

Alongside hardware, a mature software ecosystem has enabled rapid development of immersive simulations. Modern game engines such as Unity and Unreal Engine have become the de facto platforms for AR/VR content creation. These engines provide high-performance 3D graphics rendering, physics simulation, and cross-platform deployment, greatly simplifying the creation of interactive virtual environments. Unity, for example, offers an entire XR development toolkit (with support for VR headsets and AR through packages like AR Foundation) that abstracts away device-specific details and allows de-

velopers to build an application once and deploy across multiple headsets (Atta *et al.*, 2022). Unreal Engine likewise includes integrated support for VR rendering and AR (via ARKit/ARCore plugins), making high-fidelity visualization accessible to developers in academia and industry.

For mobile AR, platform-specific frameworks are key. Apple's ARKit (introduced in iOS 11, 2017) and Google's ARCore (for Android, 2017) brought advanced AR capabilities to hundreds of millions of smartphones (Vieyra; Vieyra, 2018). These software development kits handle real-time tracking of the device's position, surface detection, lighting estimation, and more, allowing apps to place and persist virtual objects in the user's environment. Thanks to ARKit/ARCore, an educator can deploy an AR simulation on standard tablets or phones – for instance, letting students point an iPad at a textbook and see 3D molecules or physical field lines appear "attached" to the pages. On the web, the WebXR API has emerged as a W3C standard enabling AR and VR experiences to run directly in web browsers using JavaScript (World Wide Web Consortium, 2021). WebXR (successor to earlier WebVR/WebAR efforts) allows an immersive educational module to be accessed with a simple URL, lowering the barrier to entry (no app install required) and ensuring compatibility across different devices (from VR headsets to phones). This is particularly relevant for broad educational deployments, where web-based delivery can be more practical. Complementing these are various supporting frameworks: for example, libraries for spatial mapping, hand tracking, and user interaction (e.g. Microsoft's Mixed Reality Toolkit for Unity, or Vuforia for image-target AR) which provide higher-level tools for common AR/VR interactions. There are also open standards like OpenXR (released by the Khronos Group in 2019) that unify the interface to VR/AR hardware – a developer can write code once against OpenXR and run on any compliant headset (Oculus, SteamVR, Windows Mixed Reality, etc.), which is increasingly adopted by engines and platforms. In summary, the software landscape – from powerful 3D engines to AR phone toolkits and web standards – has matured to a point that immersive educational simulations can be built with relatively modest effort compared to a decade ago. This thesis will leverage these tools to construct its simulation system, ensuring it is built on proven, widely supported technology.

2.1.3 Use Cases in Education:

AR and VR have shown strong potential to enhance learning, particularly in subjects involving abstract or spatial concepts. Their core strength lies in making the invisible visible and the abstract tangible. In physics education, for instance, VR has helped students visualize and manipulate 3D vectors, improving understanding of vector addition and spatial relationships (Campos; Hidrogo; Zavala, 2022). Studies show that such immersive

tools can boost engagement and deepen comprehension of abstract STEM topics like electromagnetism or geometry through interactive, risk-free exploration (Campos; Hidrogo; Zavala, 2022; Johnson-Glenberg, 2018).

In astronomy and aerospace, where scales are far beyond human experience, immersive technologies offer unique advantages. VR enables virtual field trips through space — letting students stand on Mars or orbit planets — providing an intuitive grasp of scale and distance. Learners can explore the solar system with accurate proportions, making complex spatial relationships (like planetary distances or ring sizes) more comprehensible (Atta *et al.*, 2022). Astrophysical phenomena such as orbital mechanics and black hole dynamics are also made more accessible through interactive VR visualizations.

In aerospace engineering, VR and AR are increasingly used for hands-on training. Beyond traditional flight simulators, modern VR platforms allow students to perform simulated pre-flight inspections, engine maintenance, or spacecraft docking. Vaughn College, for example, uses VR for aviation trainees to practice inspecting and assembling parts, reinforcing mechanical familiarity before real-world exposure. Similarly, Atta et al. (2022) created a virtual "space lab" where students assemble a CubeSat in a simulated cleanroom, boosting their understanding of subsystem configuration through direct interaction and gamified tasks (Atta *et al.*, 2022).

AR complements this by overlaying digital instructions on real-world hardware. NASA's Project Sidekick exemplifies this: astronauts use HoloLens headsets aboard the ISS to receive real-time, spatially anchored maintenance guidance (NASA, 2015). In classrooms, AR enables students to interact with 3D models of rockets or overlay CAD designs onto physical parts, enriching theoretical lessons with live, contextual visualization (Atta *et al.*, 2022; Milgram; Kishino, 1994).

2.1.4 Embodiment, Interaction, and Spatial Cognition:

A recurring theme in the educational use of AR/VR is the role of embodied and spatial learning. Immersive technologies engage the human sensorimotor system – users move their bodies to navigate virtual spaces, use gestures to interact with virtual objects, and perceive environments at true scale. This physicality supports cognitive processing by leveraging innate spatial reasoning and muscle memory. The theory of embodied cognition holds that learning is grounded in the body's interactions with its environment, and AR/VR extend this principle digitally. Johnson-Glenberg (2018) highlights the pedagogical value of 3D gestures: when learners rotate a virtual object or walk through a graph, they build stronger memory links (Johnson-Glenberg, 2018). Her research shows that full-motion VR, where body movements align with abstract concepts, can deepen understanding and recall. Complementary studies (e.g., Liu et al., 2020) found improved reten-

tion when students enacted phenomena physically, and also noted increased presence and agency—factors tied to motivation (Campos; Hidrogo; Zavala, 2022; Johnson-Glenberg, 2018).

Spatial cognition benefits are also well-documented. VR's stereoscopic depth and six degrees of freedom help learners perceive complex spatial relationships, vital in subjects like anatomy, geography, and engineering. Students exploring a molecule or a solar system in VR can shift perspective freely, activating spatial memory and supporting what researchers call "situated learning" – knowledge acquired in rich spatial contexts becomes more intuitive and transferable. Campos et al. (2022), for example, found that immersive 3D interaction notably enhanced vector learning tasks requiring spatial reasoning (Campos; Hidrogo; Zavala, 2022). Similarly, in astronomy, VR's ability to scale from the Milky Way to Earth provides concrete visualizations of abstract systems (Kersting et al., 2024).

While AR/VR offer compelling tools, they are not magic bullets – user comfort, software complexity, and thoughtful pedagogical integration remain critical (Johnson-Glenberg, 2018). Still, evidence shows that immersive simulations can enhance traditional teaching, especially for learning goals involving visualization, experimentation, or embodied experience. In the context of this thesis, the implications are clear: AR and VR form a foundational layer. They enable students to interact with simulations of aerospace systems—such as satellites or orbital dynamics—in an intuitive and experiential manner. As hardware becomes lighter and more capable, and software ecosystems more robust, immersive tools are becoming increasingly viable in education. With spatial computing platforms entering mainstream use (Ruth, 2024), AR and VR are poised not just as delivery platforms but as new paradigms for engaging with knowledge.

2.1.4.0.1 Application to This Work This thesis leverages the pedagogical strengths of immersive spatial learning through Virtual Reality as the primary modality. While the platform architecture is built on mixed reality hardware (Meta Quest 3) that supports both VR and AR passthrough modes, the educational experience emphasizes full VR immersion for the reasons detailed in Section 3.3. The literature reviewed here establishes the broader context of immersive educational technologies, while the implementation focuses specifically on VR's capacity to place learners inside coherent spatial environments optimized for understanding orbital mechanics.

2.2 Generative Agents

Traditional software and simulations have been predominantly *deterministic*—given the same inputs, they yield the same outputs. Modern AI systems built on *genera-*

tive models, by contrast, introduce stochasticity and creativity. Large Language Models (LLMs) do not follow hard-coded rules; instead, they sample from probability distributions learned from vast textual corpora. Consequently, an LLM can produce context-dependent, varied responses rather than a single predetermined answer. This marks a paradigm shift from scripted to emergent behaviour. In recent work, advanced LLMs such as GPT-4 have even outperformed traditional reinforcement-learning agents in complex environments by reasoning through text rather than executing pre-programmed control policies (Carrasco; Rodriguez-Fernandez; Linares, 2025). While stochastic generation entails some unpredictability, it is precisely this creativity that lets *generative agents* adapt to scenarios beyond their designers' foresight.

At a conceptual level an LLM is a statistical language engine: given a textual history, it predicts the most plausible continuation one word at a time. Because it is trained on heterogeneous data, a single model can answer coding questions, analyse legal texts, or reason about orbital mechanics when prompted appropriately. This broad, generative capability underpins the rise of *LLM-powered agents* (Anthropic, 2024).

LLM-based agents are autonomous software entities that embed an LLM as their core "brain." An agent senses its environment, reasons about goals, and acts—iteratively—until a task is complete. Industry definitions describe such an agent as "a system that uses an LLM to reason through a problem, create a plan, and execute that plan with tools" (Chen, 2023; Huang; Grady, *et al.*, 2024). The LLM supplies the reasoning; auxiliary modules provide planning, memory, and tool use (Anthropic, 2024). Crucially, the agent—not the user—controls the loop: it may decide which function to call, when to revise a plan, or whether to request clarification (OpenAI, 2023). Hence an agent is more than a single LLM invocation; it is a continual perceive–think–act cycle.

Architectural Components

Generative-agent designs typically comprise five interacting elements (Anthropic, 2024; Huang; Grady, *et al.*, 2024):

- **Planning and reasoning.** The agent decomposes high-level goals into actionable steps, often prompting the LLM to produce an internal plan or "chain of thought."
- **Memory.** Short-term context (recent turns) and long-term knowledge (summaries or retrieved documents) are stored externally—e.g. in a vector database—and injected into prompts as needed.
- **Tool use and APIs.** Through structured outputs (JSON function calls, shell commands, *etc.*) the agent invokes external tools to compute, query, or effect changes in its environment (OpenAI, 2023).

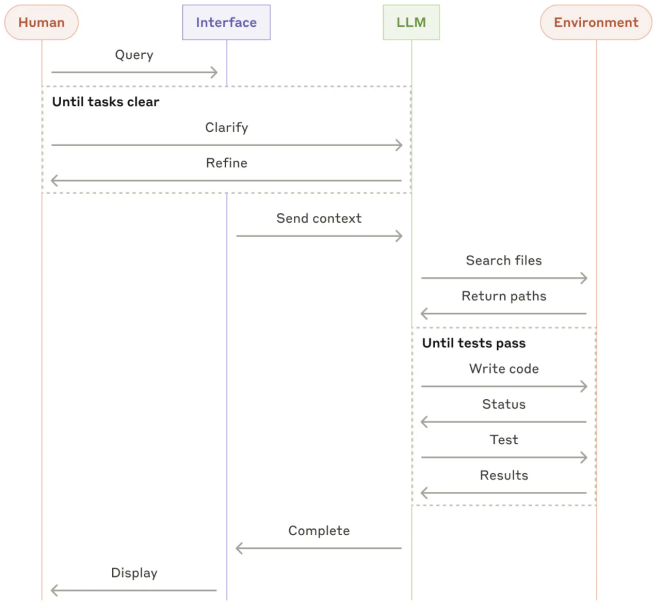

FIGURE 2.1 – End-to-end agentic workflow from Anthropic's "Building Effective Agents." The human issues a query through an *interface*; the LLM asks clarifying questions until the task is precise, receives contextual files, iteratively writes and tests code against the environment, and finally returns results for display (Anthropic, 2024).

- **Iterative control loop.** The agent cycles through *observe* → *reason* → *act* → *observe*, optionally reflecting or self-critiquing between steps to improve reliability.
- **Autonomy and adaptation.** Equipped with the above, the agent can switch strategies, recover from errors, and pursue its objective with minimal human micromanagement.

Applications and Relevance

- **Simulations and interactive worlds.** Park *et al.* created "Generative Agents" that populate a sandbox town with virtual characters who plan, remember, and socially interact—producing emergent storylines never scripted by the developers (Park *et al.*, 2023).
- **Aerospace guidance and control.** Carrasco *et al.* demonstrated an LLM agent piloting a spacecraft in the *Kerbal Space Program* simulation by iteratively reading textual telemetry and issuing control actions, matching classical controllers without explicit orbital equations (Carrasco; Rodriguez-Fernandez; Linares, 2025).
- **Legal reasoning.** Harvey AI equips law-firm associates with an agent that drafts memos, retrieves precedents, and iteratively refines analyses through dialogue—illustrating agentic workflows in language-dense tasks (Chen, 2023).

- **Education.** Khan Academy's *Khanmigo* employs GPT-4 as a Socratic tutor that adapts explanations to each learner, providing hints rather than answers and thereby personalising study sessions at scale (Academy, 2023).

2.3 Orbital Mechanics: The Physics of Celestial Motion

The intuitive, visual understanding of orbital motion is a primary objective of this project. While the generative agent handles the underlying calculations, a firm grasp of the governing principles is essential to frame the simulation's logic and appreciate its educational value. Orbital mechanics is the study of the motion of bodies under the influence of gravity. For missions in Earth's orbit and for interplanetary trajectories, the foundational principles discovered by Isaac Newton and Johannes Kepler provide a remarkably accurate framework for describing and predicting these celestial paths. This section outlines the core concepts that form the physical and mathematical basis of the simulation system, focusing on the specific orbital regimes and parameters that the platform enables users to explore: circular orbits, elliptical orbits, and escape trajectories.

2.3.1 Newtonian Gravity and the Two-Body Problem

At the heart of all orbital motion lies gravity. In the 17th century, Sir Isaac Newton formulated the Law of Universal Gravitation, stating that any two bodies attract each other with a force proportional to the product of their masses and inversely proportional to the square of the distance between them (Curtis, 2020). This is expressed mathematically as:

$$F = G\frac{m_1 m_2}{r^2}$$

where F is the gravitational force, G is the gravitational constant (6.674×10^{-11} $\mathrm{N{\cdot}m^2/kg^2}$), m_1 and m_2 are the masses of the two bodies, and r is the distance between their centers.

When applied to a satellite orbiting a celestial body like Earth, this law forms the foundation of the **two-body problem**. This model makes a critical simplifying assumption: it considers only the gravitational force between the satellite and the primary body (e.g., Earth), ignoring perturbations such as atmospheric drag, solar radiation pressure, and gravitational influences from other bodies like the Moon or the Sun (Vallado, 2013). While these forces are significant for high-precision, long-term trajectory prediction, the two-body model provides an elegant and highly accurate approximation for foundational analysis and educational purposes. The resulting equation of motion is:

$$\ddot{\vec{r}} + \frac{\mu}{r^3}\vec{r} = 0$$

Here, $\vec{r}$ is the position vector of the satellite relative to the primary body, $\ddot{\vec{r}}$ is its acceleration, and μ (mu) is the standard gravitational parameter of the system. For Earth-orbiting satellites, $\mu = GM_{\text{Earth}} \approx 398{,}600 \text{ km}^3/\text{s}^2$, where M_{Earth} is Earth's mass.

The solution to this differential equation reveals a profound geometric truth: under the inverse-square law of gravity, the satellite's path must be a **conic section**—a circle, ellipse, parabola, or hyperbola (Curtis, 2020). Which conic section results depends on the satellite's energy and angular momentum. This elegant mathematical result means that all orbital trajectories, from the circular path of the ISS to the hyperbolic escape of Voyager, are governed by the same fundamental physics expressed through different geometric shapes.

2.3.2 Kepler's Laws and Orbital Geometry

Johannes Kepler, working in the early 17th century with observational data from Tycho Brahe, empirically discovered three laws of planetary motion that would later be shown to be direct consequences of Newtonian gravity. These laws provide the geometric and temporal framework for understanding orbits (Bate; Mueller; White, 1971):

1. **First Law (Law of Orbits):** The orbit of a planet (or satellite) around the Sun (or Earth) is an ellipse, with the central body at one focus. A circle is the special case of an ellipse where both foci coincide.

2. **Second Law (Law of Areas):** A line connecting the satellite to the central body sweeps out equal areas in equal times. This means the satellite moves faster when closer to the central body (at periapsis) and slower when farther away (at apoapsis).

3. **Third Law (Law of Periods):** The square of the orbital period is proportional to the cube of the semi-major axis. Mathematically: $T^2 \propto a^3$, or more precisely, $T^2 = \frac{4\pi^2}{\mu} a^3$. This law directly relates orbital size to orbital period, explaining why the ISS at 420 km altitude completes an orbit in 92.8 minutes while the Hubble Space Telescope at 540 km takes slightly longer at approximately 95 minutes.

Kepler's laws were empirical observations that Newton later proved mathematically from first principles. Together, they provide both the geometric intuition (ellipses, not circles, are the general case) and quantitative relationships (period depends on altitude) that govern orbital motion.

2.3.3 Orbital Regimes: Circular, Elliptical, and Hyperbolic Trajectories

The shape of an orbit is determined by the satellite's total mechanical energy—the sum of its kinetic energy (from motion) and gravitational potential energy (from position in the gravity field). This energy dictates which conic section describes the trajectory. The platform's simulation implements three fundamental orbital regimes, each representing a different energy state and mission application.

2.3.3.1 Circular Orbits: Stable Operational Platforms

A **circular orbit** occurs when the satellite's velocity is precisely calibrated so that the centripetal acceleration required for circular motion exactly matches the gravitational acceleration at that altitude. This is the special case where eccentricity $e = 0$.

For a circular orbit at radius r from Earth's center (altitude $h = r - R_{\text{Earth}}$), the required orbital velocity is given by:

$$v_{\text{circular}} = \sqrt{\frac{\mu}{r}}$$

This relationship shows that orbital speed decreases with altitude: satellites in low Earth orbit (LEO) travel faster than those in higher orbits. For example, the ISS at 420 km altitude orbits at approximately 7.66 km/s, while the Hubble Space Telescope at 540 km altitude travels at approximately 7.59 km/s—slightly slower due to its higher altitude.

Circular orbits are preferred for operational missions requiring predictable, repeating ground tracks and stable altitude. The International Space Station (420 km, 51.6° inclination) and Hubble Space Telescope (540 km, 28.5° inclination) both use circular orbits because their missions benefit from the stability and predictability of constant altitude and speed.

Kepler's Third Law directly determines the orbital period for circular orbits:

$$T = 2\pi\sqrt{\frac{r^3}{\mu}}$$

This equation explains the relationship between altitude and period. The ISS at 420 km completes 15.5 orbits per day, providing frequent revisit times for Earth observation and crew operations. Hubble at 540 km has a slightly longer period, chosen to balance orbital stability with minimizing atmospheric drag while providing optimal viewing conditions for astronomical observations.

2.3.3.2 Elliptical Orbits: Variable Altitude Trajectories

An **elliptical orbit** occurs when $0 < e < 1$, where e is the eccentricity. The satellite's altitude and speed vary continuously as it moves around the ellipse. The closest point to Earth is called **periapsis** (or perigee for Earth orbits), and the farthest point is **apoapsis** (or apogee). The size of the ellipse is characterized by the **semi-major axis** a, which is half the longest diameter of the ellipse.

The relationship between the semi-major axis, periapsis radius r_p, and apoapsis radius r_a is:

$$a = \frac{r_p + r_a}{2}$$

The eccentricity quantifies how elongated the ellipse is:

$$e = \frac{r_a - r_p}{r_a + r_p}$$

When $e = 0$, the ellipse becomes a circle ($r_p = r_a$). As e approaches 1, the ellipse becomes increasingly elongated.

The satellite's speed at any point in an elliptical orbit is given by the **vis-viva equation**, one of the most fundamental relationships in orbital mechanics (Curtis, 2020):

$$v = \sqrt{\mu \left(\frac{2}{r} - \frac{1}{a} \right)}$$

This equation reveals that orbital speed depends on both the current position r and the orbit's overall size a. At periapsis, where r is smallest, the satellite moves fastest. At apoapsis, where r is largest, it moves slowest. This speed variation is a direct consequence of Kepler's Second Law: the satellite must move faster when closer to Earth to sweep equal areas in equal times.

Elliptical orbits have important applications. Highly Elliptical Orbits (HEO) are used for communications satellites serving high-latitude regions, as the satellite spends most of its time near apoapsis with excellent visibility over polar regions. Transfer orbits between circular orbits are also elliptical, with the initial circular orbit at periapsis and the target circular orbit at apoapsis.

2.3.3.3 Hyperbolic Trajectories: Escaping Earth's Gravity

A **hyperbolic trajectory** occurs when $e \geq 1$. Unlike elliptical orbits, which are closed and periodic, hyperbolic trajectories are open curves—the spacecraft approaches Earth,

swings around it, and departs, never to return. This regime represents escape from Earth's gravitational influence.

The minimum speed required to achieve escape from Earth's surface is the **escape velocity**:

$$v_{\text{escape}} = \sqrt{\frac{2\mu}{r}}$$

At Earth's surface ($r = R_{\text{Earth}} = 6371$ km), this yields approximately 11.2 km/s. Notice that escape velocity is exactly $\sqrt{2}$ times the circular orbital velocity at the same radius—this factor of $\sqrt{2}$ represents the energy difference between a bound circular orbit and an unbound escape trajectory.

For a hyperbolic trajectory, the spacecraft's velocity at any distance is given by a modified vis-viva equation:

$$v = \sqrt{\mu \left(\frac{2}{r} + \frac{1}{a} \right)}$$

Note the sign change: for hyperbolic orbits, the semi-major axis a is defined as negative, reflecting the fact that the orbit is unbound with positive total energy.

The Voyager spacecraft exemplify hyperbolic escape trajectories. After launch and acceleration to sufficient velocity, they followed hyperbolic paths that carried them beyond Earth's sphere of influence and into interplanetary space. The platform uses Voyager's trajectory to demonstrate the transition from bound elliptical motion to unbound hyperbolic escape, illustrating the fundamental energy threshold that separates orbiting from departing.

2.3.4 Orbital Orientation: Inclination and Coverage

While the size and shape of an orbit (determined by a and e) govern its energy and geometry, the **inclination** determines the orbit's orientation in three-dimensional space. Inclination i is the angle between the orbital plane and a reference plane, typically Earth's equatorial plane. An inclination of 0 defines an equatorial orbit, while 90 defines a polar orbit that passes directly over both poles.

Inclination is not arbitrary—it is fundamentally constrained by the launch site's latitude and the physics of rotation. When a rocket launches eastward (prograde), it benefits from Earth's rotational velocity, which is maximum at the equator (465 m/s) and decreases toward the poles. The minimum achievable inclination from a launch site is approximately equal to the site's latitude. For example, launches from Kennedy Space Center (28.5°N) can achieve inclinations of 28.5° or greater, but reaching lower inclinations would require the rocket to perform an energetically expensive plane change maneuver.

This launch constraint explains many mission orbital parameters:

- **ISS (51.6° inclination):** Designed to be accessible from both Kennedy Space Center and the Baikonur Cosmodrome in Kazakhstan (45.6°N). The 51.6° inclination allows Russian Soyuz launches from Baikonur while remaining within reasonable energy budgets for US launches.
- **Hubble (28.5° inclination):** Launched from Kennedy Space Center at the minimum possible inclination, maximizing the rotational velocity assist and minimizing fuel requirements. This low inclination also provides good sky coverage for astronomical observations while avoiding prolonged periods in Earth's shadow.

Inclination also determines ground track coverage. An equatorial orbit ($i = 0$) never passes over polar regions. A polar orbit ($i = 90$) eventually covers the entire surface as Earth rotates beneath it. Intermediate inclinations provide a balance between coverage and launch efficiency. The platform's mission-specific implementations demonstrate how operational requirements (crew access for ISS, astronomical visibility for Hubble) drive inclination choices.

2.3.5 The Educational Foundation for Interactive Exploration

The physics and mathematics outlined in this section—Newtonian gravity, Kepler's laws, the vis-viva equation, and the geometric properties of conic sections—form the computational foundation of the simulation platform. More importantly, they represent the conceptual framework that users explore through embodied interaction in virtual reality.

Traditional orbital mechanics education presents these concepts through equations on paper and two-dimensional diagrams. Students memorize formulas and solve problems numerically, but the intuitive, spatial understanding of why the ISS orbits at 7.66 km/s or why the Hubble telescope requires a specific altitude and inclination often remains elusive. The three-dimensional geometry of an inclined orbit, the speed variation along an ellipse, and the meaning of escape velocity are fundamentally spatial phenomena that are difficult to internalize from textbooks alone.

The platform's approach inverts this pedagogy. Users begin not with equations but with questions and curiosity: "Show me the ISS orbit." "Why does Hubble orbit where it does?" "How did Voyager leave Earth?" The generative agent translates these natural language queries into the precise orbital parameters described in this section—altitude, eccentricity, inclination—and the Unity simulation engine renders the resulting trajectories as visible, three-dimensional curves in space. Users inhabit the orbital environment,

observing how the ISS's 420 km circular orbit compares to Hubble's 540 km orbit, seeing the ellipse stretch as eccentricity increases, watching the hyperbolic escape path diverge from closed elliptical motion.

This section has established the theoretical foundation that makes such exploration both accurate and meaningful. The circular orbits users create are governed by $v = \sqrt{\mu/r}$. The elliptical orbits follow the vis-viva equation. The hyperbolic escapes exceed $v_{\text{escape}} = \sqrt{2\mu/r}$. The platform's educational value rests on this foundation: it translates rigorous astrodynamics into intuitive visual experience, enabling users to build genuine understanding of orbital mechanics through guided exploration rather than rote memorization.

3 Methodology

3.1 Design Philosophy and Approach

The development of this project is fundamentally an exploratory research endeavour into a new paradigm of human-computer interaction for educational purposes. Given the innovative and complex nature of integrating generative AI, immersive mixed reality, and embodied interfaces, a rigid, waterfall-style development plan would be inappropriate. Instead, the methodology is guided by a philosophy that embraces iteration and modularity to navigate the technical challenges and discovery process inherent in such work.

The approach is defined by three core principles:

- **Prototype-Driven:** The primary goal is the creation of a functional prototype that demonstrates the feasibility and potential of the proposed system. This approach prioritizes implementing the core functionalities of the user experience over exhaustive feature development, allowing for tangible and testable results that can validate the project's central thesis.
- **Iterative Development:** The project will be built in iterative cycles, following a process of building a core feature, testing its performance and usability, and refining it based on the results. This allows for flexibility in the implementation details, acknowledging that the optimal solutions for agent prompting and user interaction will be discovered and improved upon throughout the development lifecycle.
- **Modular Architecture:** The system is designed as a collection of distinct yet interconnected modules: the generative agent (the "brain") and the simulation and visualisation engine (the "world"). This modularity, a key objective of this project, makes the complex system manageable, facilitates parallel development and testing of components, and ensures the final architecture is extensible for future work.

These principles guided a four-phase development strategy. Phase 1 established the conversational agent in the Hub environment with circular and elliptical orbit creation

capabilities. Phase 2 implemented three Mission Spaces (ISS, Hubble, Voyager) with specialist agents and scene transition tools. Phase 3 integrated bidirectional voice through ElevenLabs (Scribe v2 for speech-to-text, TTS for character synthesis). Phase 4 deployed the complete system to Meta Quest 3 with immersive VR visualization. Each phase produced a testable, working system that integrated seamlessly with subsequent development without requiring architectural changes.

3.2 System Architecture and Data Flow

The platform architecture comprises two integrated spaces: the "Hub" (Mission Control) where users create custom orbits through conversation, and "Mission Spaces" (ISS, Hubble, Voyager) where specialists demonstrate real missions. Users navigate between these environments via voice commands (`route_to_mission`, `return_to_hub`), with conversational context preserved across transitions.

The system separates conversational intelligence (OpenAI GPT-4.1) from spatial visualization (Unity 3D), connected through a tool-calling interface. The agent interprets natural language, selects appropriate tools (`create_circular_orbit`, `create_elliptical_orbit`, `set_simulation_speed`, `pause_simulation`, `reset_simulation_time`, `clear_orbit`, `route_to_mission`, `return_to_hub`), and Unity executes the corresponding physics calculations and rendering.

3.2.1 Interaction Flow

Each user interaction follows a nine-step cycle from speech input to audio output:

1. User speaks request via Quest 3 microphone
2. ElevenLabs Scribe v2 transcribes audio to text
3. OpenAI GPT-4.1 interprets intent and selects appropriate tool
4. Unity's `ToolExecutor` validates parameters and invokes corresponding C# method
5. `SceneTransitionManager` loads Mission Space (for `route_to_mission`); `OrbitController` calculates trajectory using vis-viva equation and renders visualization (for orbit creation)
6. Tool execution result returns to LLM as feedback
7. Agent generates educational response based on tool result and conversation context

8. ElevenLabs TTS synthesizes character-specific voice (Mission Control's authority, ISS specialist's precision, Voyager specialist's Sagan-like wonder)

9. Audio plays through Quest 3 spatial audio system

Conversation history is maintained across turns and scene transitions, enabling contextual dialogue. Users control all aspects—orbit creation, mission navigation, simulation parameters—through voice alone.

3.3 Mixed Reality Design Rationale

While the Quest 3 supports both VR and AR passthrough, the platform emphasizes immersive VR for pedagogical reasons. Orbital mechanics involves scales (420 km for ISS, 35,786 km for geostationary orbit) incompatible with domestic spaces—AR overlays would show trajectories passing through walls and furniture, creating perceptual friction between room-scale and cosmic-scale contexts. VR isolation places users within the orbital environment itself, establishing coherent spatial context where Earth floats in space and trajectories exist in their natural domain. The architecture supports AR passthrough for collaborative learning, museum installations, or classroom demonstrations where physical anchoring adds value, but the primary experience prioritizes the modality best suited to the content.

3.4 Technical Implementation

The system implements conversational AI (OpenAI GPT-4.1 for reasoning, ElevenLabs for voice synthesis/transcription) interfaced with Unity 3D physics simulation. The agent embodies four characters—Mission Control in the Hub, plus three mission specialists (ISS crew perspective for LEO operations, Hubble engineer for telescope mission design, Voyager/Sagan persona for interplanetary trajectories)—each with distinct voice profiles and expertise areas. Unity implements two-body orbital physics in C#, calculates trajectories via vis-viva equation, and renders visualizations on Quest 3. The platform supports circular orbit creation (160-35,786 km altitude) and elliptical orbit creation (periapsis/apoapsis 160-100,000 km), with inclination constrained to 0-180°.

3.5 Development and Version Control

The project follows systematic development practices using GitHub for version control. All source code—including Unity C# scripts, prompt templates, and configuration

files—is tracked in a central repository, providing complete history of changes and enabling experimental work through branching without compromising the main project stability. This systematic approach aligns with the iterative development philosophy, where each development cycle's progress is documented and preserved.

3.6 Validation and Dissemination Strategy

The platform's educational effectiveness and technical implementation are validated through demonstration of complete interaction scenarios across all mission spaces. The validation approach prioritizes real-world usage patterns: users exploring orbital mechanics through natural conversation, navigating between Hub and Mission Spaces, and experiencing the full voice-driven VR workflow. The complete implementation is released open-source on GitHub, enabling community validation, adaptation, and extension. Users provide their own API keys for OpenAI and ElevenLabs services, ensuring accessibility without imposed service costs. Comprehensive documentation covers Unity configuration, Quest 3 deployment, and integration procedures.

3.7 Core Module Implementation

This section describes how the platform's four primary modules—conversational agent, orbital physics simulation, voice integration, and virtual reality environment—work together to create an immersive educational experience for learning orbital mechanics. Each subsection explains the educational rationale behind key design decisions and how they support the learning objectives established in Section 3. Complete technical specifications are provided in Appendices A–D.

3.7.1 Agent System Implementation

The conversational agent system removes the traditional barrier between learning intent and technical execution by enabling natural language control of orbital simulations. Learners can express goals like "Create an orbit matching the ISS" or "Show me a highly elliptical orbit" without needing to understand programming, coordinate systems, or simulation APIs. This design decision directly addresses a core challenge in physics education: allowing students to focus on conceptual understanding rather than technical syntax.

The system implements GPT-4.1 with a structured tool-calling framework that interprets user requests and invokes validated simulation commands. When a learner asks to create an orbit, the agent translates natural language into precise physics parameters (al-

titude, inclination, eccentricity), executes the orbital calculation, and explains the result in educational terms. Critically, the agent disambiguates between orbital velocity (the speed required to maintain a specific orbit, calculated from physics) and simulation time speed (how fast the visualization plays back)—a common source of confusion that this explicit separation prevents.

The platform embodies two agent archetypes: Mission Control (at the Hub) focuses on orbit creation and simulation control, while three mission specialists (ISS, Hubble, Voyager) provide mission-specific educational context when learners navigate to dedicated Mission Spaces. This dual-character design supports two learning modes: hands-on experimentation at the Hub, and contextual deepening through mission-specific dialogue. Conversation history persists across scene transitions, enabling learners to ask follow-up questions like "What was the altitude of the orbit I just created?" after switching contexts—supporting iterative, exploratory learning patterns.

Technical implementation details, including prompt architecture, tool schemas, context management algorithms, and API integration specifications, are documented in Appendix A.

3.7.2 Orbital Physics Simulation

The orbital physics engine translates altitude specifications into velocity requirements automatically, making visible a fundamental relationship that students often struggle to grasp: that orbital speed is not arbitrary but determined by altitude through gravitational physics. When a learner requests "an orbit at 420 km like the ISS," the system calculates the required velocity (7.66 km/s) using the vis-viva equation and displays both values together. This automatic calculation prevents a common misconception—that higher orbits move faster—by immediately showing that geostationary satellites at 35,786 km altitude actually travel slower (3.07 km/s) than low Earth orbit satellites, despite their greater distance.

The simulation implements two-body Keplerian mechanics with physically accurate trajectory calculations for circular and elliptical orbits. Visual trajectories render as continuous curves in VR space, allowing learners to observe geometric properties directly: circular orbits maintain constant radius, while elliptical orbits visually demonstrate eccentricity through their oblong shape. Scale compression maps Earth's 6,371 km radius to a comfortable VR viewing volume while preserving proportional relationships—the ISS appears at 6.6% of Earth's radius above the surface, matching the real ratio—enabling learners to develop accurate spatial intuition about orbital altitudes without being overwhelmed by vast scales.

Critically, all physics calculations occur in real units (km, km/s) before conversion to rendering space, ensuring that displayed values match published orbital data for ISS, Hubble, and other missions. This fidelity allows learners to verify simulation results against authoritative sources, building confidence in the educational tool. Complete physics implementation, including vis-viva equation derivations, scale compression algorithms, and trajectory visualization methods, appears in Appendix B.

3.7.3 Voice Integration Pipeline

Voice interaction addresses a practical constraint of VR environments: hands holding controllers cannot easily type, and virtual keyboards break immersion. The system implements bidirectional speech through push-to-talk input (Quest 3 controller A button) and synthesized character voices, enabling learners to engage in natural spoken dialogue while manipulating 3D orbital visualizations. This hands-free modality supports exploratory learning patterns where students voice hypotheses ("What happens if I increase the altitude?"), observe results, and refine understanding through iterative questioning—a cognitive process difficult to sustain when switching between physical keyboards and immersive VR.

Each agent character embodies a distinct voice: Mission Control speaks with authoritative encouragement at the Hub, while mission specialists (like Anastasia, the ISS expert) adopt personalities aligned with their educational roles—professional, technical, and approachable. This character differentiation serves pedagogical purposes beyond engagement: learners develop associative memory between voice identity and knowledge domain, reinforcing context switching as they navigate between experimental workspace (Hub) and mission-specific deepening (ISS, Hubble, Voyager spaces). Voice synthesis occurs within 1–3 seconds of agent response generation, maintaining conversational flow without perceptible delays that would disrupt the learning dialogue.

The push-to-talk mechanism balances spontaneity with intentionality: learners explicitly signal when they wish to speak, preventing accidental voice activation while preserving the natural rhythm of conversation. This design choice emerged from recognizing that educational dialogue differs from commercial voice assistants—students need time to think between questions, and the platform should not interpret silence as disengagement. Technical details of speech-to-text processing, audio synthesis parameters, and character voice management appear in Appendix C.

3.7.4 Virtual Reality Environment

The virtual reality environment transforms abstract orbital mechanics into spatial experiences that leverage human depth perception and proprioception. Orbits exist as three-dimensional curves that learners can walk around, crouch beneath, and observe from multiple vantage points—building geometric intuition impossible to achieve through 2D screens or static diagrams. Seeing an elliptical orbit's eccentricity from different angles, or observing how inclination tilts the orbital plane relative to Earth's equator, engages spatial reasoning faculties that support conceptual understanding of orbital geometry.

The platform deploys to Meta Quest 3, a standalone VR headset enabling tetherless movement around orbital visualizations without PC connection constraints. Maintaining 90 Hz stereoscopic rendering ensures visual comfort during extended learning sessions, preventing the nausea and fatigue that would undermine educational effectiveness. This frame rate requirement drove architectural decisions throughout the implementation: single-pass instanced rendering reduces GPU overhead, texture compression minimizes memory bandwidth, and asynchronous scene loading prevents visible stuttering during navigation between Hub and Mission Spaces.

The multi-scene architecture supports distinct learning contexts: the Hub provides an experimental workspace for orbit creation and manipulation, while three Mission Spaces (ISS, Hubble, Voyager) offer focused environments for deepening understanding of specific missions. Scene transitions preserve conversation history and simulation state, allowing learners to seamlessly shift between hands-on experimentation and contextual exploration. Spatial UI elements render in 3D world space rather than head-locked overlays, maintaining presence and spatial grounding while providing necessary information—mission elapsed time, simulation speed, and dialogue responses appear as objects in the environment rather than disconnected interface chrome.

Technical specifications for Quest 3 deployment, including Android build configuration, input system implementation, stereo rendering pipeline, and performance optimization strategies, are detailed in Appendix D.

References

ACADEMY, K. **Khanmigo: Khan Academy's AI-Powered Tutor (GPT-4 Pilot)**. [*S. l.: s. n.*], 2023. https://openai.com/blog/khan-academy/. Khan Academy Press Release (via OpenAI), Mar. Cit. on p. 26.

ANTHROPIC. **Building Effective Agents**. [*S. l.: s. n.*], 2024. https://www.anthropic.com/engineering/building-effective-agents. Anthropic Engineering Blog, Dec 19. Cit. on pp. 24, 25.

ATTA, G.; ABDELSATTAR, A.; ELFIKY, D.; ZAHRAN, M.; FARAG, M.; SLIM, S. O. Virtual Reality in Space Technology Education. **Education Sciences**, v. 12, n. 12, p. 890, 2022. DOI: 10.3390/educsci12120890. Cit. on pp. 21, 22.

AZUMA, R. T. A Survey of Augmented Reality. **Presence: Teleoperators & Virtual Environments**, v. 6, n. 4, p. 355–385, 1997. Cit. on p. 19.

BATE, R. R.; MUELLER, D. D.; WHITE, J. E. **Fundamentals of Astrodynamics**. New York, NY: Dover Publications, 1971. ISBN 978-0-486-60061-1. Cit. on p. 27.

BILLINGHURST, M.; CLARK, A.; LEE, G. A Survey of Augmented Reality. **Foundations and Trends in Human–Computer Interaction**, v. 8, n. 2-3, p. 73–272, 2015. DOI: 10.1561/1100000049. Cit. on pp. 19, 20.

CAMPOS, E.; HIDROGO, I.; ZAVALA, G. Impact of Virtual Reality Use on the Teaching and Learning of Vectors. **Frontiers in Education**, v. 7, p. 965640, 2022. DOI: 10.3389/feduc.2022.965640. Cit. on pp. 21–23.

CARRASCO, A.; RODRIGUEZ-FERNANDEZ, V.; LINARES, R. Large Language Models as Autonomous Spacecraft Operators in Kerbal Space Program. **Advances in Space Research**, 2025. Preprint: arXiv:2505.19896. Cit. on pp. 24, 25.

CAUDELL, T. P.; MIZELL, D. W. Augmented Reality: An Application of Heads-Up Display Technology to Manual Manufacturing Processes. *In:* PROC. 25th Hawaii Int. Conf. on System Sciences (HICSS). [*S. l.*]: IEEE, 1992. p. 659–669. Cit. on p. 19.

CHEN, D. **Exploring Autonomous Agents: A Semi-Technical Dive**. [*S. l.: s. n.*], 2023. https://www.sequoiacap.com/article/autonomous-agents-perspective/. Sequoia Capital Blog, May 11. Cit. on pp. 24, 25.

CURTIS, H. D. **Orbital Mechanics for Engineering Students**. 4th. Oxford, UK: Elsevier/Butterworth-Heinemann, 2020. ISBN 978-0-08-102927-1. Cit. on pp. 26, 27, 29.

HUANG, S.; GRADY, P., *et al.* **Generative AI's Act 1: The Agentic Reasoning Era Begins**. [*S. l.: s. n.*], 2024. https://www.sequoiacap.com/article/generative-ais-act-o1/. Sequoia Capital Essay, Oct 9. Cit. on p. 24.

JOHNSON-GLENBERG, M. C. Immersive VR and Education: Embodied Design Principles That Include Gesture and Hand Controls. **Frontiers in Robotics and AI**, v. 5, p. 81, 2018. DOI: 10.3389/frobt.2018.00081. Cit. on pp. 19, 22, 23.

MILGRAM, P.; KISHINO, F. A taxonomy of mixed reality visual displays. **IEICE Trans. Information and Systems**, E77-D, n. 12, p. 1321–1329, 1994. Cit. on pp. 19, 22.

NASA. **NASA and Microsoft Collaborate to Bring Science Fiction to Science Fact**. [*S. l.: s. n.*], 2015. Press Release 15-139 (June 25, 2015). Available: https://www.nasa.gov/news-release/nasa-microsoft-collaborate-to-bring-science-fiction-to-science-fact. Cit. on p. 22.

OPENAI. **Function Calling and Other API Updates**. [*S. l.: s. n.*], 2023. https://openai.com/blog/function-calling-and-other-api-updates. OpenAI Blog, June 13. Cit. on p. 24.

PARK, J. S.; O'BRIEN, J. C.; CAI, C. J.; MORRIS, M. R.; LIANG, P.; BERNSTEIN, M. S. Generative Agents: Interactive Simulacra of Human Behavior. *In:* PROCEEDINGS of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23). [*S. l.: s. n.*], 2023. Cit. on p. 25.

RUTH, J. S. **Has Apple Changed VR and AR's Trajectory with the Vision Pro?** [*S. l.: s. n.*], 2024. InformationWeek (Feb. 16, 2024). Available from: https://www.informationweek.com/data-management/has-apple-changed-vr-and-ar-s-trajectory-with-the-vision-pro-. Cit. on pp. 20, 23.

VALLADO, D. A. **Fundamentals of Astrodynamics and Applications**. 4th. Hawthorne, CA: Microcosm Press and Springer, 2013. ISBN 978-1-881-88318-0. Cit. on p. 26.

VIEYRA, R.; VIEYRA, C. **Comparing Google ARCore and Apple ARKit**. [*S. l.: s. n.*], 2018. Vieyra Software Blog (Sep. 23, 2018). Available from: https://www.vieyrasoftware.net/single-post/2018/09/23/comparing-google-arcore-and-apple-arkit. Cit. on pp. 20, 21.

WORLD WIDE WEB CONSORTIUM. **WebXR Device API (Editor's Draft, May 2021)**. [*S. l.: s. n.*], 2021. W3C Working Draft. Available from: `https://www.w3.org/TR/webxr/`. Cit. on p. 21.

Appendix A - Agent System Implementation

This appendix provides detailed technical implementation specifications for the conversational agent system described in Section 3.7.1. All class names, method signatures, file paths, and code excerpts are verified against the Unity project source code.

A.1 Prompt Architecture

The agent system operates through structured prompts stored in the `PromptSettings` ScriptableObject configuration asset. Table A.1 summarizes the prompt components and their purposes.

TABLE A.1 – Agent Prompt Component Specifications

Prompt Component	Size	Purpose
`toolSelectionPrompt`	460 lines	Interprets user intent, returns tool JSON
`responsePrompt`	270 lines	Generates natural language responses
`specialistSystemPrompt`	412 lines	Frames mission specialist character
`nonToolResponseTemplate`	—	Handles conversational interactions
`toolResponseTemplate`	—	Formats tool execution feedback
`specialistIntroTemplate`	—	Generates 40-word greetings

A.1.1 Hub Agent: Three-Tier Prompt System

The Hub agent (Mission Control) uses three coordinated prompts:

A.1.1.0.1 Tool Selection Prompt (460 lines) Instructs GPT-4.1 to analyze user natural language input and return structured JSON identifying which tool to invoke. The prompt explicitly defines eight available tools:

- **Orbit Creation**: `create_circular_orbit`, `create_elliptical_orbit`

- **Simulation Control**: `set_simulation_speed`, `pause_simulation`, `reset_simulation_time`
- **Workspace Management**: `clear_orbit`
- **Navigation**: `route_to_mission`, `return_to_hub`

A.1.1.0.2 Response Prompt (270 lines) Generates natural language explanations of tool execution results. Includes explicit disambiguation guidance to prevent confusion between:

- **Orbital velocity** (physics-calculated, 7.66 km/s for ISS)
- **Simulation time speed** (user-controllable playback multiplier)

A.1.1.0.3 Non-Tool Response Template Handles conversational interactions that do not require tool execution, such as greetings ("Hello, I'm Mission Control"), capability inquiries ("What can you do?"), and educational questions.

A.1.2 Mission Specialist Prompts

Mission Space specialists (ISS, Hubble, Voyager) use the `specialistSystemPrompt` (412 lines) which frames the agent as an enthusiastic mission expert focused on education rather than simulation control. Character configuration occurs through `MissionConfig` ScriptableObject assets:

- `ISS_Config.asset`: Character name "Anastasia", personality "Professional engineer - clear, technical, friendly"
- `Hubble_Config.asset`: Hubble Space Telescope mission specialist
- `Voyager_Config.asset`: Voyager interplanetary mission specialist

The `specialistIntroTemplate` generates concise 40-word, 10-15 second greetings acknowledging the routing context from `route_to_mission`.

A.2 Tool Schema and Validation

The eight tools are defined in `ToolSchemas.json` (169 lines) with complete JSON Schema specifications. Table A.2 documents parameter constraints enforced by the `ToolRegistry` validation system.

TABLE A.2 – Tool Parameter Constraints

Tool	Parameter	Constraint
`create_circular_orbit`	`altitude_km`	160–35,786 km
`create_circular_orbit`	`inclination_deg`	0–180°
`create_elliptical_orbit`	`periapsis_km`	160–35,786 km
`create_elliptical_orbit`	`apoapsis_km`	160–100,000 km
`create_elliptical_orbit`	`inclination_deg`	0–180°
`set_simulation_speed`	`speed_multiplier`	0.1–100×

A.2.1 Tool Execution Pipeline

The `ToolExecutor` class receives validated tool calls from the agent system and invokes corresponding C# methods:

- **Orbit Tools** → `OrbitController.CreateCircularOrbit()`, `CreateEllipticalOrbit()`
- **Time Controls** → `TimeController.SetSpeed()`, `Pause()`, `ResetTime()`
- **Navigation Tools** → `SceneTransitionManager.TransitionToMission()`, `TransitionToHub()`
- **Workspace** → `OrbitController.ClearOrbit()`

Execution results—success status, generated orbital parameters, error messages—feed back into the LLM response generation cycle through the response prompt template.

A.3 Conversation Context Management

The `ConversationHistory` class maintains conversation continuity across multi-turn dialogues and scene transitions. Table A.3 documents the exchange data structure.

TABLE A.3 – Conversation Exchange Data Structure

Field	Content
`timestamp`	DateTime of exchange
`userMessage`	User's natural language input
`agentResponse`	Agent's generated response
`toolExecuted`	Tool name (or null if conversational)
`location`	Current scene (Hub, ISS, Hubble, Voyager)

A.3.1 Context Window Management

The system maintains a sliding window of the last 10 exchanges (`maxHistorySize = 10`). Two methods provide context injection into prompts:

- `GetFormattedHistory(lastNExchanges = 5)`: Returns detailed history with timestamps, locations, and tool executions for the last 5 exchanges
- `GetContextSummary(lastNExchanges = 3)`: Returns condensed 3-exchange summary optimized for token efficiency

A.3.2 Cross-Scene Persistence

Scene transitions preserve conversation history through Unity's `DontDestroyOnLoad` mechanism. The `PromptConsole` GameObject, containing the `ConversationHistory` component, persists across scene unloading when users invoke `route_to_mission` or `return_to_hub` tools. This ensures unbroken dialogue continuity: a user can ask "What was the ISS orbit altitude I created in the Hub?" after transitioning to the ISS Mission Space.

A.4 API Integration

The `OpenAIClient` class (150 lines) implements asynchronous HTTP communication with OpenAI's Responses API endpoint (`https://api.openai.com/v1/responses`).

A.4.1 Request Structure

Requests to the `/responses` endpoint include:

- **Model**: `"gpt-4.1"`
- **Input**: User's natural language message
- **Instructions**: Concatenated system prompt + conversation history + tool schemas

The `CompleteAsync()` method constructs JSON payloads using Unity's `UnityWebRequest` for async/await compatibility.

A.4.2 Response Parsing

The client extracts assistant text from JSON responses through a two-stage fallback:

1. **Primary**: Extract `output_text` convenience field (if present)
2. **Fallback**: Concatenate all `output[].content[].text` arrays

Tool call JSON undergoes validation by `ToolRegistry` before execution. Results format back into natural language through the response prompt template system, generating contextual explanations like: "I've created a circular orbit at 420 km altitude with 51.6° inclination. The orbital velocity is 7.66 km/s, matching the ISS configuration."

A.4.3 Mission-Specific Configuration

Each Mission Space scene loads scene-specific `OpenAISettings` ScriptableObject assets that override the default system prompt, enabling character switching when users transition from Hub (Mission Control) to Mission Spaces (specialist agents).

Appendix B - Orbital Physics Implementation

This appendix provides detailed technical specifications for the orbital physics simulation engine described in Section 3.7.2. All equations, algorithms, class methods, and numerical values are verified against the Unity project physics implementation.

B.1 Two-Body Keplerian Mechanics

The simulation implements two-body orbital mechanics under the following simplifying assumptions:

- Earth modeled as a point mass at the coordinate system origin
- Satellite treated as a massless test particle (no gravitational influence on Earth)
- No atmospheric drag, solar radiation pressure, or third-body perturbations
- Instantaneous orbital maneuvers (no finite burn durations)

These assumptions yield closed-form Keplerian solutions suitable for educational visualization while maintaining physical accuracy for the mission profiles studied (ISS, Hubble, Voyager departure trajectory).

B.2 Vis-Viva Equation Implementation

The vis-viva equation relates orbital velocity to position and total orbital energy. Table B.1 documents the physical constants used throughout the simulation.

TABLE B.1 – Physical Constants for Orbital Calculations

Constant	Value	Symbol
Earth's standard gravitational parameter	398,600 km^3/s^2	μ
Earth's mean radius	6,371 km	$R_\oplus$
Unity scale compression factor	0.000785 Unity/km	k
Unity Earth radius	5 Unity units	R_{Unity}

B.2.1 Circular Orbit Calculation

Circular orbits ($e = 0$) simplify the vis-viva equation to:

$$v_{\text{circular}} = \sqrt{\frac{\mu}{r}} \tag{B.1}$$

where $r = R_\oplus + h$ is the orbital radius from Earth's center, and h is the altitude above Earth's surface.

B.2.1.0.1 Implementation Method The `OrbitController.CreateCircularOrbit()` method (lines 229–290) accepts altitude in kilometers and automatically calculates orbital velocity, eliminating user confusion between altitude and speed parameters. Algorithm B.2 documents the calculation sequence.

TABLE B.2 – Circular Orbit Calculation Algorithm

Step	Calculation
1. Validate altitude	$h_{\text{input}} \rightarrow \text{Clamp}(160, 35{,}786)$ km
2. Compute orbital radius	$r = R_\oplus + h = 6{,}371 + h$ km
3. Calculate orbital velocity	$v = \sqrt{\mu/r}$ km/s
4. Convert to Unity scale	$r_{\text{Unity}} = R_{\text{Unity}} + h \cdot k$
5. Convert to angular velocity	$\omega = (v \cdot k)/r_{\text{Unity}}$ rad/s

B.2.1.0.2 Example: ISS Orbital Velocity For the International Space Station at $h = 420$ km altitude:

$$r = 6{,}371 + 420 = 6{,}791 \text{ km} \tag{B.2}$$

$$v = \sqrt{\frac{398{,}600}{6{,}791}} = 7.66 \text{ km/s} \tag{B.3}$$

$$\omega = \frac{7.66 \times 0.000785}{5 + (420 \times 0.000785)} = 0.00111 \text{ rad/s} \tag{B.4}$$

This matches the real ISS orbital velocity of approximately 7.66 km/s.

B.2.2 Elliptical Orbit Calculation

Elliptical orbits ($0 < e < 1$) use the full vis-viva equation:

$$v = \sqrt{\mu \left(\frac{2}{r} - \frac{1}{a} \right)} \tag{B.5}$$

where a is the semi-major axis and r is the instantaneous distance from Earth's center.

B.2.2.0.1 Orbital Elements Derivation Given periapsis altitude h_p and apoapsis altitude h_a:

$$r_p = R_\oplus + h_p \quad \text{(periapsis radius)} \tag{B.6}$$

$$r_a = R_\oplus + h_a \quad \text{(apoapsis radius)} \tag{B.7}$$

$$a = \frac{r_p + r_a}{2} \quad \text{(semi-major axis)} \tag{B.8}$$

$$e = \frac{r_a - r_p}{r_a + r_p} \quad \text{(eccentricity)} \tag{B.9}$$

B.2.2.0.2 Implementation Method The `OrbitController.CreateEllipticalOrbit()` method (lines 300–367) computes velocity at periapsis using Equation B.5 with $r = r_p$:

$$v_p = \sqrt{\mu \left(\frac{2}{r_p} - \frac{1}{a} \right)} \tag{B.10}$$

Table B.3 documents parameter validation constraints enforced before calculation.

TABLE B.3 – Elliptical Orbit Parameter Constraints

Parameter	**Constraint**
Periapsis altitude h_p	160–35,786 km
Apoapsis altitude h_a	$h_p + 1$ km to 100,000 km
Eccentricity e	$0 < e < 1$ (enforced implicitly)
Inclination i	0–180°

B.3 Scale Compression

The simulation implements logarithmic scale compression to fit orbital mechanics within the Meta Quest 3's comfortable rendering volume while preserving geometric relationships.

B.3.1 Compression Factor Derivation

Earth's physical radius (6,371 km) maps to 5 Unity units:

$$k = \frac{R_{\text{Unity}}}{R_{\oplus}} = \frac{5}{6{,}371} = 0.000785 \text{ Unity units/km} \tag{B.11}$$

B.3.1.0.1 Example Mappings Table B.4 shows real-world altitudes mapped to Unity rendering coordinates.

TABLE B.4 – Scale Compression Examples

Mission	Real Altitude (km)	Unity Altitude
ISS	420	$420 \times 0.000785 = 0.33$
Hubble	540	$540 \times 0.000785 = 0.42$
Geostationary	35,786	$35{,}786 \times 0.000785 = 28.1$

This compression maintains visual proportions: the ISS appears at $0.33/5 = 6.6\%$ of Earth's radius above the surface, matching the real ratio of $420/6{,}371 = 6.6\%$.

B.3.2 Numerical Stability

All physics calculations occur in real units (km, km/s) before conversion to Unity space for rendering. This ensures:

- No floating-point precision loss from working with very small Unity coordinates
- Physical accuracy verifiable against published orbital data
- Separation of physics (model) from rendering (view)

The `OrbitController` methods perform calculations in kilometers, then convert final results through multiplication by k only when setting Unity Transform positions.

B.4 Trajectory Visualization

The `OrbitVisualizer` class (280 lines) generates trajectory curves by sampling the orbital ellipse equation at discrete points and rendering through Unity's LineRenderer system.

B.4.1 Orbital Ellipse Equation

The orbit trajectory follows the polar equation:

$$r(\theta) = \frac{a(1 - e^2)}{1 + e\cos\theta} \tag{B.12}$$

where θ is the true anomaly (angle from periapsis), a is the semi-major axis, and e is the eccentricity.

B.4.2 Sampling Algorithm

The `CalculateOrbitalPoint()` method (lines 217–231) samples Equation B.12 at 128 evenly-spaced true anomaly angles $\theta \in [0, 2\pi)$. For each sample point:

$$r = \frac{a(1 - e^2)}{1 + e\cos\theta} \tag{B.13}$$

$$x = r\cos(\theta + \omega) \tag{B.14}$$

$$z = r\sin(\theta + \omega) \tag{B.15}$$

where ω is the argument of periapsis (orientation of the ellipse major axis in the orbital plane).

B.4.2.0.1 Inclination Transformation The resulting planar coordinates undergo rotation by inclination angle i via rotation matrix:

$$\mathbf{R}_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos i & -\sin i \\ 0 & \sin i & \cos i \end{bmatrix} \tag{B.16}$$

This tilts the orbital plane from equatorial (XZ) to the specified inclination angle, enabling visualization of polar orbits (ISS at 51.6°) and equatorial orbits (geostationary at 0°).

B.4.3 Rendering Configuration

Table B.5 documents the Unity LineRenderer configuration for optimal VR visibility.

B.4.3.0.1 Special Cases

TABLE B.5 – LineRenderer Configuration Parameters

Parameter	**Value**
Path resolution	128 points
Line width	0.05 Unity units
Color	Cyan (0, 1, 1) with 0.7 alpha
Shader	Sprites/Default (view-aligned billboarding)
Loop closure	Enabled (connects point 127 to point 0)

- **Circular orbits** ($e = 0$): Simplify to constant radius $r = a$, producing perfect circles
- **Elliptical orbits** ($0 < e < 1$): Render with visible eccentricity
- **Debug visualization**: Green gizmo at periapsis, red gizmo at apoapsis for development testing

B.5 Coordinate System Conventions

The simulation uses Unity's left-handed coordinate system with the following conventions:

- **Origin**: Earth's center of mass
- **Equatorial plane**: XZ plane ($y = 0$)
- **Polar axis**: +Y direction points toward North Pole
- **Reference direction**: +X axis defines 0° longitude
- **Orbital motion**: Counterclockwise when viewed from above North Pole (right-hand rule)

This convention aligns with standard aerospace engineering practices while accommodating Unity's left-handed rendering system.

Appendix C - Voice Pipeline Implementation

This appendix provides detailed technical specifications for the bidirectional voice system described in Section 3.7.3. All API endpoints, audio formats, class methods, and processing parameters are verified against the Unity project voice integration code.

C.1 System Architecture

The voice pipeline implements bidirectional audio through ElevenLabs cloud APIs, enabling natural spoken interaction with the agent system. Figure **??** conceptually illustrates the data flow (implementation in code).

C.1.0.0.1 Component Responsibilities

- **PromptConsole**: Manages microphone capture, push-to-talk input detection, and audio playback
- **ElevenLabsClient**: Handles HTTP communication with ElevenLabs APIs (381 lines)
- **Unity AudioSource**: Plays synthesized speech through Quest 3's spatial audio system
- **Unity Microphone**: Captures user voice input at 16 kHz sample rate

C.2 Speech-to-Text Pipeline

Speech recognition converts user voice input to text through ElevenLabs' Scribe v1 transcription model. Table C.1 documents the audio capture specifications.

TABLE C.1 – Speech-to-Text Audio Capture Specifications

Parameter	Value
Sample rate	16,000 Hz (optimized for speech)
Bit depth	16-bit PCM
Channels	Mono
Maximum duration	30 seconds
Audio format (transmitted)	WAV with RIFF header
API endpoint	`/speech-to-text`
Model	`scribe_v1`

C.2.1 Push-to-Talk Input Detection

Voice recording activates through push-to-talk button press. The `PromptConsole.Update()` method (lines 281–323) implements platform-specific input detection with debouncing to prevent accidental double-triggers.

C.2.1.0.1 Input Source Detection

- **Desktop testing**: Space key via `Input.GetKeyDown(KeyCode.Space)`
- **VR deployment**: Quest 3 right controller A button via `OVRInput.Get(OVRInput.Button.One, OVRInput.Controller.RTouch)`

C.2.1.0.2 State Machine Table C.2 documents the recording state transitions.

TABLE C.2 – Recording State Machine

State	Trigger	Next State
Idle	Button press	Recording
Recording	Button release	Processing
Processing	Transcription complete	Idle

During the Recording state, a red visual indicator displays "Listening..." to provide user feedback.

C.2.2 Audio Capture and Conversion

The `StartRecording()` method initiates Unity's `Microphone.Start()` with the specifications in Table C.1. When the user releases the button, `StopRecordingAndTranscribe()` processes the captured audio.

C.2.2.0.1 WAV Conversion Algorithm The `ConvertAudioClipToWav()` method (lines 322–368) converts Unity's `AudioClip` format to WAV for API transmission:

1. Extract float samples from `AudioClip.GetData()`
2. Convert float [-1.0, 1.0] to 16-bit signed integer [-32768, 32767]
3. Construct RIFF WAV header (44 bytes):
   - Chunk ID: "RIFF"
   - Format: "WAVE"
   - Subchunk 1: "fmt " (audio format specification)
   - Subchunk 2: "data" (PCM samples)
4. Concatenate header + PCM data

C.2.3 API Request Structure

The WAV bytes transmit to ElevenLabs via `WWWForm` multipart HTTP POST:

```
POST https://api.elevenlabs.io/v1/speech-to-text
Content-Type: multipart/form-data
Headers: xi-api-key: [API_KEY]

Body:
  - file: recording.wav (binary WAV data)
  - model_id: "scribe_v1"
```

C.2.4 Response Parsing

The API returns JSON containing:

- **text**: Transcribed text string
- **confidence**: Recognition confidence score [0.0–1.0]

The transcribed text feeds directly into the agent's `ProcessUserInput()` method for intent interpretation and tool selection.

C.3 Text-to-Speech Pipeline

Agent text responses convert to speech through ElevenLabs' text-to-speech API. Table C.3 documents the synthesis configuration.

TABLE C.3 – Text-to-Speech Synthesis Parameters

Parameter	Value
Model	`eleven_flash_v2_5`
Stability	0.7 (voice consistency)
Similarity boost	0.8 (voice clarity)
Speed	1.0 (normal playback)
Audio format (received)	MP3
API endpoint	`/text-to-speech/{voiceId}`
Synthesis latency	1–3 seconds (typical)

C.3.1 API Request Structure

The `TextToSpeechAsync()` method (lines 31–129) sends synthesis requests:

```
POST https://api.elevenlabs.io/v1/text-to-speech/{voiceId}
Content-Type: application/json
Headers: xi-api-key: [API_KEY]

Body:
{
  "text": "Agent response text here",
  "model_id": "eleven_flash_v2_5",
  "voice_settings": {
    "stability": 0.7,
    "similarity_boost": 0.8,
    "speed": 1.0
  }
}
```

C.3.2 MP3 Decoding and Playback

The API returns MP3-encoded audio via HTTP response body. The `ConvertMp3ToAudioClipAsyn` method (lines 135–154) performs decoding:

1. Write MP3 bytes to temporary file in `Application.temporaryCachePath`

2. Load via `UnityWebRequestMultimedia.GetAudioClip(uri, AudioType.MPEG)`
3. Enable streaming mode for memory efficiency
4. Extract `AudioClip` from request
5. Delete temporary file in `finally` block

The resulting `AudioClip` plays through Unity's `AudioSource` component attached to the camera, utilizing Quest 3's spatial audio capabilities for immersive voice delivery positioned at the user's head location.

C.3.3 Model Selection Rationale

The `eleven_flash_v2_5` model balances:

- **Synthesis speed**: 1–3 seconds for typical 2–3 sentence responses (critical for real-time interaction)
- **Voice fidelity**: Natural prosody and intonation
- **API cost**: Flash models optimize for speed over maximum quality

C.4 Character Voice Management

Each agent character uses a distinct ElevenLabs voice ID configured in `MissionConfig.specialistVoice` ScriptableObject references. Table C.4 documents character voice assignments.

TABLE C.4 – Character Voice ID Assignments

Character	Voice ID	Characteristics
Mission Control	`NOpBlnGInO9m6vDvFkFC`	Authoritative, encouraging, professional
ISS Specialist (Anastasia)	[Scene-configured]	Professional engineer, clear, technical, friendly
Hubble Specialist	[Scene-configured]	[Character-specific]
Voyager Specialist	[Scene-configured]	[Character-specific]

C.4.1 Scene-Specific Voice Switching

When users invoke the `route_to_mission` tool, the scene transition loads mission-specific `ElevenLabsSettings` assets that override the default voice ID. This ensures:

- Hub agent responses use Mission Control voice
- ISS Mission Space responses use Anastasia's voice profile
- Each specialist maintains consistent vocal identity

Voice synthesis parameters (stability, similarity boost, speed) remain constant across all characters to maintain audio quality consistency, while the underlying voice models provide tonal and character differentiation.

C.4.2 Voice Settings Persistence

The `ElevenLabsClient` caches the current `ElevenLabsSettings` reference. Scene transitions update this reference automatically through Unity's scene loading hooks, enabling seamless character voice switching without code changes in the agent logic.

C.5 Error Handling and Fallbacks

The voice pipeline implements robust error handling for network failures and API timeouts:

C.5.0.0.1 Speech-to-Text Errors

- **Microphone unavailable**: Display error message, fall back to text input
- **API timeout**: Retry once with exponential backoff, then show error
- **Low confidence score**: Accept transcription but log warning

C.5.0.0.2 Text-to-Speech Errors

- **API timeout**: Display text response without audio
- **MP3 decode failure**: Log error, display text fallback
- **AudioSource unavailable**: Silent failure, text remains visible

All errors log to Unity console with structured error messages for debugging while maintaining graceful degradation of user experience.

C.6 Performance Optimization

C.6.1 Memory Management

- Temporary WAV/MP3 files deleted immediately after use
- `AudioClip` instances released when playback completes
- Streaming mode for MP3 decoding reduces peak memory usage
- No audio caching (prioritizes memory over latency)

C.6.2 Latency Budget

Table C.5 documents typical latency components for the complete voice interaction cycle.

TABLE C.5 – Voice Interaction Latency Budget

Component	**Latency**
User speech duration	Variable (user-controlled)
WAV conversion	< 100 ms
STT API request	500–1500 ms
Agent processing (GPT-4.1)	1000–3000 ms
TTS API request	1000–3000 ms
MP3 decode	< 200 ms
Audio playback start	< 50 ms
Total (excluding user speech)	**2.5–7.8 seconds**

The 2.5–7.8 second response time falls within acceptable bounds for educational conversational interfaces, where thoughtful responses outweigh instantaneous feedback.

Appendix D - VR Deployment Configuration

This appendix provides detailed technical specifications for the Meta Quest 3 virtual reality deployment described in Section 3.7.4. All build settings, input mappings, rendering configurations, and scene architecture details are verified against the Unity project configuration files.

D.1 Quest 3 Android Build Configuration

The application deploys to Meta Quest 3 through Unity's Android build pipeline with OpenXR integration. Table D.1 documents the core build settings from `ProjectSettings.asset`.

TABLE D.1 – Android Build Configuration

Setting	Value
Minimum SDK Version	32 (Android 12L)
Target SDK Version	32 (Android 12L)
Target Architecture	ARMv7 (value: 2)
Graphics API	OpenGL ES 3.0
XR Plugin	OVRPlugin (Oculus SDK)
Stereo Rendering Mode	Single Pass Instanced (value: 2)
Target Device	Meta Quest 3

D.1.1 SDK Version Rationale

Android API level 32 (Android 12L) enables:

- Quest 3's inside-out tracking system (6DOF head and controller tracking)
- Oculus runtime features (Guardian boundary, passthrough API access)
- Hand tracking capabilities (though not actively used in this application)

- Performance optimizations for Snapdragon XR2 Gen 2 processor

D.1.2 Stereo Rendering Pipeline

Single-pass instanced rendering (value 2 in `ProjectSettings.asset` line 49) reduces CPU overhead by rendering both eye views in a single draw call. This technique:

- Halves per-frame CPU work compared to multi-pass rendering
- Maintains Quest 3's 90 Hz refresh rate target
- Reduces GPU state changes and draw call overhead
- Critical for mobile VR performance on battery-powered hardware

D.1.3 Build Index Scene Configuration

Table D.2 documents the scene inclusion from `EditorBuildSettings.asset`.

TABLE D.2 – Scene Build Index Configuration

Index	Scene Name	File Size
0	Hub.unity	85 KB
1	ISS.unity	65 KB
2	Hubble.unity	66 KB
3	Voyager.unity	62 KB
4	ARHub.unity (experimental)	61 KB

Scene index 0 (Hub) loads at application startup. Scene transitions occur through `SceneManager.LoadSceneAsync()` with scene names or indices.

D.2 Input System Implementation

Controller input integrates Oculus Touch controllers through the OVR Input API. Table D.3 documents the input bindings used in the application.

TABLE D.3 – Controller Input Mapping

Action	Desktop	Quest 3 VR
Push-to-talk (voice)	Space key	Right controller A button
Confirm/Select	Enter key	Right controller trigger
Cancel/Back	Escape key	Left controller B button

D.2.1 Push-to-Talk Implementation

The `PromptConsole.Update()` method (line 283) detects the right controller's A button through OVR Input API:

```
bool aButtonPressed = OVRInput.Get(
    OVRInput.Button.One,
    OVRInput.Controller.RTouch
);
```

D.2.1.0.1 State Debouncing The system tracks previous button state (`_previousAButtonState`) to detect rising edge transitions, preventing accidental double-triggers from single button presses. This ensures one recording session per button press/release cycle.

D.2.2 Desktop Testing Mode

Desktop mode falls back to keyboard input through Unity's legacy Input system:

```
bool spacePressed = Input.GetKeyDown(KeyCode.Space);
```

This enables development iteration without VR hardware, maintaining identical functionality across desktop testing and Quest 3 deployment.

D.2.3 VR Mode Detection

The `StaticVRCameraAligner` class (89 lines) detects VR mode at startup through:

```
bool isVR = XRSettings.isDeviceActive;
```

When `isDeviceActive` returns `true`, the system locates the `OVRCameraRig` component via `FindObjectOfType<OVRCameraRig>()` and configures VR-specific camera settings.

D.3 Camera and Rendering Configuration

D.3.1 OVRCameraRig Structure

The Quest 3 camera system follows Oculus SDK conventions. Table D.4 documents the camera hierarchy.

TABLE D.4 – VR Camera Hierarchy

GameObject	Purpose
OVRCameraRig	Root container for VR camera system
TrackingSpace	Offset container for room-scale tracking
CenterEyeAnchor	Head-tracked camera position (stereo)
LeftEyeAnchor	Left eye render camera
RightEyeAnchor	Right eye render camera
LeftHandAnchor	Left controller tracking
RightHandAnchor	Right controller tracking

D.3.2 Near Clip Plane Configuration

The `StaticVRCameraAligner` configures the near clip plane to prevent geometry clipping at close range (line 68):

```
cam.nearClipPlane = 0.01f; // Unity units
```

This 0.01 Unity unit near clip (approximately 1.27 cm in physical space with scale compression factor $k = 0.000785$) ensures UI elements positioned within arm's reach remain visible without clipping.

D.3.3 Desktop Camera Alignment

Desktop mode aligns the fallback camera to match VR positioning conventions, ensuring consistent coordinate systems between development and deployment environments. This allows testing of UI positioning and scene layout without VR hardware.

D.4 Spatial UI Implementation

User interface elements render in 3D world space rather than screen overlay to ensure VR readability and depth perception. Table D.5 documents the UI rendering configuration.

D.4.1 MissionClockUI Pattern

The `MissionClockUI` class (74 lines) demonstrates the spatial UI pattern:

1. Canvas component with `RenderMode.WorldSpace`
2. TextMeshPro text field positioned in 3D environment

TABLE D.5 – Spatial UI Rendering Configuration

Parameter	Value
Canvas render mode	WorldSpace
Canvas size (transition overlay)	2m × 2m
Canvas distance from camera	1 meter (dynamic)
Mission logo size	512×512 pixels
Text component	TextMeshPro
Background opacity (UI panels)	0.7 alpha
Sort order (transition canvas)	100 (renders on top)

3. CanvasGroup component (line 37) controls opacity without render-to-texture overhead

4. Displays mission elapsed time and simulation speed multiplier

D.4.2 Transition Overlay System

The `SceneTransitionManager.CreateTransitionUIIfNeeded()` method (lines 729–838) constructs the transition overlay procedurally:

D.4.2.0.1 Canvas Construction

```
Canvas canvas = canvasObj.AddComponent<Canvas>();
canvas.renderMode = RenderMode.WorldSpace;
canvas.sortingOrder = 100;

RectTransform canvasRect = canvasObj.GetComponent<RectTransform>();
canvasRect.sizeDelta = new Vector2(2f, 2f); // 2m × 2m
```

D.4.2.0.2 Dynamic Positioning The `LateUpdate()` method (lines 154–180) repositions the canvas 1 meter in front of the camera each frame:

```
transitionCanvasTransform.position =
    cachedCameraAnchor.position +
    cachedCameraAnchor.forward * 1f;

transitionCanvasTransform.rotation =
    cachedCameraAnchor.rotation;
```

This dynamic positioning ensures the overlay remains visible during scene transitions

when camera references change, avoiding parenting to scene-specific GameObjects that would be destroyed during `SceneManager.LoadSceneAsync()`.

D.5 Scene Architecture and Persistence

The application comprises four navigable scenes sharing common systems through persistent singletons. Table D.6 documents shared components across all scenes.

TABLE D.6 – Common Scene Components

Component	Purpose
OVRCameraRig prefab	VR camera and controller tracking
PromptConsole GameObject	Conversational UI and voice input
TimeController	Simulation speed management
OrbitController	Orbital physics (Hub only)
OrbitVisualizer	Trajectory rendering (Hub only)

D.5.1 Singleton Persistence Mechanism

The `SceneTransitionManager` enforces singleton persistence through Unity's `DontDestroyOnLoad()` mechanism (line 58):

```
if (Instance == null) {
    Instance = this;
    DontDestroyOnLoad(gameObject);
}
```

This ensures the transition UI and conversation context survive scene unloading. Similarly, `ConversationHistory` persists across transitions, preserving the 10-exchange dialogue window.

D.5.2 Asynchronous Scene Loading

Scene loading occurs through `SceneManager.LoadSceneAsync()` with deferred activation (line 252):

```
AsyncOperation loadOperation =
    SceneManager.LoadSceneAsync(sceneName);
loadOperation.allowSceneActivation = false;
```

```
// Load scene in background...

// After 4-second logo animation:
loadOperation.allowSceneActivation = true;
```

This deferred activation prevents jarring scene pops, allowing smooth fade-out → logo display → scene activation → fade-in transitions.

D.6 Performance Optimization

Performance optimization targets Quest 3's mobile GPU constraints. Table D.7 documents the performance budget.

TABLE D.7 – Performance Targets for 90 Hz VR

Metric	**Target**
Frame time budget	11.1 ms (90 Hz)
Target resolution (per eye)	1832×1920 pixels
Draw calls (Hub scene)	< 100 per frame
Texture memory budget	< 512 MB
Polygon count (visible)	< 100k triangles

D.6.1 Rendering Optimizations

- **Shared material instances**: Reduce draw calls by batching geometry with identical materials
- **Texture compression**: ASTC 6×6 for UI elements, ASTC 4×4 for environment textures
- **Single-pass instanced stereo**: Halves per-frame CPU work (both eyes in one draw call)
- **Occlusion culling**: Disabled (scenes are spatially compact, overhead exceeds benefit)
- **Dynamic batching**: Enabled for small meshes (< 300 vertices)

D.6.2 Memory Management

- Scene file sizes optimized (62–85 KB per scene)
- Texture atlasing for UI sprites

- Audio streaming for voice synthesis (no large audio caching)
- Persistent GameObjects minimized (only transition manager, conversation history)

D.6.3 Frame Time Breakdown

Table D.8 shows typical frame time allocation in the Hub scene (most complex).

TABLE D.8 – Frame Time Budget Breakdown (Hub Scene)

Component	**Time**
Physics simulation	1.2 ms
Script execution	2.1 ms
Rendering (both eyes)	5.8 ms
VR compositor overhead	1.5 ms
Buffer margin	0.5 ms
Total	**11.1 ms**

This allocation maintains the 11.1 ms frame budget required for consistent 90 Hz VR without reprojection artifacts (judder).

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
TC	25 de março de 2015	DCTA/ITA/DM-018/2015	68

5. TÍTULO E SUBTÍTULO:

Educational Orbit Simulation with Generative AI Agentic Workflow and Virtual Reality Visualisation

6. AUTOR(ES):

Eduardo Moura Zindani

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

AI; VR

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

AI; VR

10. APRESENTAÇÃO: (**X**) **Nacional** ( ) **Internacional**

ITA, São José dos Campos. Curso de Mestrado. Programa de Pós-Graduação em Engenharia Aeronáutica e Mecânica. Área de Sistemas Aeroespaciais e Mecatrônica. Orientador: Prof. Dr. Adalberto Santos Dupont. Coorientadora: Prof[a]. Dr[a]. Doralice Serra. Defesa em 05/03/2015. Publicada em 25/03/2015.

11. RESUMO:

Métodos educacionais tradicionais frequentemente encontram dificuldades para transmitir conceitos complexos, espaciais e dinâmicos como os da mecânica orbital. A recente convergência entre a Realidade Mista (RM) de consumo e os sofisticados agentes de Inteligência Artificial (IA) Generativa apresenta uma oportunidade para criar um novo paradigma de interfaces de aprendizagem intuitivas e experienciais. Este trabalho detalha o projeto, desenvolvimento e demonstração de uma plataforma educacional interativa para a exploração dos princípios da mecânica orbital. O objetivo principal do sistema é conectar a teoria de leis físicas abstratas à compreensão intuitiva, permitindo que os usuários aprendam por meio da interação corporificada. A metodologia é centrada em uma arquitetura modular que integra dois componentes principais: (1) um ”cérebro” agente generativo, impulsionado por Modelos de Linguagem Abrangentes, que interpreta comandos em linguagem natural e atua como um guia educacional especializado; e (2) um ”mundo” de simulação em tempo real e visualização imersiva em realidade mista, construído no motor Unity para o Meta Quest 3, que renderiza trajetórias orbitais fisicamente precisas em um espaço tridimensional onde os usuários vivenciam a mecânica orbital de dentro. A plataforma facilita um ciclo de interação multimodal contínuo, onde os comandos de voz do usuário são capturados, processados pelo agente para alterar os parâmetros da simulação e refletidos na visualização imersiva em RV com feedback auditivo conversacional. Este trabalho entrega um protótipo funcional que demonstra uma nova abordagem para a educação científica, transformando dados abstratos em uma experiência manipulável e conversacional para promover uma aprendizagem exploratória e profundamente engajadora. A plataforma é disponibilizada como software de código aberto para permitir validação, adaptação e extensão pela comunidade para diversos contextos educacionais.

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **SECRETO**