POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Tesi Magistrale

Introduzione del concetto di tempo all'interno di una simulazione VR dell'universo

Relatore **Candidato**

Prof. Andrea SANNA **Simone TERZUOLO**

Luglio, 2021

Ringraziamenti

Giunto alla conclusione del percorso di studi intrapreso, desidero ringraziare tutti coloro che mi hanno supportato nel raggiungimento di un simile traguardo.

Innanzitutto vorrei esprimere la mia gratitudine al mio relatore, il professor Andrea Sanna, per i consigli e la disponibilità datami nel corso della stesura della tesi.

Vorrei, inoltre, ringraziare ALTEC per avermi dato la possibilità di lavorare ad un progetto così ambizioso nonostante il difficile periodo di pandemia. Un ringraziamento speciale è rivolto ad Eugenio Topa, capace di supervisionare costantemente il mio lavoro dedicandomi molto del suo prezioso tempo con consigli ed insegnamenti che mi hanno permesso di crescere professionalmente nel corso di questi mesi. Desidero ringraziare anche tutti gli altri membri del team che sono stati capaci di creare un ambiente lavorativo sereno, rilassante ma allo stesso tempo estremamente professionale.

Ringrazio il mio collega Paolo Gallo, con cui ho sviluppato *Astra Data Navigator* ed ho condiviso la maggior parte del tempo in ALTEC.

Un grazie molto importante va rivolto a tutti i miei amici, con cui ho collezionato una serie infinita di storie da raccontare, alcune particolarmente degne di nota.

Infine vorrei ringraziare la mia famiglia che mi ha sostenuto ed incoraggiato durante questi lunghi anni di studi, con particolare attenzione ai miei nonni poiché senza di loro tutto questo non sarebbe stato possibile. A Sara dedico il mio ultimo e più importante ringraziamento per avermi aiutato a superare ogni ostacolo incontrato, avermi ascoltato parlare ancora e ancora la sera di quello fatto durante la tesi, ma in generale per starmi accanto e rendere uniche le mie giornate da ormai molti anni.

Abstract

Il progetto di tesi si colloca nell'ambito della Realtà Virtuale, ponendosi come principale obiettivo quello di estendere la già esistente versione di *Astra Data Navigator*, una simulazione virtuale dell'universo, introducendone all'interno il concetto di tempo mediante l'utilizzo degli *SPICE kernel*, strumento sviluppato dal *Navigation and Ancillary Information Facility (NAIF)* sotto la direzione del *Planetary Science Division* della *NASA*, capace di calcolare le precise posizioni della maggior parte dei corpi celesti conosciuti.

L'applicazione è stata realizzata nel laboratorio VR di *ALTEC - Aerospace Logistics Technology Engineering Company* - all'interno del *NEANIAS European project*. Gli *SPICE* sono tra i pilastri fondamentali per la pianificazione missioni spaziali, nonostante ciò non sono presenti molte applicazioni VR in grado di dargli un volto grafico.

L'applicazione offre all'utente tre possibili modalità di calcolo delle posizioni nel tempo, da scegliere basandosi sulle disponibilità di risorse a disposizione e la precisione desiderata. Due dei tre metodi di rivoluzione si basano completamente sull'utilizzo degli *SPICE*, con la differenza che una soluzione sfrutta una libreria interna, sviluppata sempre nel progetto di tesi, basata sul codice di *NAIF*, mentre l'altra è in grado di connettere ad un servizio web capace di esporne le principali funzionalità. Tali soluzioni sono capaci di mostrare all'utente la precisa evoluzione temporale delle posizioni dei corpi celesti del sistema solare presenti all'interno della scena virtuale. Il terzo metodo si basa solo parzialmente sull'utilizzo degli *SPICE*, sfruttati in fase di creazione dei cataloghi caricati dall'applicazione, poiché la rivoluzioni è pilotata dalle equazioni matematiche formulate da *Keplero*, il questo caso non è possibile ottenere grande precisione ma solamente un'idea realistica di come i corpi ruotano sui loro assi.

Ottenere la precisione richiesta in tempo reale può essere un compito arduo, tuttavia sono state sviluppate tecniche di gestione dei calcoli e dei dati capaci di offrire anche la possibilità di velocizzare la simulazione, in grado di non perdere mai di vista l'obiettivo richiesto. Tali tecniche sono ampiamente descritte all'interno di questo elaborato.

In questo studio vengono anche presentati analisi di performance e test soggettivi della versione finale dell'applicazione, capaci di evidenziare come la maggior parte degli obiettivi del progetto siano stati portati a termine con successo, facendone inoltre risaltare alcune carenze che potrebbero trovare soluzione in lavori futuri.

Indice

Elenco delle figure

Elenco delle tabelle

Glossario

astrofili dilettanti appassionati di studi astronomici.

dataset insieme di dati strutturati in forma relazionale.

dll dynamic linked library, indici una libraria che viene caricata dinamicamente in fase di esecuzione.

driver insieme di procedure software, spesso scritte in assembly, che permette ad un sistema operativo di pilotare un dispositivo hardware.

framerate frequenza alla quale immagini consecutive compaiono sullo schermo.

framework piattaforma che funge da strato intermedio tra un sistema operativo ed il software che lo utilizza.

frustum regione di spazio del mondo modellato che può apparire sullo schermo, ovvero il campo visivo di un sistema di telecamere virtuali prospettiche.

headset Un headset è uno strumento da indossare in testa in modo da poter vedere immagini rimprese da una camere direttamente davanti agli occhi.

jitter indica la variazione di una o più caratteristiche di un segnale.

kernel Componente fondamentale per il calcolo delle geometrie spaziali contenete dati di basso livello.

motion capture processo di registrazione dei movimenti di un oggetto o di una persona, convertendo queste informazioni nell'animazione di soggetti virtuali.

plugin programma non autonomo che amplia o estende le funzionalità originarie di un altro programma.

raycasting tecnica alla base del rendering di immagini, consiste nel tracciare raggi di luce virtuale partendo dal centro focale della camera.

refresh rate numero di volte in un secondo in cui viene ridisegnata l'immagine su un display.

scattering diffusione ottica, si intende un'ampia classe di fenomeni di interazione radiazione-materia in cui onde o particelle vengono deflesse a causa della collisione con altre particelle o onde.

shader insieme di algoritmi usati soprattutto in grafica computerizzata 3D che conferiscono al materiale virtuale a cui sono abbinati delle caratteristiche o proprietà che ne descrivono o ne influenzano il modo di reagire alla luce.

spacecraft Veicolo in grado di viaggiare nello spazio.

texture immagine bidimensionale in formato raster che viene riprodotta su una o più facce di un modello poligonale tridimensionale.

toolkit insieme di strumenti software di base, in genere librerie, usati per facilitare e uniformare lo sviluppo di applicazioni derivate più complesse.

tooltip comune elemento dell'interfaccia grafica dell'utente. Attivato dal cursore, di solito il puntatore del mouse, passando quest'ultimo sopra un oggetto, senza cliccarlo, si produce l'apertura di un piccolo "box" con informazioni supplementari riguardo all'oggetto stesso.

wrapper modulo software che ne "riveste" un altro, ossia che funziona da tramite fra i propri clienti, che usano l'interfaccia del wrapper, ed il modulo rivestito, che svolge effettivamente i servizi richiesti, su delega dell'oggetto wrapper.

Acronimi

2D bidimensionale.

3D tridimensionale.

ADN Astra Data Navigator.

AI Intelligenza Artificiale.

API Application Programming Interface.

AR Realità Aumentata.

CAVE Cave Automatic Virtual Environment.

CPU Central Processing Unit.

ESA European Space Agency.

FOV Field of View.

FPS Frames per Second.

GPU Graphics Processing Unit.

GUI Graphical User Interface.

HMD Head-Mounted-Display.

IPD distanza interpupillare.

ISS International Space Station.

JSON JavaScript Object Notation.

MIT Massachusetts Institute of Technology.

NASA National Aeronautics and Space Administration.

PMS Position Manager Service.

RAM Random Access Memory.

SGT Space Graphics Toolkit.

SPK Spacecraft and Planet Kernel.

TDB Barycentric Dynamical Time.

UI User Interface.

URL Uniform Resource Locator.

UTC Tempo Coordinato Universale.

VR Realtà Virtuale.

Introduzione

Le simulazioni e la Realtà Virtuale rendono visibile l'invisibile. Partendo da questo presupposto, è ora possibile visitare ciò che prima si poteva solo descrivere a parole, come lo spazio, oppure rimpicciolirsi fino ad osservare il mondo subatomico.

La Realtà Virtuale permette di imparare attraverso situazioni reali generate artificialmente. Quest'opportunità viene sfruttata quando non è possibile effettuare delle reali esperienze. Con la Realtà Virtuale possiamo scoprire, esplorare e costruirci una cultura di luoghi e situazioni che altrimenti sarebbero inaccessibili. La grande potenzialità di questa tecnologia risiede anche nella possibilità di manipolare virtualmente ciò che viene esplorato, analizzato o studiato [1].
La VR permette agli studenti di comprendere temi complessi che spesso risultano impossibili da visualizzare attraverso i canonici metodi di insegnamento.

Le esperienze sono una parte fondamentale per la costruzione di modelli mentali dei concetti che si vogliono apprendere, tutto ciò sta alla base della conoscenza. Gli umani imparano mediante le esperienze e interagendo con l'ambiente circostante utilizzando i sensi al fine di captare informazioni dal mondo [2]. La Realtà Virtuale è la tecnologia che permette di replicare le sensazioni derivate dal mondo reale con quelle create attraverso un computer.

L'esperienza naturale di un sistema VR deriva da tre sorgenti:

- ***immersività***, ovvero la capacità del sistema di coinvolgere attraverso l'ambiente, assicurando all'utente il senso di presenza e la sensazione di trovarsi realmente nel mondo che lo circonda;
- ***interattività***, o concetto di azione-reazione, l'abilità di controllare eventi all'interno della simulazione mediante i movimenti del corpo che a loro volta scaturiscono in reazioni nella simulazione riproducendo l'esatto comportamento del mondo reale;
- ***feedback multi-sensoriale***, le informazioni che percepisce l'utente devono derivare da più sensi in modo da accrescere ulteriormente il senso di presenza ed eliminare eventuale ambiguità e confusione, solitamente il rinforzo sensoriale di informazioni proviene da due o tre sorgenti.

Lo scopo della Realtà Virtuale è quello di sostituire il mondo reale con una sua versione virtuale e permettere all'utente di comportarsi esattamente come se si trovasse in quello reale.

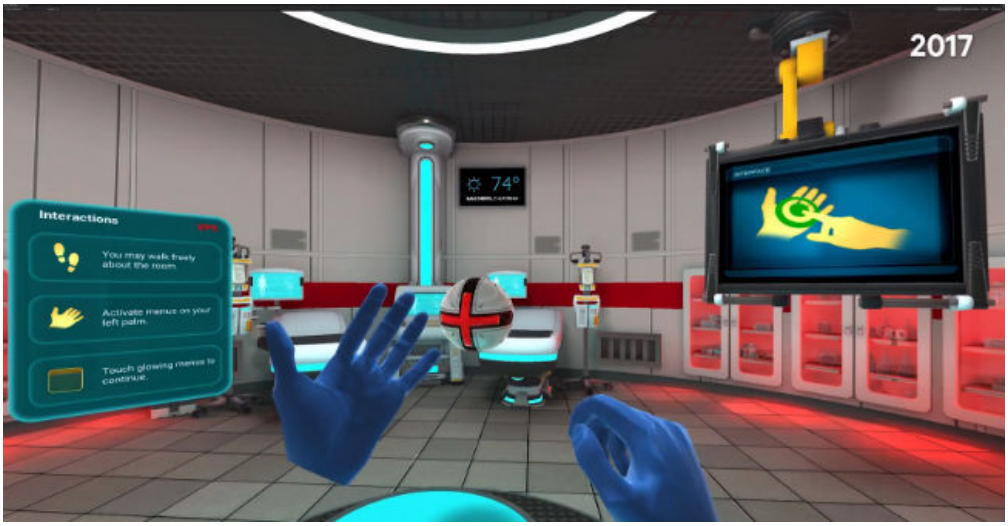


Figura 1: Esempio di ambiente in VR

Le basi di quella che oggi chiamiamo Realtà Virtuale vennero gettate nei primi anni sessanta quando il regista Morton Heiling brevettò una macchina chiamata *Sensorama* basandosi su quello che veniva definito il "cinema di esperienza", una tipologia di rappresentazione che coinvolgesse tutti i sensi dello spettatore in maniera efficacie.

Uno dei più noti HMD della storia fu quello che oggi viene ricordato come *"La Spada di Damocle"* a causa della necessità di essere ancorato al soffitto per essere indossato, il dispositivo era capace di tracciare sia la posizione dell'utente che quella dei suoi occhi adattando l'immagine stereoscopia proiettata [3].

Il termine Realtà Virtuale nacque negli anni '80 e si diffuse grazie ad uno dei pionieri di questo campo, Jaron Lanier, il quale con la propria azienda sviluppò diversi dispositivi VR quali Data Glove, Eye Phone e Audio Sphere.
Nel corso degli anni si è assistito ad una incremento repentino delle applicazioni destinate alla Realtà Virtuale e come conseguenza anche delle tecnologie in grado di supportarle. HTC, Valve Corporation, Playstation VR e Oculus VR sono solamente alcuni esempi di società che popolano il florido mondo della Realtà Virtuale, capaci di sviluppare visori e sensori sempre più all'avanguardia.

Un Sistema di Realtà Virtuale necessita di diversi componenti sia hardware che software per funzionare correttamente:

- ***Head-Mounted-Display (HMD)***, una delle prime immagini che vengono in mente quando si pensa ad un sistema VR, dispongono di display stereoscopici e sistemi di tracking integrati i quali permettono all'utente di estendere il proprio campo visivo rendendolo in grado di osservare l'intero mondo virtuale che lo circonda. Esistono delle versioni orientate ad applicazioni da gaming, ne è un esempio l'*Oculus Rift* (Figura 2), il quale processa dati provenienti da giroscopi, accelerometri e magnetometri.

- ***Cave Automatic Virtual Environment (CAVE)***, stanze per la Realtà Virtuale composte da proiettori capaci di coprire più pareti con immagini stereoscopiche. All'utente viene richiesto l'utilizzo di particolari occhialini sincronizzati con le immagini. Sono inoltre dotate di altoparlanti posti in punti strategici in grado di generare nell'utente la sensazione di audio tridimensionale.

- ***Dispositivi di Input***, provvedono a captare dall'utente le informazioni necessarie al fine di essere elaborate e convertite in azioni e reazioni all'interno dell'ambiente virtuale. I *wired gloves* consentono di misurare le articolazioni, gli angoli, la pressione e la posizione delle mani dell'utente. Sono inoltre necessari per il sistema di feedback aptico utilizzato per stimolare nell'utilizzatore le stesse sensazioni che proverebbe nel mondo reale. Possono essere prodotti sfruttando le fibre ottiche, un inchiostro conduttivo capace di misurare la resistenza elettrica oppure dei sensori meccanici.
  Le *wands* attraverso numerosi trasmettitori ad infrarossi posti al loro interno permettono di interpolare la posizione ad il movimento dell'utente che le tiene in mano. Questo tipo di controller sono equipaggiati con giroscopi ed accelerometri al fine di rendere più precisa la trasposizione del movimento nel mondo virtuale. I *Playstation Eye* (Figura 2) sono dotati, inoltre, di magnetometri per calcolare il campo magnetico della Terra utilizzato per calibrare i sensori di inerzia dei controller.
  Sempre più diffusa la *Computer Vision* quale dispositivo di input, questa tecnologia permette l'uso di una camera in grado di riconoscere un modello ed identificarne il movimento. Il *Kinect* ne è un esempio (Figura 2).

Figura 2: In ordine da sinistra verso destra: *Playstation Eye*, *Kinect*, *Oculus Rift*

- ***Framework VR***, per lo sviluppo di applicazioni di Realtà Virtuale vengono spesso utilizzati dei *Game Engine*, nel caso del progetto di tesi si è utilizzato Unity (paragrafo 2.1), essi semplificano il lavoro degli sviluppatori in quanto gestiscono il rendering 3D, l'AI, ma sopratutto la fisica che permette di rendere credibile il mondo generato virtualmente attraverso, ad esempio, il corretto comportamento nella collisione tra oggetti.

Ad oggi la Realtà Virtuale potrebbe essere applicata a qualunque area nella quale vengono utilizzati i computer, tuttavia gli ambiti più comuni attualmente coinvolti sono:

- ***Educazione***, le più note sono le applicazioni volte all'insegnamento della fisica che permettono agli studenti di interagire con il mondo attraverso mani virtuali con cui afferreranno oggetti regolati dalle reali leggi della fisica. Esistono simulazioni incentrate sulla visualizzazione e la spiegazione dei moti in cui si possono lanciare oggetti in aria e sui muri per comprendere meglio i moti elastici o la forza di gravità, quelle rivolte al magnetismo in cui si possono posizionare cariche elettriche nell'aria per osservare le interazioni tra esse, oppure osservare da vicino la struttura atomica di cui è fatta la materia.

- ***Intrattenimento***, ogni gioco per computer moderno presenta gli aspetti base della Realtà Virtuale: simulazione, immersione e interazione. Tuttavia non ci si limita al campo videoludico, basti pensare ad i cinema progettati per garantire agli spettatori esperienze uniche che coinvolgano maggiormente i sensi.

- ***Militare***, ben noto in questo settore l'utilizzo di simulatori per l'addestramento. Si spazia dai simulatori di volo fino ad arrivare a quelli sottomarini in cui vengono riprodotte situazioni reali nelle quali occorre eseguire corrette procedure al fine di portare a termine il compito assegnato.

La Realtà Virtuale nell'industria aerospaziale

AR e VR, ormai da molti anni, vengono sfruttate da numerose aziende aerospaziali al fine di riprodurre e simulare missioni spaziali. Viene fatto un largo utilizzo di queste tecnologie per l'addestramento degli astronauti, i quali possono esercitarsi a svolgere i compiti che saranno in futuro chiamati ad eseguire all'interno delle stazioni spaziali, oppure abituarsi alle camminate spaziali e alla vita a gravità zero, tutto ciò all'interno di ambienti controllati e privi di rischi [4].
Modellando opportunamente il sistema è possibile riprodurre con estrema fedeltà qualunque compito, dai più semplici a quelli più complessi. Ciò comporta un'innovazione fondamentale all'interno di quest'industria che è ora in grado di preparare al meglio i propri astronauti abbattendo rischi e costi con eccellenti risultati.

Uno dei primi utilizzi della VR all'interno dell'industria aerospaziale risale alla fine degli anni '80 con il *Virtual Interface Environment Workstation (VIEW)* (figura 3) un HMD stereoscopico nel quale i display potevano mostrare un ambiente generato a computer oppure utilizzare le immagini fornite da telecamere remote [5].

L'operatore poteva letteralmente stare all'interno di questo ambiente ed interagire con esso, venivano inoltre utilizzati dei *DataGlove*, ovvero dei guanti dotati di fibre ottiche collegate ad un computer in grado di captare e riprodurre con precisione i movimenti delle mani all'interno della simulazione.

Figura 3: *NASA* - Virtual Interface Environment Workstation (VIEW)

Sempre di *NASA* è la simulazione 3D della camera a vuoto termico di Goddard[1] per determinare se i componenti dello Spacecraft si adatterebbero o meno all'interno della struttura prima che la fase di test inizi. Altro utilizzo dell'applicazione consiste nell'addestramento all'utilizzo dei bracci robotici presenti all'interno dell'*International Space Station (ISS)*. Grazie all'utilizzo della VR e dell'AR tutte le informazioni necessarie all'operatore vengono visualizzate in tempo reale all'interno del suo FOV allertandolo di eventuali problemi prima che questi insorgano [6].

Il progetto di tesi

La tesi è stata svolta presso l'*ALTEC - Aerospace Logistics Technology Engineering Company* [7], la quale è il centro di eccellenza italiano per la fornitura di servizi ingegneristici e logistici a supporto delle operazioni e dell'utilizzazione della Stazione Spaziale Internazionale, dello sviluppo e della realizzazione delle missioni di esplorazione planetaria.

ALTEC prende parte al *NEANIAS European project* [8] il quale ha come scopo quello di produrre servizi tematici per la comunità scientifica in grado di adattarsi in diversi ambiti favorendone la riproducibilità e la riutilizzabilità. Proprio all'interno di questo contesto nasce *Astra Data Navigator* al fine di poter rappresentare in VR il comportamento nel tempo di corpi celesti quali pianeti e satelliti.

[1] camera a vuoto in cui vengono controllate le condizioni termiche radiative

L'obiettivo del progetto è quello di estendere il già esistente *Astra Data Navigator* sviluppando un sistema in grado di utilizzare i dati forniti dagli *SPICE Kernels* con i quali è possibile stabilire con estrema precisione la posizione dei corpi celesti nel tempo, tutto ciò all'interno di una simulazione real-time utilizzando il game engine Unity.

Le applicazioni VR hanno la caratteristica di essere immersive, è quindi necessario stimolare contemporaneamente più sensi in modo da creare all'utente l'illusione di trovarsi davvero in un altro mondo. Da questo presupposto si deduce facilmente che la progettazione e la realizzazione di ambienti in Realtà Virtuale richiedano la risoluzione, da parte degli sviluppatori, di problemi sia software che hardware a volte molto complessi.

Per quanto riguarda *Astra Data Navigator* si elencano di seguito una panoramica dei principali requisiti per la realizzazione del progetto:

- ***Ricostruzione realistica***: l'utente deve avere la certezza che ciò che osserva all'interno dell'universo virtuale rispecchi la realtà. Non si parla solo di realismo grafico degli oggetti dei quali è conosciuta la conformazione del terreno, quali Terra, Marte o Luna, ma anche di realismo delle loro posizioni spaziali ed evoluzione nel tempo.

- ***Caricamento di dati scientifici***: l'accuratezza scientifica che l'applicazione deve garantire viene supportata dalla possibilità di caricamento di dati quali cataloghi stellari e *SPICE* kernel.

- ***Configurabilità da parte dell'utente***: essendo *ADN* realizzata per diversi scopi, gli utenti devono avere la possibilità di configurarlo al fine di soddisfare le proprie esigenze, alcune soluzioni richiedono dei datasets esterni che non è possibile inserire direttamente all'interno dell'applicazione a causa delle grandi moli di dati.

- ***Navigazione fluida ed intuitiva***: occorre dare la possibilità agli utenti di poter navigare all'interno dell'universo virtuale al fine di esplorare ogni angolo da loro desiderato. In quanto i dati rappresentabili permettono di generare un mondo estremamente vasto, è necessario che il movimento all'interno di esso risulti fluido. L'utilizzatore dovrà essere agevolato nella navigazione sfruttando la GUI tramite informazioni riguardanti la sua posizione attuale e cosa si trova attorno a lui.

- ***Supporto per stereoscopia 3D***: l'applicazione è progettata al fine di supportare la visualizzazione stereoscopica 3D in modo da rendere una maggiore immersività, nel caso particolare è stata sperimentata utilizzando un CAVE[2]

[2]ambiente per la realtà virtuale immersiva

a singolo schermo ed un proiettore a stereoscopia attiva presente all'interno del laboratorio VR dell'ALTEC.

Per ovviare ad alcune delle problematiche riscontrate sono stati sfruttati alcuni software o API[3], sia proprietari che open source, resi disponibili dalla comunità scientifica. Ciò ha permesso di concentrarsi maggiormente su requisiti unici per *Astra Data Navigator*, investendo meno tempo su quelli condivisi con altre applicazioni simili. Verranno, nei capitoli successivi, presentate alcune delle tecnologie utilizzate.

La tesi si compone di cinque capitoli, di seguito ne viene fornita una breve descrizione.

Nel primo capitolo vengono analizzati le attuali soluzioni per la visualizzazione dei movimenti dei corpi celesti nel tempo, evidenziando quali supportano l'utilizzo degli *SPICE Kernels*.

Nel secondo capitolo vengono descritte le tecnologie e i tools utilizzati nello sviluppo della tesi, con particolare attenzione e approfondimento sugli *SPICE Kernels*.

Nel terzo capitolo viene delineato il design e lo sviluppo dell'applicazione, analizzando una ad una le nuove features realizzate e quali tecniche sono state utilizzate al fine di permettere la visualizzazione realtime senza perdita di prestazioni per l'utente.

Il quarto capitolo esamina le problematiche e le criticità riscontrate durante lo sviluppo dell'applicazione, fornendo, inoltre, un'analisi statistica di test soggettivi effettuati sulle funzionalità, le prestazioni e l'utilità dell'applicazione.

Il quinto capitolo affronta le conclusioni soffermandosi sui risultati ottenuti, fornendo, inoltre, nuovi possibili aggiornamenti che, in futuro, potrebbero accrescere le caratteristiche di *Astra Data Navigator*.

[3]insieme di procedure atte all'espletamento di un dato compito

Capitolo 1

Stato dell'arte

In ambito aerospaziale conoscere l'esatta posizione ed il comportamento temporale dei corpi celesti è una delle chiavi per la buona riuscita di una missione spaziale. Queste informazioni possono essere calcolate attraverso l'utilizzo degli *SPICE Kernels* (approfonditi nel paragrafo 2.2), che permettono di ricavare, in un sistema di riferimento dato, le coordinate vettoriali dei corpi celesti quali pianeti, satelliti e rover. Il *NAIF (Navigation and Ancillary Information Facility)* della *NASA* offre ai progetti di volo e ai ricercatori il sistema di informazione sulla geometria di osservazione *"SPICE"* per assistere gli scienziati nella pianificazione e interpretazione delle osservazioni scientifiche da strumenti spaziali a bordo di veicoli spaziali. *SPICE* è anche utilizzato a supporto delle attività di ingegneria associate a queste missioni. Mentre le missioni planetarie erano l'obiettivo originale, oggi *SPICE* viene utilizzato anche in alcune missioni di eliofisica[1] e scienze della terra [9].

All'interno di una simulazione in tempo reale, l'utilizzo degli *SPICE* comporta alcune limitazioni, essendo progettati per effettuare calcoli puntuali precisi, occorre effettuare un trade-off tra precisione e prestazioni, interpolando eventualmente i risultati ottenuti in momenti separati del tempo. I calcoli effettuati dalle diverse librerie degli *SPICE* richiederebbero troppo tempo per essere svolti ad ogni frame della simulazione, così facendo i Frames per Second (FPS) risulterebbero troppo pochi causando fastidio nell'osservatore.

Nonostante la difficoltà di rappresentare in tempo reale le informazioni geometriche dei corpi celesti sono state sviluppate diverse applicazioni che permettono questa caratteristica.

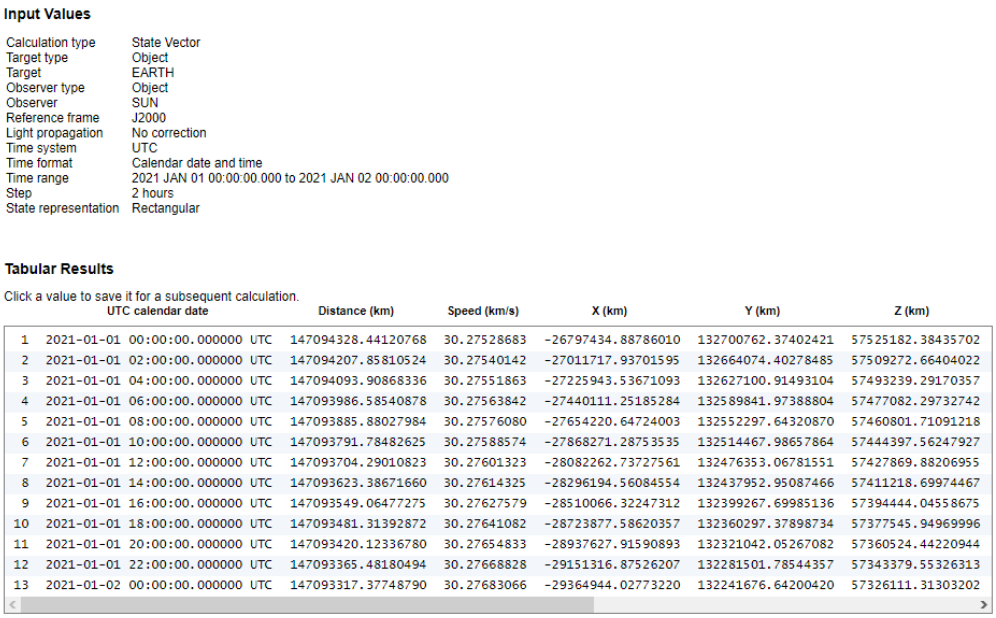
1.1 Tool che supportano gli SPICE Kernels

Esistono diversi tool che sfruttano le funzionalità degli *SPICE*, sia in ambiente Web che attraverso applicazioni VR.

[1] la scienza delle connessioni fisiche tra il Sole e il sistema solare

1.1.1 WebGeocalc

WebGeocalc è un prodotto di *NAIF* sviluppato presso il *Jet Propulsion Laboratory*, *California Institute of Technology*, facente parte della *Planetary Science Division* del *Science Mission Directorate* della *NASA*. È uno strumento che fornisce un'interfaccia utente Web esponendo delle funzionalità di calcolo disponibili dal sistema *SPICE*. L'uso di questa piattaforma rende estremamente semplice l'utilizzo dei Kernel senza necessità di scrivere programmi ad hoc. Il server *WGC* è creato partendo dal codice del Toolkit di *NAIF*.



Input Values

Calculation type	State Vector
Target type	Object
Target	EARTH
Observer type	Object
Observer	SUN
Reference frame	J2000
Light propagation	No correction
Time system	UTC
Time format	Calendar date and time
Time range	2021 JAN 01 00:00:00.000 to 2021 JAN 02 00:00:00.000
Step	2 hours
State representation	Rectangular

Tabular Results

Click a value to save it for a subsequent calculation.

	UTC calendar date	Distance (km)	Speed (km/s)	X (km)	Y (km)	Z (km)
1	2021-01-01 00:00:00.000000 UTC	147094328.44120768	30.27528683	-26797434.88786010	132700762.37402421	57525182.38435702
2	2021-01-01 02:00:00.000000 UTC	147094207.85810524	30.27540142	-27011717.93701595	132664074.40278485	57509272.66404022
3	2021-01-01 04:00:00.000000 UTC	147094093.90868336	30.27551863	-27225943.53671093	132627100.91493104	57493239.29170357
4	2021-01-01 06:00:00.000000 UTC	147093986.58540878	30.27563842	-27440111.25185284	132589841.97388804	57477082.29732742
5	2021-01-01 08:00:00.000000 UTC	147093885.88027984	30.27576080	-27654220.64724003	132552297.64320870	57460801.71091218
6	2021-01-01 10:00:00.000000 UTC	147093791.78482625	30.27588574	-27868271.28753535	132514467.98657864	57444397.56247927
7	2021-01-01 12:00:00.000000 UTC	147093704.29010823	30.27601323	-28082262.73727561	132476353.06781551	57427869.88206955
8	2021-01-01 14:00:00.000000 UTC	147093623.38671660	30.27614325	-28296194.56084554	132437952.95087466	57411218.69974467
9	2021-01-01 16:00:00.000000 UTC	147093549.06477275	30.27627579	-28510066.32247312	132399267.69985136	57394444.04558675
10	2021-01-01 18:00:00.000000 UTC	147093481.31392872	30.27641082	-28723877.58620357	132360297.37898734	57377545.94969996
11	2021-01-01 20:00:00.000000 UTC	147093420.12336780	30.27654833	-28937627.91590893	132321042.05267082	57360524.44220944
12	2021-01-01 22:00:00.000000 UTC	147093365.48180494	30.27668828	-29151316.87526207	132281501.78544357	57343379.55326313
13	2021-01-02 00:00:00.000000 UTC	147093317.37748790	30.27683066	-29364944.02773220	132241676.64200420	57326111.31303202

Figura 1.1: *WebGeocalc*: esempio di calcolo dei vettori di stato della Terra

WebGeocalc supporta la comunità scientifica e la ricerca in diversi modi:

- mettendo a disposizione molte delle capacità di calcolo della libreria *SPICE*;
- offrendo un meccanismo che scienziati ed ingegneri possono utilizzare per verificare rapidamente la correttezza dei dati estrapolati da software basati sempre sul codice degli *SPICE*;
- fornendo un mezzo semplice e veloce per revisionare a campione gli archivi di dati scientifici;
- permettendo risposte rapide sulla geometria dei corpi celesti in caso di necessità immediata.

Per l'utilizzo di questo tool, viene richiesta una certa conoscenza sulle geometrie spaziali e sui diversi sistemi di riferimento, in quanto le possibilità che presenta all'utente sono molto vaste ed una scorretta comprensione dei risultati forniti porterebbe a sostanziali incongruenze nei calcoli futuri.

Esistono diverse istanze di *WebGeocalc*, una appartenente a *NASA* [10] ed una appartenete ad *ESA* [11] le funzionalità offerte sono le stesse, la differenza tra i due servizi è data dai diversi kernel caricabili per i calcoli. Ognuno dei servizi offerti dalle due agenzie spaziali permette l'utilizzo dei kernel basati sulle proprie missioni in aggiunta ad alcuni generici.

1.1.2 Cosmographia

NAIF offre, per uso pubblico, un'estensione dell'applicazione open source[2] *Cosmographia* sviluppata da Chris Laurel nella quale è possibile l'utilizzo degli *SPICE* [12]. Lo strumento è in grado di produrre una visualizzazione tridimensionale delle effemeridi, dimensioni e forme dei pianeti e delle traiettorie ed orientamenti dei veicoli spaziali. Per il rendering, *Cosmographia* si serve della libreria 3D *VESTA* per i modelli strutturali e i dati volumetrici.
L'applicazione è stand-alone eseguibile su macOS, Windows e Linux.

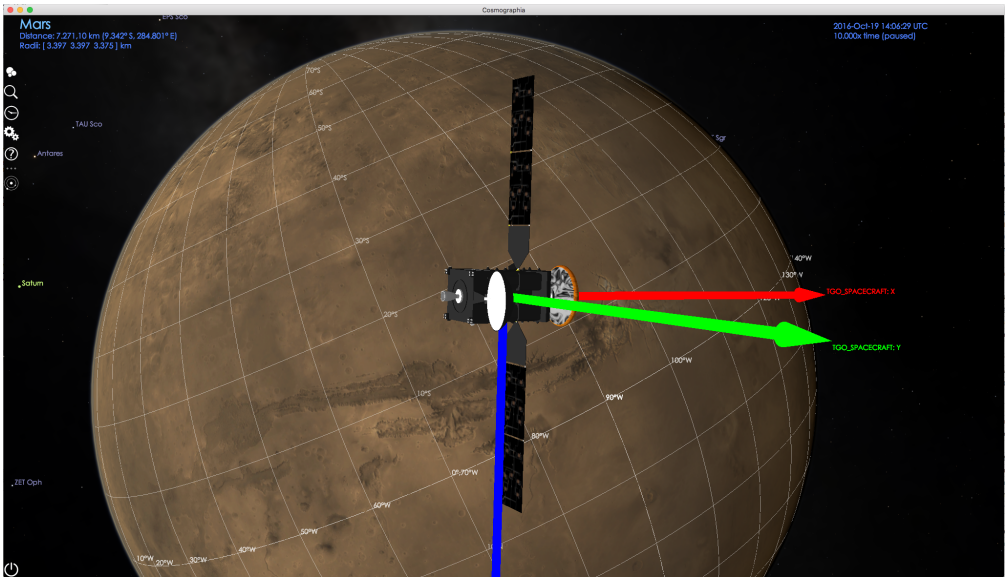


Figura 1.2: *Cosmographia*: visualizzazione di TGO della missione ExoMars

I controlli utente consentono di gestire il punto di osservazione, la velocità delle animazioni ed aggiungere annotazioni per indicare quando uno strumento sta acquisendo dati.

Per utilizzare gli *SPICE* all'interno di *Cosmographia* occorre configurare dei file JSON, i quali definiscono quali kernel utilizzare per la simulazione. All'interno dei file di installazione sono presenti gli *Spacecraft and Planet Kernel (SPK)* della maggior parte dei pianeti e dei satelliti per un periodo che va dal 1950 al 2050, ciò

[2]software non protetto da copyright e liberamente modificabile dagli utenti

consente comunque all'utente di estendere i cataloghi con file aggiuntivi presenti in locale.

Cosmographia nasce come simulatore del nostro sistema solare, attualmente *NAIF* continua ad estenderlo per incrementarne l'accuratezza per l'osservazione delle missioni planetarie per le quali sono disponibili diversi dataset *SPICE*. In futuro, quest'applicazione, diventerà uno strumento di supporto grafico per scienziati ed ingegneri. Esiste una versione di *Cosmographia* che supporta i dati delle missioni di *ESA*.

1.1.3 CosmoScoutVR

German Aerospace Center (DLR), nel 2019, rilascia, sotto licenza del *Massachusetts Institute of Technology (MIT)* la prima versione di *CosmoScout VR* [13] un software open source eseguibile sia per piattaforme Linux che Windows.

CosmoScout VR si presenta come un universo virtuale e modulare che consente di esplorare, analizzare e mostrare enormi dataset planetari in tempo reale. Con modulare si intende la possibilità di aggiungere diversi plugin all'interno della simulazione in modo da incrementare il numero di features disponibili, si elencano alcuni di questi:

- *csp-atmosphere*: permette di rappresentare le atmosfere dei pianeti, calcolando lo scattering di Mie e Rayleigh tramite raycasting in tempo reale;
- *csp-fly-to-location*: aggiunge scorciatoie per i corpi celesti e per posizioni geografiche;
- *csp-lod-bodies*: renderizza pianeti e lune utilizzando dati trasmessi tramite *Web-Map-Services (WMS)*;
- *csp-measurement-tools*: fornisce strumenti per la misurazione del terreno come distanze, profili di altezza, volumi o aree;
- *csp-trajectories*: disegna le traiettorie dei corpi celesti e delle spacecraft basate su *SPICE*;
- *csp-stars*: permette il caricamento di cataloghi stellari quali *Tycho*, *Tycho2* e *Hipparcos*.

1.2 Tool VR per la visualizzazione dell'universo

Sin dall'antichità l'uomo ha osservato il cielo, catalogando centimetro dopo centimetro tutti i corpi celesti visibili dalla Terra. Da Socrate a Hawking passando per Galilei moltissimi filosofi, scienziati e astrofili si sono affascinati e hanno passato la vita guardando ciò che non avrebbero mai potuto raggiungere.
Oggi lo spazio non è ancora completamente accessibile, nonostante ciò lo sviluppo

tecnologico ci permette di viaggiare ugualmente attraverso l'universo conosciuto senza uscire dall'atmosfera terrestre. Grazie all'abbondanza di dati sono nati moltissimi progetti e applicazioni per rendere partecipe il pubblico alle grandi meraviglie dell'universo.
Con la Realtà Virtuale ogni luogo è potenzialmente a portata di mano.

Esistono numerosi tool il cui interesse non è necessariamente legato alla rappresentazione accurata di dati planetari quali posizione e orientamento ma si concentrano maggiormente sulla visualizzazione grafica dell'universo.

1.2.1 SpaceEngine

Scritto in C++ e creato da Vladimir Romanyuk, sviluppatore e astronomo russo, *SpaceEngine* [14] è un universo pseudo-realistico esplorabile dal proprio computer. Permette di viaggiare tra le stelle, le galassie, atterrare sui pianeti o sui satelliti e altro ancora. Offre la possibilità di alterare la velocità della simulazione al fine di poter osservare i fenomeni astronomici. La grandezza dell'universo esplorabile è pressoché infinita. La generazione del mondo è procedurale e basata, per quanto possibile, sulla conoscenza scientifica che abbiamo dell'universo.

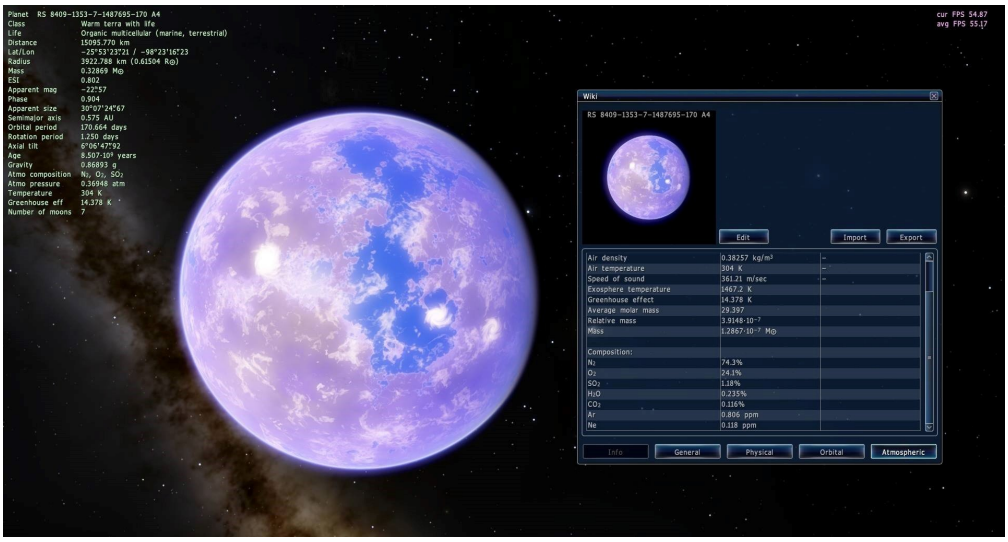


Figura 1.3: *SpaceEngine*: esempio di generazione procedurale di un pianeta

SpaceEngine è disponibile per Windows e supporta sia una modalità di uso normale che VR, la quale è sfruttabile con visori come *Valve Index*, *HTC Vive* e *Oculus Rift*.
Per quanto riguarda il sistema di input è possibile utilizzare mouse e tastiera, gamepad ma anche sistemi di tracking dei movimenti rendendo l'applicazione un perfetto esempio di interazione in Realtà Virtuale tra umano e computer.

La modalità Spacecraft è una feature nella quale l'utente ha la possibilità di muoversi liberamente nell'universo stando all'interno di una navicella spaziale

soggetta alle forze di inerzia. Altra caratteristica rilevante è la simulazione dinamica di aurore boreali, nuvole delle giganti gassose, effetto lente gravitazionale per buchi neri e stelle di neutroni, fenomeni di curvatura dello spazio nell'ipotetico caso in cui un astronauta a bordo di una navicella spaziale si avvicinasse all'orizzonte degli eventi di un buco nero.

Curiosità interessante: nel caso in cui un pianeta presenti caratteristiche simili a quelle della Terra viene etichettato con la possibilità di presenza di vita su di esso.

1.2.2 Celestia

Celestia [15] è un simulatore spaziale multipiattaforma eseguibile per Linux, Windows, macOS, Android e iOS che permette una vista tridimensionale del nostro universo.
Scritto in C++ e Lua con licenza GNU GPL[3], sfrutta la libreria grafica *OpenGL*, è sviluppato da Chris Laurel, programmatore di Seattle, che rilascia la prima versione nel 2001.

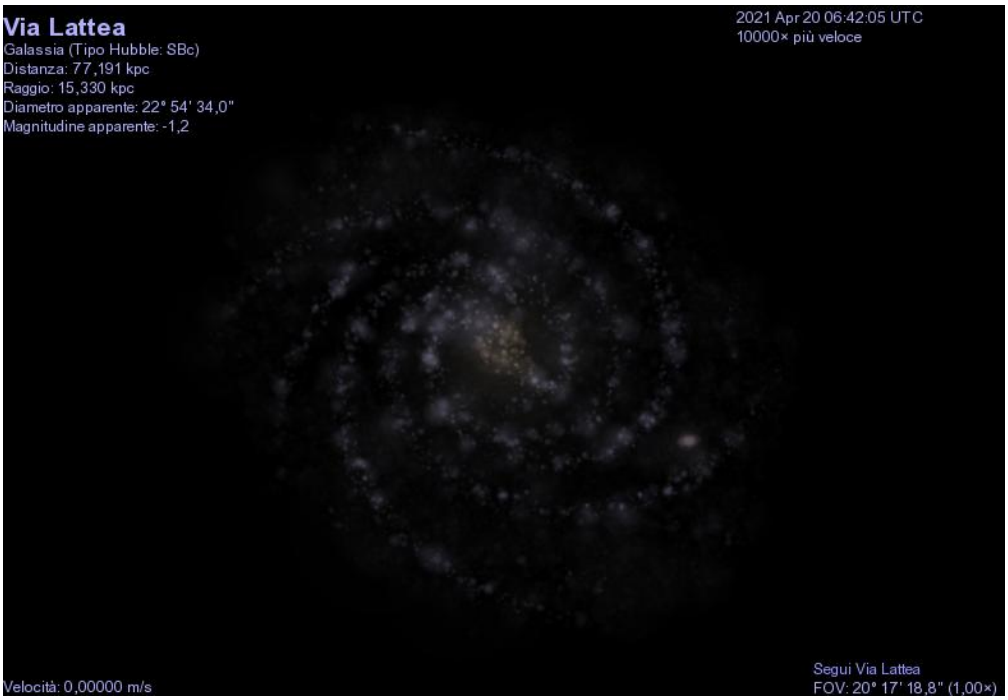


Figura 1.4: *Celestia*: rappresentazione della Via Lattea

Come planetario digitale *Celestia* non si limita a mostrare la volta celeste terrestre ma rende possibile viaggiare nel sistema solare, verso oltre 100.000 stelle e fuori dalla galassia. Viene fornito con all'interno cataloghi contenenti stelle,

[3]GNU GPL: licenza copyleft per software libero

galassie, pianeti, lune, spacecraft e satelliti artificiali ed è eventualmente espandibile scaricando decine di componenti aggiuntivi.
Tutti i movimenti sono continui e la funzione di zoom esponenziale permette di esplorare lo spazio su una vasta gamma di scale.

Nel caso in cui l'utente non dovesse trovare il corpo celeste che stava cercando, il tool offre la possibilità di creare interi sistemi planetari, nebulose, galassie o oggetti di fantasia.

Per incrementare l'interesse educativo all'interno di *Celestia* si possono creare degli script contenenti una sequenza di istruzioni da dare al programma per eseguire determinate procedure che filmate possono essere utilizzate a scopo divulgativo.

Applicazione	**Supporto *SPICE* kernel**	**Navigazione spaziale**	**Generazione procedurale dell'universo**	**Supporto VR**
Cosmographia	Completo	Punta e clicca, catalogo visivo oggetti, ricerca testuale	No, applicazione accurata scientificamente	Sistema stereoscopico con anaglifi o doppia immagine
CosmoScout VR	Si, installando il relativo plugin	Movimento libero, mappa del Sistema Solare	No	VR headset, sistema stereoscopico, tracking hardware, CAVE
SpaceEngine	No	Movimento libero simulando le forze di inerzia a cui si è sottoposti, punta e clicca, zoom esponenziale	Si, creando un universo potenzialmente infinito	SteamVR, Windows Mixed Reality
Celestia	No	Movimento libero, punta e clicca, zoom esponenziale, creazione di script con sequenze di istruzioni	Si, possibilità di creare interi sistemi planetari, nebulose, galassie e oggetti di fantasia	CAVE

Tabella 1.1: Confronto tra le applicazioni VR analizzate

1.3 Astra Data Navigator

Astra Data Navigator (ADN) è un'applicazione stereoscopica 3D sviluppata da *ALTEC* nella quale vengono rappresentati molti dei corpi celesti del sistema solare, posizionandoli nello spazio ricavando le coordinate X, Y, Z attraverso gli *SPICE kernels*. Allo stato di partenza la simulazione era in grado di mostrare il catalogo stellare *Hipparcos*[4] costituito da circa 110 mila elementi, è tuttavia in corso un lavoro di sviluppo per incrementare la capacità di visualizzazione di cataloghi stellari, in accordo con le limitazioni tecnologiche, fino a raggiungere quello di *Gaia*[5] composta da oltre 1,8 miliardi di stelle.

I pianeti ed i satelliti sono oggetti tridimensionali ai quali vengono applicate delle texture al fine di riprodurre fedelmente la superficie visibile, per il rendering delle stelle viene utilizzato un sistema di particelle basato sulla distanza che esse hanno dalla camera, nel caso in cui l'utente si trovi in prossimità di una stella questa viene visualizzata come un oggetto 3D a cui è stato assegnato uno shader parametrizzato utilizzando i valori presenti all'interno del catalogo.

ADN offre due diverse modalità di movimento all'interno della scena:

- ***movimento libero***: utilizzabile con l'input da tastiera tramite i comandi W/A/S/D oppure le frecce, in questa modalità l'utente è libero di esplorare potendo regolare la velocità di movimento della camera, utilizzando la rotella del mouse, raggiungendo valori astronomici;

- ***warp***: attraverso un pannello di ricerca è possibile digitare il nome di un corpo celeste o l'identificativo di una stella presenti all'interno della simulazione per poi viaggiare automaticamente fino a raggiungerlo per poterlo osservare da vicino.

1.3.1 SPICE in Astra Data Navigator

L'obiettivo della nuova versione di *ADN* è quello di introdurre il concetto di tempo all'interno della simulazione. Inizialmente gli oggetti visualizzati erano tutti statici e privi di movimento nonostante lo scorrere del tempo. Per raggiungere lo scopo prefissato non era sufficiente conoscere la posizione iniziale ricavata attraverso gli *SPICE kernels* ma occorreva realizzare delle interfacce che permettessero ad *ADN* la possibilità di conoscere le coordinate spaziali degli oggetti in ogni istante di tempo. Ciò permette all'utente di potersi spostare non solo attraverso lo spazio ma anche attraverso il tempo avendo la sicurezza che la rappresentazione spaziale risulti sempre accurata.

[4]High Precision Parallax Collecting Satellite, prima missione spaziale di *ESA* dedicata all'astrometria, 1989 - 1993

[5]Global Astrometric Interferometer for Astrophysics, continuazione della missione astrometrica di Hipparcos, 2013 - in corso

Nella versione più recente di *ADN* l'utente ha la possibilità di configurare la simulazione per quanto riguarda il metodo di rivoluzione scegliendo tra tre possibili opzioni:

- ***SpiceKernels***: utilizza una dll compilata in C++ che espone le principali funzionalità dello *SPICE Toolkit* di *NAIF* (capitolo 2.2), questa soluzione necessita di una versione in locale dei kernel che si vogliono caricare, il che comporta un grande dispendio di spazio e conoscenza minima del mezzo che si va ad utilizzare;
- ***PMS***: servizio web facente parte del progetto *NEANIAS* (capitolo 2.3) che permette ai suoi utenti di poter richiedere facilmente le coordinate spaziali di quasi tutti i corpi celesti del sistema solare. Attraverso questa soluzione *ADN* sfrutta le API esposte dal servizio per ricavare i dati scientifici, soluzione che non richiede nessun dispendio di memoria e nemmeno conoscenze dei *kernel*;
- ***Kepler***: configurazione standalone di *ADN* basata sulle leggi di Keplero (capitolo 2.4), non assicura la precisione scientifica delle posizioni mostrate ma non richiede alcuna conoscenza della materia e sopratutto permette elevatissime prestazioni e caratteristiche grafiche aggiuntive che le altre soluzioni non hanno possibilità di offrire.

Le funzionalità di movimento all'interno della simulazione rimangono pressoché le stesse, migliorando le performance grafiche ed il framerate. Viene aggiunto un pannello informativo per ogni corpo celeste all'interno dell'universo in grado di mostrare all'utente le principali informazioni conosciute riguardo all'oggetto selezionato, sono inoltre presenti circa un centinaio di nomi propri di stelle e vengono fornite informazioni riguardanti le costellazioni a cui esse appartengono.

Astra Data Navigator è progettato per funzionare su sistemi operativi Windows e macOS, oltre che per piattaforme web come WebGL, supporta perfettamente proiettori a stereoscopia attiva, mentre è in via di sviluppo una versione utilizzabile tramite headset VR.

Capitolo 2

Tecnologie

Nei paragrafi successivi verranno presentate alcune fondamentali tecnologie utilizzate per la realizzazione della tesi.

2.1 Unity

Il software principale con cui è stato sviluppato *Astra Data Navigator* è Unity [16], un motore grafico multipiattaforma scritto in C++, creato da Unity Technologies, che consente lo sviluppo di applicazioni interattive e videogiochi in tempo reale sia 2D che 3D. Disponibile per Windows, macOS e Linux, permette di creare contenuti per quasi ogni piattaforma in commercio. Il linguaggio di programmazione utilizzato in Unity è il C#. Oltre alla presenza di numerosi tools e librerie all'interno del programma, Unity dispone di una gigantesca community che sviluppa contenuti riutilizzabili caricati poi sull'Asset Store, sia gratis che a pagamento, all'interno dei propri progetti. Alcuni di questi sono stati utilizzati in *ADN* per risolvere facilmente uno dei problemi maggiori che si riscontrano nello sviluppo di applicazioni in ambito spaziale, ovvero il problema della virgola mobile.

La scelta del motore grafico è ricaduta su Unity poiché risulta essere uno dei framework più semplici ed intuitivi, in grado di generare versioni delle applicazioni prodotte per diverse piattaforme. La manutenibilità risulta estremamente più semplice rispetto ad un software scritto interamente da zero utilizzando le primitive API grafiche come *OpenGL*. Come ogni compromesso, occorre tenere in considerazione il fattore limitante di questo genere di framework, ovvero quello di perdere il pieno controllo di alcuni elementi a runtime e dover sottostare a regole imposte da altri sviluppatori che potrebbero causare, in alcuni casi, cali di prestazioni e di conseguenza di FPS. È comunque possibile, adottando diverse tecniche, ottenere risultati estremamente positivi, tenendo conto del tempo risparmiato in fase di sviluppo non essendo necessario preoccuparsi delle complesse primitive grafiche riservate a programmatori esperti a cui occorre controllare interamente ogni processo eseguito dalle applicazioni.

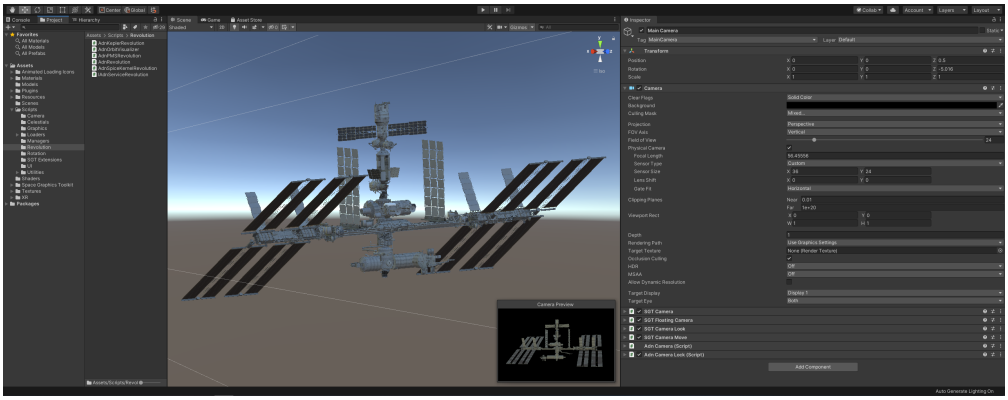


Figura 2.1: Visualizzazione dell'ISS all'interno dell'editor di *Unity*

È utile considerare anche l'immediato supporto per la stereoscopia 3D, uno dei requisiti del progetto citati precedentemente, che Unity mette a disposizione dei propri utenti installando un componente aggiuntivo il quale permette la configurazione dei parametri principali come ad esempio la disparità binoculare, controllata aggiustando la distanza tra le due camere virtuali.

2.1.1 Floating Origin

Quando si sviluppa un'applicazione per la rappresentazione dell'universo all'interno di un mondo virtuale, oppure più in generale quando si ha a che fare con ambienti virtuali di vastissime dimensioni, si incorre nel problema della virgola mobile. In Unity, per rappresentare la posizione di un oggetto all'interno della scena viene utilizzato un Vector3 composto dai tre valori X / Y / Z in float, ovvero con 32bit di memoria, ciò genera problemi quando si cercano di salvare dati estremamente grandi che, quindi, vengono approssimati per farli rientrare all'interno dei 32bit. Il degrado dell'accuratezza risulta quindi direttamente proporzionale alla distanza che l'oggetto ha rispetto all'origine.

Graficamente viene chiamato jitter, ovvero gli oggetti troppo lontani verranno renderizzati in posizioni leggermente differenti ad ogni frame, causando così uno sfarfallio che combinato alla difficoltà dello Z-buffer a calcolare quale sia la prossimità corretta di un corpo rispetto ad un altro, provocheranno un effetto di movimento quando in realtà l'oggetto in questione dovrebbe risultare perfettamente fermo.

In soccorso a questo problema viene creata la tecnica del Floating Origin che consiste nello spostare l'origine del mondo in base a dove si trova la camera in quel momento, all'atto pratico non è possibile spostare l'origine di un sistema cartesiano dallo zero ma è possibile traslare inversamente tutti i corpi rispetto alla camera. Per essere più chiari, se la camera si dovesse spostare di cento unità positive sull'asse X tutti gli oggetti verrebbero traslati di cento unità negative sullo stesso asse mantenendo la camera centrata nell'origine (figura 2.2), l'utente non si accorgerebbe di alcuna differenza ma così facendo viene sfruttata la regione di

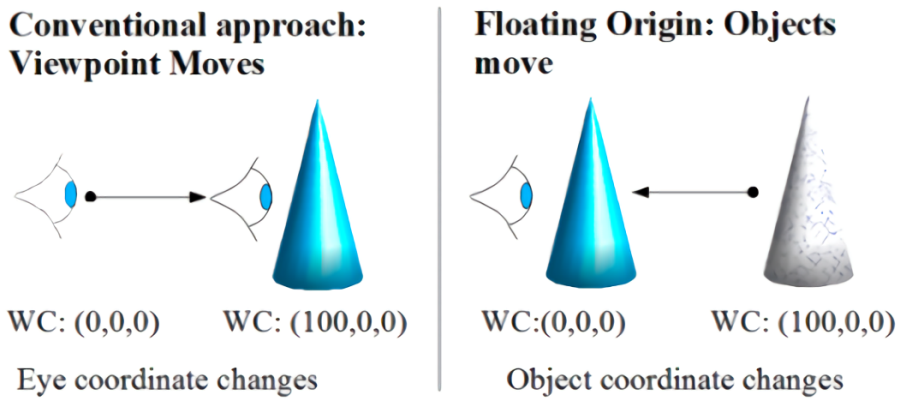


Figura 2.2: Rappresentazione della tecnica del Floating Origin

massima precisione per la rappresentazione delle coordinate, quella prossima allo zero.

2.1.2 Space Graphics Toolkit

Per risolvere completamente la criticità del jitter spaziale non è sufficiente la sola tecnica del Floating Origin, occorre infatti combinare anche un metodo efficacie per evitare che le coordinate di un oggetto diventino troppo grandi da perdere precisione. Tale tecnica consiste nella suddivisione del mondo virtuale in celle di grandezza fissa minore del massimo numero rappresentabile attraverso un float prima di perdere precisione. Così facendo si otterrà un sistema di coordinate basato sulla combinazione delle coordinate globali, le quali indicizzano a quale cella ci si riferisce, con le coordinate locali all'interno della cella stessa.

L'unione di queste due tecniche permette di eliminare completamente il fenomeno del jitter spaziale evitando che spiacevoli artefatti vengano generati in fase di rendering.

Esiste un toolkit, disponibile nell'Asset Store di Unity [17], chiamato *Space Graphics Toolkit (SGT)*, sviluppato da Carlos Wilkes, in grado di gestire le tecniche citate precedentemente. Una volta acquistato e scaricato mette a disposizione dello sviluppatore innumerevoli scripts, modelli di corpi celesti, shader e texture completamente customizzabili.

Ad ogni *GameObject* di Unity viene attaccato un `SgtFloatingObject` in grado di farlo fluttuare nel mondo utilizzando il metodo sopracitato (paragrafo 2.1.1). Per questioni di ottimizzazione l'aggiornamento delle posizioni viene eseguito quando la camera si sposta di circa cento blocchi rispetto all'origine e non ad ogni frame. Il componente designato per la gestione del sistema a doppie coordinate è chiamato `SgtPosition`, elemento fondamentale per lo sviluppo del progetto di tesi, che utilizza sei interi a 64bit per gestire le tre coordinate globali e le tre locali; lo

sviluppatore di *SGT* dichiara che attraverso l'utilizzo di questo asset sia possibile rappresentare l'universo senza rischiare di incorrere in problemi di precisione.

2.2 SPICE Kernels

Creati dal *Navigation and Ancillary Information Facility (NAIF)* sotto la direzione del *Planetary Science Division* della *NASA*, sono uno strumento essenziale per assistere gli scienziati di tutto il mondo per la modellazione e pianificazione di missioni di esplorazione e per l'interpretazione dei dati derivati dalle osservazioni planetarie. *SPICE* è l'acronimo di Spacecraft Planet Instrument C-matrix (camera matrix) Events.

L'uso degli *SPICE* si estende dallo sviluppo del concetto di missione fino alla fase di analisi dei dati post-missione, includendo il supporto per correlare le informazioni derivate dagli strumenti presenti su di uno o più veicoli spaziali. Questa potente risorsa permette di calcolare numerosi parametri della geometria di osservazione planetaria in determinati istanti di tempo.

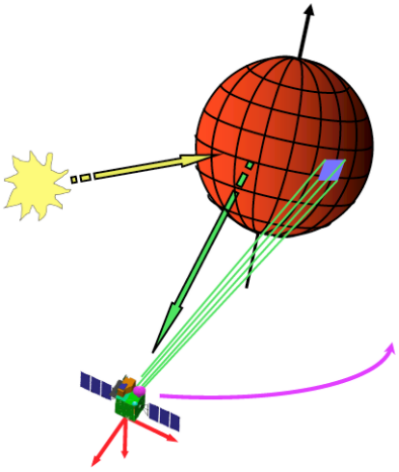


Figura 2.3: Esempio delle potenzialità degli *SPICE*

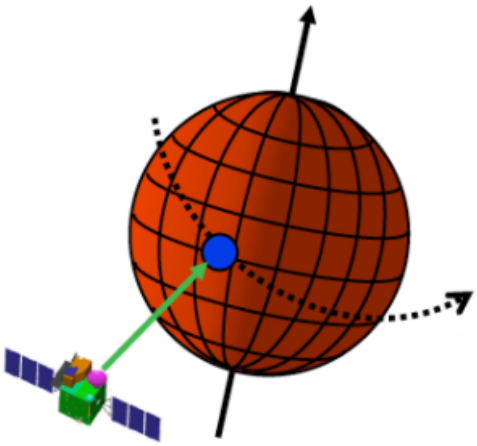


Figura 2.4: Evento di un corpo che transita davanti ad un satellite

Le innumerevoli funzionalità degli *SPICE* variano dal calcolo del vettore posizione e velocità della maggior parte dei pianeti, satelliti, comete, asteroidi e spacecraft; ottenere informazioni riguardo a grandezza, forma e orientamento degli stessi; visualizzare movimenti e orientamento dei veicoli spaziali; ricavare la posizione del Field of View (FOV) di uno strumento sulla superficie di un corpo (Figura 2.3).

Vi è la possibilità inoltre, di calcolare il momento esatto dell'occorrenza di un determinato evento, un astro si trova in opposizione (ombra) rispetto ad un altro, un veicolo spaziale si trova in un determinato range di distanza da un corpo celeste,

uno strumento punta verso un oggetto, un corpo transita davanti ad una spacecraft (Figura 2.4).

2.2.1 I Kernel

I set di dati primari degli *SPICE* sono genericamente chiamati kernel, i principali compongono la parola *SPICE*, all'interno dei quali sono contenute differenti informazioni riguardanti i corpi celesti, nello specifico si differenziano in moduli:

- ***SPK***: spacecraft and planet kernel, all'interno di questi moduli è possibile trovare le effemeridi[1] dei corpi celesti naturali o costruiti dall'uomo. Più genericamente, sono contenuti i dati di posizione di un oggetto in funzione del tempo.
  Per quanto riguarda il lavoro di tesi, questo modulo risulta essere uno dei più importanti in quanto contiene la maggior parte delle informazioni necessarie alla funzionalità degli *SPICE* usate in *ADN*.

- ***PCK***: physical, dynamical and cartographic kernel, racchiudono le costanti fisiche e cartografiche di un oggetto, come ad esempio forma, dimensione e orientamento degli assi di rotazione.
  Altro componente fondamentale usato da *ADN*.

- ***IK***: instrument kernel, contengono le informazione riguardanti le peculiarità geometriche di un dato strumento, come dimensione del Field of View, forma e parametri per l'orientamento.

- ***CK***: C-matrix kernel, permettono di calcolare le matrici di trasformazione tra due frame di riferimento partendo dai dati di orientamento di diversi veicoli spaziali, opzionalmente contengono informazioni riguardanti le trasformazioni di velocità angolari.
  Kernel utilizzato per la gestione della rotazione all'interno di *ADN*.

- ***EK***: events information kernel, all'interno dei quali sono contenute le informazioni sulle attività delle missioni spaziali.

Esistono ulteriori moduli altrettanto importanti che non formano l'acronimo *SPICE*, ma risultano essenziali al fine del corretto funzionamento del sistema:

- ***FK***: frames kernel, contenenti le specifiche dei frame di riferimento tipicamente usati nei progetti di volo. Il frame utilizzato all'interno di *ADN* è il J2000, definito sulla base del piano equatoriale della Terra centrato nel Sole il primo gennaio 2000 alle ore 12:00:00.000 Barycentric Dynamical Time (TDB)[2].

[1] tabelle che contengono valori calcolati, nel corso di un particolare intervallo di tempo, di diverse grandezze astronomiche variabili

[2] sistema relativistico di coordinate temporali

- ***SCLK e LSK***: spacecraft clock kernel e leap second kernel, vengono utilizzati per convertire istanti di tempo tra diversi sistemi di misurazione.
- ***DSK***: digital shape model kernel, racchiudono le informazioni riguardanti alcuni corpi celesti dalla forma irregolare come asteroidi e comete.

La lettura diretta dei kernel risulta estremamente complessa, anche per esperti come ingegneri aerospaziali e fisici, per questo motivo sono stati creati diversi strumenti che permettono un accesso più diretto alle informazioni, come ad esempio WebGeocalc (Paragrafo 1.1.1). Lo *SPICE Toolkit* di *NAIF* è disponibile in diversi linguaggi come C, IDL e MATLAB, sono stati fatti, da terze parti, alcuni wrapper in Python (SpiceyPy utilizzato per generare i cataloghi per *ADN*), Ruby, Swift e Julia.

Nel corso del tempo sono stati creati innumerevoli kernel per ogni missione o corpo celeste, a volte i dati di diversi oggetti vengono racchiusi all'interno di un solo modulo. Al fine di ottenere la maggior precisione possibile, viene richiesta di una vastissima quantità di informazioni. È possibile caricare dati ridondanti riguardo un corpo, tuttavia, il sistema riconosce ed utilizza unicamente quelli più aggiornati. Per venire incontro alla difficoltà di caricamento di così tanti moduli è stata data la possibilità di utilizzare un *metakernel*: ovvero, un file con una particolare dicitura, nella quale vengono forniti i path di ogni singolo modulo da utilizzare.
Scaricando dai siti di *NASA* ed *ESA* i kernel delle varie missioni spaziali saranno già presenti all'interno della cartella alcuni *metakernel* precompilati in modo da caricare correttamente tutti i moduli. È comunque sempre possibile modificarli al fine di integrare nuovi kernel per costruire il proprio personale dataset.

Le principali funzioni utilizzate per interfacciarsi ai kernel messe a disposizione dal toolkit di *NAIF* saranno:

- ***furnsh***: utilizzata per caricare un dato *metakernel*.
- ***spkpos***: come si deduce dal nome, necessaria per calcolare il vettore posizione di un corpo rispetto ad un altro dato un frame di riferimento in un determinato istante di tempo.
- ***str2et***: permette di convertire una stringa rappresentante una data in Ephemeris Time, ovvero secondi passati dal J2000.
- ***oscltx***: utilizzata per ricavare i parametri orbitali necessari al calcolo delle equazioni di *Keplero*.

2.2.2 I NAIF ID

I corpi celesti presenti all'interno dei kernel non possono essere rappresentati unicamente con il nome comune con cui sono conosciuti, per questo motivo si è resa necessaria la creazione di un sistema di identificazione univoco degli elementi: i *NAIF ID* (Tabella 2.1).

Le cifre che compongono l'identificativo assumono tutte un particolare significato, la prima fa riferimento al baricentro attorno a cui orbita o ruota l'oggetto in questione, le successive due cifre permettono di differenziare tra pianeti e satelliti, per la rappresentazione dei pianeti viene utilizzato il numero 99, i satelliti seguono una numerazione sequenziale progressiva a partire dallo 01 basata sulla distanza dal proprio padre. Per comprendere con più chiarezza, la Terra avrà come identificativo univoco il 399, 3 in quanto terzo pianeta del sistema solare, mentre la Luna 301 poiché primo satellite non artificiale del nostro pianeta.
Per i pianeti nani la notazione risulta leggermente diversa, eccezion fatta per Plutone che segue la stessa logica dei primi 8 pianeti del sistema solare.

Più complesso ricavare gli ID delle spacecraft in quanto negativi e non propriamente legati al pianeta o satellite riguardante la missione spaziale.

Nell'eventualità che un oggetto non dovesse disporre dei kernel, la notazione adottata all'interno dei cataloghi di *ADN* sarà quella di porre il suo *NAIF ID* a 0. Ciò non esclude, in futuro, la possibilità di calcolare i kernel anche per quel corpo.

Nome comune	***NAIF* ID**	**Nome comune**	***NAIF* ID**
Mercurio	199	Tethys	603
Venere	299	Dione	604
Terra	399	Rhea	605
Luna	301	Titano	606
ISS	0	Iapetus	608
Gaia	-123	Urano	799
Marte	499	Ariel	701
Phobos	401	Umbriel	702
Deimos	402	Titania	703
TGO	-143	Oberon	704
Giove	599	Nettuno	899
Io	501	Tritone	801
Europa	502	Plutone	999
Ganymede	503	Charon	901
Callisto	504	Haumea	2136108
Saturno	699	Makemake	2136472

Tabella 2.1: *NAIF* ID dei corpi presenti in *Astra Data Navigator*

2.3 Position Manager Service (PMS)

Servizio web sviluppato da Eugenio Topa all'interno del programma spazio nella commessa H20202 di *NEANIAS*, può essere utilizzato da *Astra Data Navigator* per implementare gli *SPICE kernel* all'interno del proprio universo virtuale.

Espone solo alcune delle funzionalità offerte dagli *SPICE*, è tuttavia in via di sviluppo l'integrazione di nuove API. Dispone di un dataset di kernel in grado di calcolare le geometrie di quasi tutti gli elementi del nostro sistema solare.
Il servizio è caricato all'interno di un cluster Kubernetes[3] come web application che mette a disposizione diverse API.

Gli utenti possono controllare per quali oggetti possono utilizzare il *PMS* accedendo alla risorsa *list/bids* la quale risponde con la lista intera di ogni corpo presente all'interno dei kernel caricati dal servizio.

Per quanto riguarda il flusso operativo tra il *PMS* e *ADN*, quest'ultimo invia richieste web contenenti un *x-www-form-urlencoded* con al suo interno le informazioni necessarie ad eseguire i calcoli con gli *SPICE* ed attende le risposte codificate in *JSON*.

2.4 Equazioni di Keplero

Johannes Kepler (1561 - 1630) è stato un astronomo, astrologo, matematico e cosmologo tedesco che scoprì empiricamente le omonime leggi che regolano il movimento dei pianeti. Le tre leggi di Keplero rappresentano un modello di descrizione del moto dei pianeti del sistema solare:

- L'orbita descritta da ogni pianeta nel proprio moto di rivoluzione è un'ellisse di cui il Sole occupa uno dei due fuochi.
- Durante il movimento del pianeta, il raggio che unisce il centro del Pianeta al centro del Sole (distanza perifocale) descrive aree uguali in tempi uguali.
- Il quadrato del periodo di rivoluzione di un pianeta è proporzionale al cubo della sua distanza media dal Sole.

Il lavoro svolto all'epoca da questo matematico ha permesso di gettare le basi per la comprensione dei moti orbitali di pianeti e satelliti, ad oggi grazie all'utilizzo degli *SPICE kernel* è possibili ricavare i parametri orbitali dei corpi celesti con i quali si possono calcolare le loro orbite.

Per *Astra Data Navigator* era necessario sviluppare una soluzione stand-alone a cui non servissero risorse esterne per creare il verosimile movimento dei pianeti. Per questo motivo si è deciso di utilizzare i modelli orbitali formulati da Keplero che all'epoca, per ovvie ragioni, non tenne conto di contributi secondari, terziari (e così

[3]sistema open-source di orchestrazione e gestione di container

via) degli altri corpi celesti orbitanti attorno al Sole, e nemmeno della correzione necessaria dei leap second per riallineare il Tempo Coordinato Universale (UTC) al giorno solare medio.

I parametri orbitali di un corpo, necessari per il calcolo della sua orbita sono i seguenti:

- ***ρ distanza perifocale***, distanza tra il centro del corpo celeste ed il centro del Sole.
- ***ϵ eccentricità***, misura di quanto l'orbita sia deviata rispetto ad un cerchio.
- ***M anomalia vera***, angolo compreso tra il pericentro (argomento del periapside) dell'orbita e la posizione del corpo orbitante nel tempo di riferimento, misurato sul piano orbitale.
- ***L longitudine del nodo ascendente***, angolo rispetto al Sole compreso tra il Punto d'Ariete (o Punto Vernale) ed il nodo ascendente del corpo.
- ***ω argomento del periapside***, angolo tra il vettore che punta al periapside ed il vettore che punta al nodo ascendente.
- ***I inclinazione***, distanza angolare del piano orbitale dal piano di riferimento.

Attraverso questi parametri è possibile calcolare l'orbita in due dimensioni sul piano XY utilizzando la formula: $R = \begin{Bmatrix} \frac{\rho * \cos M}{1+\epsilon * \cos M} \\ \frac{\rho * \sin M}{1+\epsilon * \cos M} \\ 0 \end{Bmatrix}$, successivamente occorre applicare una matrice di trasformazione in modo da portare l'orbita bidimensionale appena calcolata in tre dimensioni:

$$T = \begin{Bmatrix} \cos L * \cos\omega - \sin L * \sin\omega * \cos I & -\cos L * \sin\omega - \sin L * \cos\omega * \cos I & \sin L * \sin I \\ \sin L * \cos\omega + \cos L * \sin\omega * \cos I & -\sin L * \sin\omega - \cos L * \cos\omega * \cos I & -\cos L * \sin I \\ \sin\omega * \sin I & \cos\omega * \sin I & \cos I \end{Bmatrix}$$

Per semplicità tutti i parametri orbitali sopracitati ed utilizzati all'interno di *ADN* sono stati calcolati nell'istante di tempo *2019 JAN 00:00:00.000* nonostante essi cambino nel corso del tempo ed in base a dove il corpo si trova lungo la propria orbita. Si è effettuata questa scelta per fornire all'utente una soluzione graficamente credibile ad elevate prestazioni seppur non attendibile a livello scientifico.

2.5 Hardware

Negli ultimi anni sempre più sistemi di interazione utente-macchina hanno aggiunto ai loro requisiti la possibilità di versioni supportanti la stereoscopia 3D. Per ottenere il risultato di tridimensionalità occorre modellare il sistema affinché vengano prodotte due diverse immagini dello stesso frame, una destinata all'occhio destro ed una destinata all'occhio sinistro.
Per far in modo che il cervello umano misceli le due immagini in una sola tridimensionale occorre farle differire della distanza interpupillare (IPD), ovvero la distanza tra i due centri della pupilla dell'occhio destro e sinistro.

Il problema maggiore di questo tipo di tecnologia è racchiuso nell'unicità del corpo umano, ogni persona differisce leggermente rispetto dalle altre, di conseguenza risulta necessario un compromesso che non causi fastidio o senso di nausea quando si utilizza l'applicazione.
Esistono due diversi modi per ottenere l'effetto di tridimensionalità:

- ***stereoscopia passiva***, sistema basato sulla capacità degli schermi o proiettori di visualizzare le due immagini contemporaneamente. Per far in modo che i due occhi ricevano una sola delle immagini esse verranno polarizzate una verticalmente ed una orizzontalmente, l'utente in questo caso dovrà avvalersi dell'utilizzo di particolari occhialini, a loro volta polarizzati nello stesso verso delle immagini, in modo che i due occhi possano captare solamente quella a loro dedicata.
  Nel caso in cui l'utente dovesse ruotare accidentalmente la testa l'effetto 3D si perderebbe all'istante, in quanto l'immagine dedicata all'occhio destro verrebbe catturata dal sinistro e viceversa.

  Esistono anche sistemi a polarizzazione circolare che risolvono quasi completamente questo tipo di problema.
  Tecnologia più datata ma ancora utilizzata è quella degli anaglifi che consiste nell'utilizzo di immagini e occhialini muniti di filtri di colore complementare l'uno rispetto all'altro, tipicamente viene utilizzato il colore rosso per il canale dell'occhio sinistro ed il colore ciano per quello destro.

- ***stereoscopia attiva***, in questo caso le due immagini non vengono proiettate simultaneamente ma rapidamente alternate. Gli occhiali dovranno essere in grado di oscurare in rapida sequenza prima l'occhio destro e poi l'occhio sinistro. L'effetto può essere ottenuto grazie alla composizione a cristalli liquidi delle lenti che in base alla tensione applicatagli possono oscurarsi oppure divenire trasparenti.
  Questo sistema necessita di precisa sincronizzazione tra schermo (o proiettore) e occhialini.

Figura 2.5: Laboratorio VR dell'*ALTEC*

Figura 2.6: Proiettore stereoscopico *BARCO*

ALTEC dispone di un laboratorio VR (Figura 2.5) dotato di un proiettore stereoscopico attivo *Barco RLM-W14* (Figura 2.6) con risoluzione di 1920x1200 pixel ed un refresh rate di 120Hz, che utilizza degli occhialini sincronizzati tramite un segnale ad infrarossi.

Astra Data Navigator è stato progettato per funzionare su una macchina *Dell Precision Tower 7810*, che dispone di due processori *Intel Xeon E5-2650 v3* (ciascuno con 10 core e una frequenza di 2,30GHz), 32GB di RAM ed una scheda grafica *NVIDIA Quadro K5200*.

Capitolo 3

Concetti e Sviluppo

3.1 L'architettura

Come accennato in precedenza l'obiettivo principale del progetto di tesi era quello di introdurre il concetto di tempo all'interno della già esistente simulazione dell'universo. Inizialmente, i corpi celesti caricati all'interno della scena, venivamo solamente posti all'esatta posizione, calcolata attraverso gli *SPICE*, e successivamente non più mossi. Per riprodurre fedelmente il comportamento temporale degli oggetti occorreva introdurre una logica capace di calcolare le variazioni di posizioni tenendo conto dello scorrere del tempo che a sua volta poteva essere velocizzato dall'utente.

Il progetto Unity *Astra Data Navigator* è composto da una sola scena, contenente diversi elementi a cui sono demandate l'inizializzazione, attraverso la lettura dei cataloghi, dei corpi celesti ed il conseguente popolamento dell'universo, la successiva gestione in tempo reale degli oggetti appena creati e l'interazione utente.

Il diagramma di flusso 3.1 illustra una panoramica semplificata dell'esecuzione dell'applicazione, dalla fase di inizializzazione fino alla gestione dell'interazione utente.

3.2 I cataloghi dei corpi celesti

L'applicazione in fase di inizializzazione legge da un file di testo quali corpi celesti generare all'interno della scena, ogni oggetto presente nei cataloghi deve avere un corrispondente *Prefab*[1] in *Unity* creato in fase di progettazione, necessario in quanto i corpi celesti presenti all'interno di *ADN* richiedono delle texture uniche. È tuttavia considerata la possibilità di estendere la generazione degli oggetti attraverso un sistema procedurale in grado di creare corpi generici a cui non vengono applicate

[1] permette di creare, configurare e salvare un *GameObject* completo di tutti i suoi componenti e proprietà

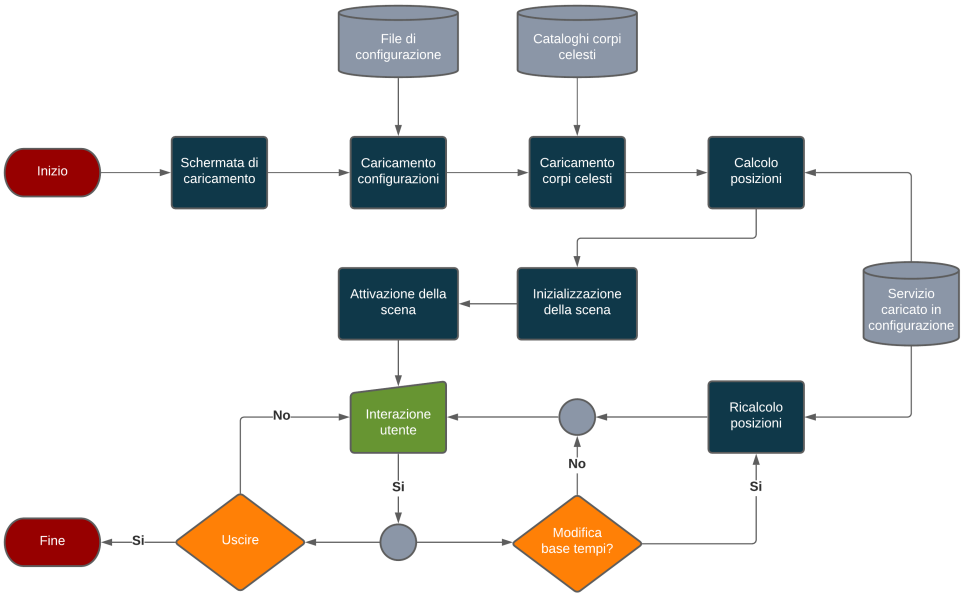


Figura 3.1: Diagramma di flusso di *Astra Data Navigator*

le texture precise ma unicamente i parametri geometrici ricavati dal catalogo. Attualmente l'utente non ha la possibilità di accedere ai cataloghi e di modificarli.

I cataloghi vengono differenziati per tipi di oggetto, uno per i pianeti ed uno per i satelliti. Nonostante le tipologie di corpi presenti abbiano la maggior parte dei parametri in comune è stato comunque necessario diversificarli a causa delle minime differenze. Non essendo di grandi dimensioni si è optato per l'utilizzo di semplici file testuali privi di alcuna serializzazione, come per esempio JSON, caricabili come *Unity TextAsset* all'interno dell'editor.

3.2.1 I parametri

I parametri necessari ed obbligatori al fine del caricamento di un oggetto all'interno della scena vengono ricavati attraverso l'utilizzo degli *SPICE* kernel, calcolati in data *2019 JAN 01 00:00:00.000* ed assunti tutti per costanti, nonostante nella realtà essi cambino nel corso del tempo. Questa scelta si è resa necessaria, come accennato nei capitoli precedenti, al fine di produrre una soluzione sostenibile da qualsiasi macchina su cui si esegue l'applicazione. La data scelta è priva di alcun valore scientifico ed è possibile modificarla ricalcolando tutti i parametri all'interno dei cataloghi.

Le soluzioni per il moto di rivoluzione dei corpi celesti che sfruttano il supporto della libreria *SPICE* non terranno conto di alcuni dei parametri all'interno dei cataloghi in quanto ricalcoleranno puntualmente le informazioni necessarie.

Di seguito una breve spiegazione di tutti i parametri differenziati per categoria.

- ***Parametri generici***, necessari a classificare e memorizzare correttamente i corpi celesti.
  - ***Name***, nome comune dell'oggetto.
  - ***Category***, categoria del pianeta (parametro non presente nel catalogo dei satelliti).
  - ***Parent***, corpo attorno a cui orbita.
  - ***NAIF ID***, identificativo univoco nel sistema *SPICE* (Paragrafo 2.2.2).
  - ***Scale***, diametro del corpo.
- ***Parametri di posizione***, necessari a collocare correttamente nello spazio i corpi celesti.
  - ***X / Y / Z***, posizione nel frame di riferimento *J2000*.
- ***Parametri di rotazione***, necessari a ruotare correttamente nel tempo i corpi celesti.
  - ***PHI / THETA / PSI***, angoli di rotazione del corpo rispetto al piano equatoriale del padre.
  - ***AV MAGNITUDE***, velocità angolare dell'oggetto.
- ***Parametri orbitali***, necessari per realizzare i modelli di Keplero (Paragrafo 2.4).
  - ***Period***, periodo di rivoluzione del corpo.
  - ***RP***, distanza perifocale.
  - ***ECC***, eccentricità.
  - ***INC***, inclinazione.
  - ***LNODE***, longitudine del nodo scendente.
  - ***ARGP***, argomento del periapside.
  - ***NU***, anomalia vera.
  - ***GM***, parametro gravitazionale riferito al corpo attorno a cui orbita.

All'interno dell'applicazione gli script demandati al caricamento dei cataloghi sono *AdnPlanetDefaultLoader* e *AdnSatelliteDefaultLoader* i quali sono interfacce di *AdnLoader*.

In futuro verrà esplorata la possibilità di estendere i cataloghi anche ad asteroidi ed altri corpi celesti.

Parametro	**Tipo**	**Unità di misura**	**Precisione**	**Necessario**	**Esempio**
Name	`String`	NaN	NaN	Si	Earth
Category	`String`	NaN	NaN	Si	Terrestrial Planet
Parent	`String`	NaN	NaN	Si	Sun
NAIF ID	`Int`	*Numero puro*	10^0	Si	399
Scale	`Float`	Km	10^{-4}	Si	12756.2732
X	`Float`	Km	10^{-8}	Si	-25546746.65177911
Y	`Float`	Km	10^{-8}	Si	132914684.98109049
Z	`Float`	Km	10^{-8}	Si	57618024.68305236
PHI	`Float`	Deg	10^{-4}	Default 0	169.8297
TETHA	`Float`	Deg	10^{-4}	Default 0	0.1058
PSI	`Float`	Deg	10^{-4}	Default 0	90.1218
AV MAGNITUDE	`Float`	Deg/s	10^{-8}	Default 0	0.00417807
Period	`Float`	s	10^{-4}	Default 0	31580071.6077
RP	`Float`	Km	10^{-4}	Default 0	147100052.7024
ECC	`Float`	*Numero puro*	10^{-8}	Default 0	0.01715103
INC	`Float`	Deg	10^{-6}	Default 0	23.436447
LNODE	`Float`	Deg	10^{-6}	Default 0	0.001527
ARGP	`Float`	Deg	10^{-6}	Default 0	101.802062
NU	`Float`	Deg	10^{-6}	Default 0	358.197675
GM	`Float`	Km^3/s^2	10^{-4}	Default 0	132712440023.3100

Tabella 3.1: Panoramica dei parametri dei cataloghi

3.2.2 AdnCatalogueGenerator

Dato l'elevato numero di parametri richiesti dall'applicazione e la scarsa reperibilità di alcuni di questi, è stato creato un tool di facile utilizzo a disposizione di coloro che in futuro volessero accrescere i cataloghi di *ADN*. Scritto in *Python* utilizza la libreria *SpiceyPy* [18], un wrapper del *Toolkit* di *NAIF* (Appendice A.3).

Attraverso l'uso di *SpiceyPy* è stato possibile caricare una vasta gamma di kernel in modo da calcolare facilmente tutti i parametri necessari ai cataloghi, ciò nonostante alcuni oggetti all'interno di *ADN* non dispongono ancora dei kernel in quanto non ancora calcolati dalle agenzie spaziali.

Nell'eventualità che il generatore di cataloghi non riesca a calcolare un parametro ritenuto indispensabile, domanda all'utente l'immissione manuale. Alcuni di questi non essendo indispensabili al corretto funzionamento dell'applicazione vengono direttamente segnati a 0.

Viene successivamente gestita, all'interno di *ADN*, in fase di caricamento dei dati, la casistica dei parametri a 0. Nella tabella 3.1 viene presentata una panoramica completa dei parametri dei cataloghi indicando quali risultano necessari e quali invece vengono automaticamente impostati a 0 se non trovati, vengono inoltre illustrate le unità di misura e la precisione adottata.

Utilizzando la funzione *spkpos* messa a disposizione dalla libreria degli *SPICE* è stato possibile calcolare l'esatta posizione di un corpo dato un istante di tempo ed un frame di riferimento. Il *J2000*, frame di riferimento scelto, come accennato precedentemente, è basato sul piano equatoriale della Terra centrato nel Sole e calcolato il *2000 JAN 01 12:00:00.000 TDB* (Figura 3.2) il quale non coincide con il sistema di riferimento adottato in Unity (Figura 3.3), è quindi necessario, in fase di caricamento, applicare una trasformazione ed una rotazione in modo da rendere i due sistemi coerenti. Questo viene eseguito ogniqualvolta viene calcolata una posizione con gli *SPICE*.

Il generatore è completamente modificabile ed espandibile in linea con le future implementazioni all'interno di *ADN*. Non viene esclusa la possibilità di serializzare i risultati nonostante questo possa generare un overhead di memoria a causa dell'attuale quantità di oggetti rappresentabili.

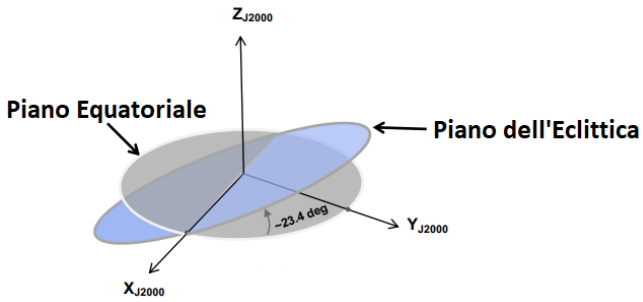


Figura 3.2: Sistema di coordinate *J2000*

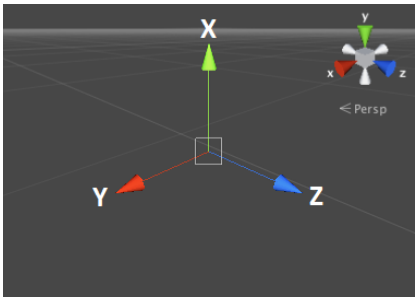


Figura 3.3: Sistema di coordinate *Unity*

3.2.3 AdnCelestialOrbiter

La classificazione di un oggetto all'interno della scena si basa sul tipo e soprattutto sulle azioni che esso deve compiere. Tutti i corpi celesti presenti nell'universo verranno creati come *AdnCelestialObject* contenente le informazioni generiche condivise da pianeti, satelliti e stelle, quali nome e posizione, sono tuttavia esclusive le informazioni riguardanti la rivoluzione presenti unicamente in oggetti orbitanti nello spazio. Viene quindi creata una classe specifica, la quale identifica tutti quei corpi celesti nei quali è necessario salvare i parametri presentati nei paragrafi precedenti, *AdnCelestialOrbiter* è una classe astratta che eredita da *AdnCelestialObject* e ne incrementa le informazioni.

Nel momento in cui viene istanziato un oggetto di tipo *AdnCelestialOrbiter* ne viene calcolata la sua matrice di trasformazione T necessaria per la rappresentazione della rivoluzione tramite le equazioni di Keplero (Paragrafo 2.4), mentre il suo raggio vettore R verrà calcolato unicamente quando richiesto in quanto i parametri utilizzati vengono modificati nel tempo.

3.3 File di configurazione

Tra i requisiti di progetto, l'utente ha la possibilità di configurare alcuni aspetti del comportamento di *ADN*, ciò viene realizzato attraverso l'utilizzo di un file di configurazione, chiamato `config.xml`, situato all'interno della cartella dell'applicazione, e lo script *AdnSettingManager*.

La scelta è ricaduta sul formato XML per via della facilità con cui anche gli utenti meno esperti possono modificarlo, è inoltre facilmente leggibile in quanto diviso in sezioni dedicate ad ogni campo di configurazione. Attraverso la libreria `System.xml`, il linguaggio C# supporta i formati XML.
Il file è suddiviso in macro-sezioni chiamate nodi con al loro interno delle sottosezioni, entrambe contengono alcune configurazioni modificabili dall'utente:

- ***Execution***, permette di specificare la modalità di esecuzione dell'applicazione, può essere impostata a [PC, WebGL, Headset].
- ***Revolution***, consente di configurare il metodo di rivoluzione utilizzato da *ADN*:
  - ***Method***, specifica quale metodo utilizzare tra [SpiceKernel, PMS, Kepler].
  - ***Source***, campo utilizzato solo se si impostano come metodo SpiceKernel o PMS, specifica il path del meta kernel oppure l'indirizzo IP del servizio da utilizzare.
- ***StarLoader***, contiene le specifiche di caricamento per le stelle.
- ***Simulation***, permette di cambiare alcune opzioni riguardanti la simulazione quale data di inizio.

- ***Logger***, specifica il comportamento del logger degli eventi dell'applicazione.

Lo script demandato al caricamento delle configurazioni iniziali è *AdnSettingsManager* il quale viene eseguito prima di ogni altra direttiva all'interno dell'applicazione, questo avviene grazie all'inserimento dell'ordine di esecuzione (`[DefaultExecutionOrder(-1000)]`) in cima al file, il quale comunica a *Unity* quale script eseguire prima degli altri.

All'interno della funzione *Awake* viene aperto il file XML, creato un oggetto `XmlDocument` sul quale è possibile invocare il metodo *SelectSingleNode* con cui vengono letti i valori contenuti nel file e salvati all'interno di variabili dichiarate `public static` accessibili da ogni altra parte del codice. Nel caso in cui occorrano errori di lettura, errori di parsificazione oppure venissero violate le condizioni di accettabilità dei valori, *ADN* ricadrebbe in una prestabilita configurazione di default decisa in fase di progettazione ignorando parzialmente o completamente il file di configurazione.

3.4 Gestione e componenti della rivoluzione

Avvenuto il caricamento delle impostazioni e la successiva generazione di tutti i corpi celesti, all'interno di *ADN* si inizia a simulare lo scorrimento del tempo e di conseguenza pianeti e satelliti inizieranno ad orbitare nello spazio.

Per effettuare la traslazione di un oggetto in *Unity* è necessario modificare il suo componente *transform* di una quantità pari a quanto lo si vuole spostare nelle 3 dimensioni. Come spiegato nel paragrafo 2.2, quando si ha a che fare con unità di misura astronomiche occorre sfruttare particolari tecniche al fine di evitare perdita di precisione. Per questo motivo in *ADN* non è possibile modificare direttamente la *transform* dell'oggetto bensì è necessario accedere al campo `Position` memorizzato all'interno dell'`SgtFloatingPoint`, il quale successivamente ne modificherà la *transform*.

Nei paragrafi successi vengono illustrati i componenti sviluppati per la gestione della rivoluzione.

3.4.1 AdnRevolution

Componente principale e fondamentale in grado di muovere gli oggetti a cui è attaccato adattandosi nel tempo alle eventuali alterazioni della velocità della simulazione.

AdnRevolution è un componente *MonoBehaviuor* da agganciare unicamente ai corpi celesti di cui è possibile gestirne la rivoluzione, non richiede alcun parametro aggiuntivo se non quelli forniti dal file di configurazione.

Non essendo possibile conoscere a priori il metodo con cui verrà gestita la rivoluzione durante l'esecuzione dell'applicazione, poiché parametro configurabile da parte dell'utente, *AdnRevolution* è stato programmato per ricevere da *AdnSettingsManager* l'istanza creata chiamando il metodo `GetServiceRevolution`.

Quest'ultimo dopo aver letto dal file `config.xml` quale servizio utilizzare, ne genera la sua istanza per *reflection* attraverso il metodo `GetConstructor`, invocabile sul tipo da istanziare. Al fine di gestire correttamente gli eventuali errori, viene generata in ogni caso l'istanza del servizio di default la quale non richiede alcuna risorsa esterna. I tre servizi attualmente utilizzabili all'interno di *ADN* vengono presentati nel paragrafo 1.3.1.

Il comportamento successivo di *AdnRevolution* dipende dal tipo di metodo configurato.

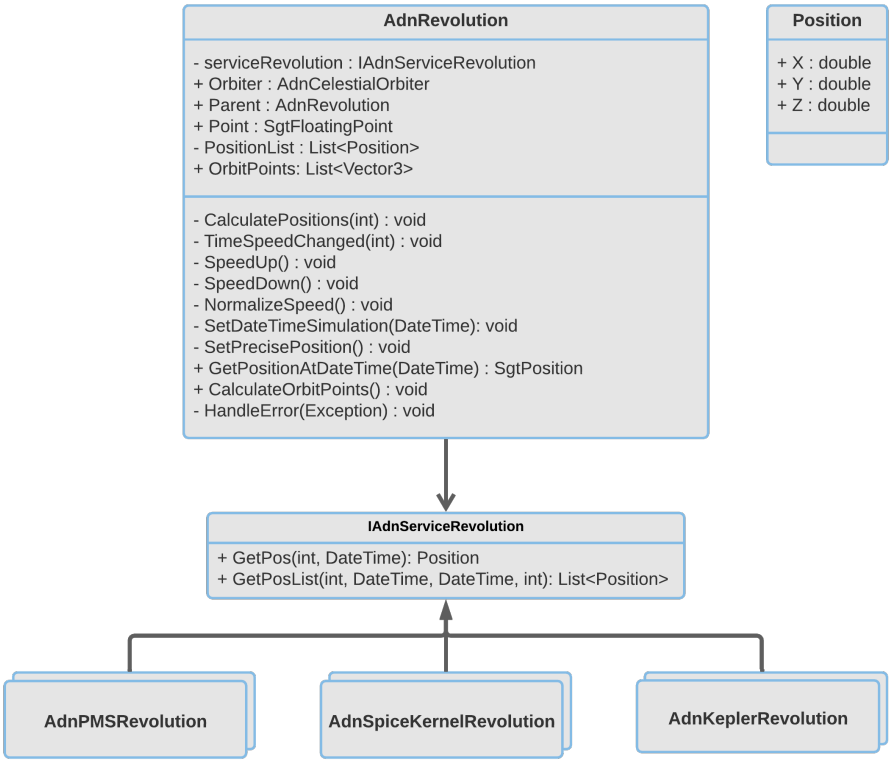


Figura 3.4: Diagramma delle classi di *AdnRevolution*

Rivoluzione dipendente dagli *SPICE*

L'utilizzo di questa soluzione è consigliata quando l'utente ricerca la precisione nell'evoluzione temporale delle posizioni dei corpi celesti.

Il primo controllo compiuto da *AdnRevolution* rispetto all'oggetto a cui è agganciato è la disponibilità dei kernel per quest'ultimo, in caso negativo richiede ad *AdnSettingsManager* l'istanza della rivoluzione di default che successivamente verrà utilizzata per quel corpo celeste.

Superati i controlli preliminari occorre calcolare una lista di posizioni da salvare in cache da utilizzare nel corso della simulazione. Ogni richiesta di calcolo di

posizioni da parte di *AdnRevolution* al servizio configurato richiede un tempo considerevole, occorre quindi trovare il giusto compromesso tra tempo di richiesta e numero di posizioni in modo da non sovraccaricare la memoria.

La precisione con cui il sistema muove i corpi celesti rimane un parametro configurabile unicamente dal programmatore. Attualmente vengono calcolate posizioni con un intervallo di 60 secondi l'una dall'altra per un periodo di 24 ore per un totale di 1440 terne di coordinate memorizzate utilizzando il metodo `GetPosList` (Appendice A.1). Tutte le posizioni intermedie tra le due calcolate vengono interpolate ottenendo risultati non molto distanti da quelle che sarebbero le posizioni effettive. Diminuire l'intervallo in secondi, incrementerebbe di poco la precisione aumentando notevolmente l'utilizzo della memoria ed il tempo di calcolo.

In prima istanza questo calcolo viene eseguito sequenzialmente per ogni *AdnRevolution* presente nella scena non curandosi del blocco di tutte le altre funzionalità in quanto non si è ancora entrati effettivamente all'interno dell'universo esplorabile ma si è ancora nella schermata di caricamento iniziale. Proseguendo nel tempo le posizioni memorizzate vengono consumate e risulterà quindi necessario calcolarne di nuove rispettando le regole sopracitate. In questo caso non sarà più possibile effettuare calcoli sequenziali ma si sarà costretti a demandare a thread paralleli la gestioni di questi ultimi. Ciò è stato realizzato utilizzando i `Task` messi a disposizioni dalla libreria `System.Threading.Task` di C#. Tutti i calcoli successivi a quelli iniziali verranno eseguiti in modo concorrenziale.

Il ricalcolo delle posizioni, come accennato precedentemente, è un processo che richiede un lasso di tempo non trascurabile e non sempre definito, nel caso di utilizzo del servizio PMS è da considerare l'indeterminatezza dei tempi di trasmissione nella rete, per questo motivo viene eseguito non appena vengono consumate metà delle posizioni calcolate all'iterazione precedente, in modo da evitare il più possibile l'insorgere di errori causati dalla mancanza di posizioni in memoria.

Ogni tipo di errore incorso durante l'esecuzione della simulazione viene notificato e gestito, sfruttando le `Exception` di C#, all'interno di *AdnRevolution* il quale comunicherà all'utente l'accaduto e passerà la gestione della rivoluzione dell'oggetto al servizio di default.

Attraverso l'interfaccia grafica, l'utente ha la possibilità di alterare il normale scorrimento della simulazione velocizzandola. Quando il tempo trascorso tra due frames successivi diverrà maggiore dei secondi configurati tra due posizioni calcolate attraverso gli *SPICE* occorrerà modificare la base tempi aumentando i secondi di intervallo, questo però comporterà perdita di precisione. In questo caso la modifica alla velocità verrà resa effettiva successivamente alla notifica da parte di ogni *AdnRevolution* di termine dei calcoli in modo da non incorrere in errori indesiderati.

Rallentando la simulazione dopo averla velocizzata si ricadrebbe nel caso in cui la base tempi iniziale possa nuovamente essere utilizzata, per questo motivo verrà ricalcolata sovrascrivendo la lista precedente al fine di evitare eccessivi sprechi di memoria.

Quando la simulazione viene messa in pausa viene effettuata una richiesta puntuale, ovvero utilizzando ora, secondi e millisecondi attuali, agli *SPICE* tramite

il metodo `GetPos` garantendo precisione massima.

AdnRevolution mette a disposizione un metodo pubblico invocabile da ogni altro punto del codice chiamato `GetPositionAtDateTime` con il quale gli altri script di *ADN* possono richiedere la posizione di un oggetto in un dato istante di tempo futuro o passato.

Rivoluzione indipendente dagli *SPICE*

Nei casi in cui *ADN* dovesse venir configurato in modalità indipendente dagli *SPICE* verrà istanziata, da parte di *AdnSettingsManager*, unicamente la classe di default (*AdnKeplerRevolution*).

In simili situazioni non sarà necessario precalcolare una lista di posizioni, bensì sarà sufficiente l'offset di spostamento rispetto al frame precedente. Ciò viene calcolato moltiplicando la velocità angolare del corpo, memorizzata all'interno dell'*AdnCelestialOrbiter*, per il lasso di tempo trascorso tra un frame e l'altro. Trovato l'angolo di spostamento verrà sommato all'anomalia vera per poi ricalcolare il raggio vettore R dell'orbita in due dimensioni a cui occorre infine applicare la trasformazione geometrica T (Paragrafo 2.4). Tale operazione è applicabile qualunque sia la velocità della simulazione, ciò ne semplifica notevolmente la gestione.

3.4.2 cspice.dll

Uno degli obiettivi principali del progetto di tesi era quello di permettere ad *ADN* l'utilizzo degli *SPICE* senza la richiesta di risorse esterne, quali connessioni a servizi remoti in grado di effettuare i calcoli necessari. Il raggiungimento di questo scopo è stato possibile compilando una libreria in C++ basata sul *Toolkit* di *NAIF* in grado di implementare tutte le complesse funzionalità di quest'ultimo.

Precedentemente al lavoro di tesi non era disponibile una versione della dll generata attraverso il compilatore `.NET` di *Visual Studio*[2]. La realizzazione di quest'ultima è stata possibile grazie al lavoro svolto da un team di sviluppatori, i quali hanno creato una *CMake-based build* del *Toolkit*, disponibile presso la loro repository *GitHub* [19], ovvero il file *CMake* in grado di inizializzare il progetto *Visual Studio* con il quale sarebbe successivamente stato possibile compilare la libreria.

Data la notevole varietà di librerie installate all'interno dei diversi PC su cui vengono svolti questi lavori, si è resa necessaria un'attenta analisi di quali fossero indispensabili per il corretto funzionamento della libreria e quali invece si sarebbero potute rimuovere senza comprometterne l'integrità. Una volta ottenuta una versione stabile del *CMake* è stato possibile compilare una prima versione della dll che non

[2]ambiente di sviluppo integrato sviluppato da Microsoft, utilizzato per la creazione di progetti per varie piattaforme

interagiva correttamente con il codice di *ADN*, a causa della mancanza degli *entry point* necessari, ovvero i punti in cui il codice della libreria viene invocato da parte di applicazioni esterne. Per poter definire correttamente tutti gli *entry point* necessari si è dovuto modificare il codice sorgente del *Toolkit* inserendoli manualmente attraverso la direttiva `__declspec(dllexport)` anteposta alle implementazioni delle funzioni.

La libreria *SPICE* è composta di migliaia di funzioni che svolgono ognuna differenti calcoli sui dati planetari, in *ADN* non era necessario integrare tutte le funzionalità disponibili ma solamente un ristretto insieme, per questo motivo si è deciso di creare *entry point* solamente per quelle indispensabili. Nell'eventualità che in futuro dovessero essere incrementate le capacità di *ADN* rimane la possibilità di aggiungere nuovi *entry point* laddove fosse necessario ricompilando successivamente l'intera libreria.

Oltre alle funzionalità citate precedentemente (Paragrafo 2.2) è stato necessario aggiungerne alcune di controllo e configurazione del comportamento della libreria:

- ***erract***, permette di configurare le azioni eseguite in caso di occorrenza di un errore.
- ***unload***, utilizzata per scaricare dalla memoria i kernel precedentemente caricati attraverso la *furnsh*.
- ***ktotal***, consente di notificare all'utente il numero esatto di kernel caricati correttamente, eventualmente divisi per categoria.

Essendo lo *SPICE Toolkit* nativamente scritto in C, contiene al suo interno le dichiarazioni delle funzioni utilizzanti i puntatori nativi del linguaggio. Alcuni di essi non vengono più supportati in linguaggi più recenti, sopratutto nel caso del C# nel quale vengono sostituiti con direttive più articolate e sicure, evitando accessi indesiderati in memoria. Rimane comunque presente la dualità di utilizzo di alcuni di essi in quanto riconosciuti allo stesso modo dai diversi linguaggi. Per questo motivo quando vengono invocate delle funzioni della libreria all'interno di *ADN* (scritto in C#) occorre rispettare delle regole di conversione per il linguaggio C. Gli sviluppatori del *Toolkit* al fine di conservare la manutenibilità a lungo termine hanno optato per mappare i principali tipi nativi del linguaggio C ridefinendoli.

È utile sottolineare il fatto che come tutte le risorse le quali implementano gli *SPICE* anche la dll richiede il caricamento dei kernel attraverso un *metakernel*, con le stesse regole e modalità descritte precedentemente (Paragrafo 2.2).

Allo stato attuale la libreria e lo stesso *Toolkit* di *NAIF* non sono multithread e quindi in grado di parallelizzare correttamente richieste simultanee multiple, *NAIF* stesso dichiara che il comportamento del sistema potrebbe ricadere in uno stato indefinito ed imprevedibile producendo risultati potenzialmente errati. I test effettuati per rendere multithread la dll non hanno prodotto risultati positivi confermando le dichiarazioni effettuate da *NAIF*. Attualmente i programmatori del *Toolkit* affermano di essere al lavoro per produrre una versione parallelizzabile ma che questo potrebbe richiedere diversi anni.

Tipo nativo C	Tipo *SPICE*	Tipo C#
`char*`	`SpiceChar*`	`StringBuilder`
`bool`	`SpiceBoolean`	`bool`
`int`	`SpiceInt`	`int`
`double`	`SpiceDouble`	`double`
`double*`	`SpiceDouble*`	`double*`
`double array[n]`	`SpiceDouble array[n]`	`double array[]`

Tabella 3.2: Principali conversioni e ridefinizioni dei tipi

3.4.3 AdnSpiceKernelRevolution

La classe *singleton* dedicata al caricamento ed all'utilizzo della dll all'interno di *ADN* è *AdnSpiceKernelRevolution*, la quale implementa i metodi esposti dalla classe interfaccia *IAdnServiceRevolution*, essa permette di invocare le funzionalità degli *SPICE* attraverso gli *entry point*.

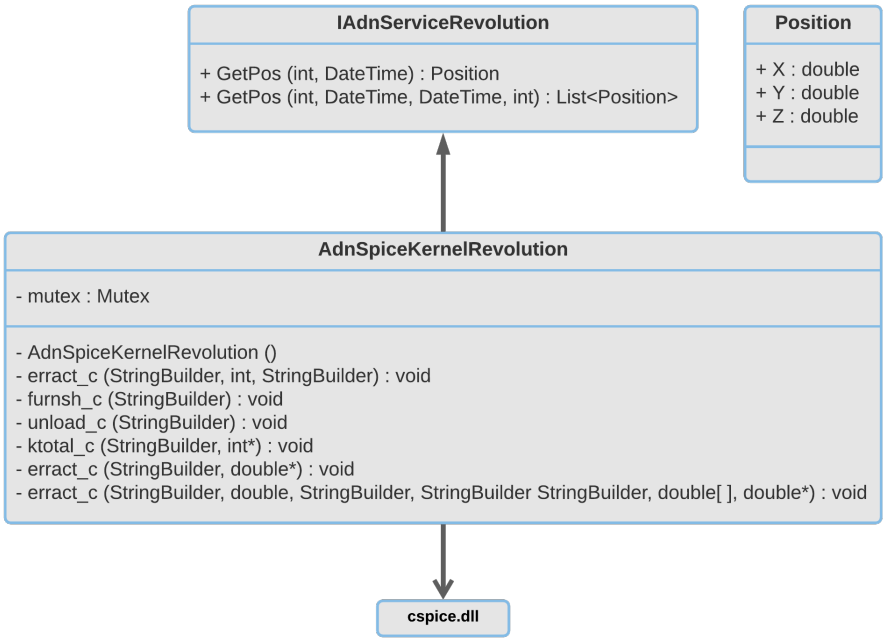


Figura 3.5: Diagramma delle classi di *AdnSpiceKernelRevolution*

La direttiva C# che consente la chiamata di una funzione di una libreria non nativa scritta in C/C++ è `[DllImport("cspice.dll")]` seguita dalla dichiarazione della funzione stessa rispettando le regole di conversione dei tipi descritte nel paragrafo precedente, ne è un esempio `unsafe static extern void furnsh_c(StringBuilder file)`.

Ogni funzione la quale utilizza i puntatori nativi del linguaggio C deve necessariamente essere dichiarata `unsafe` in quanto informa il compilatore di un blocco di codice non verificabile in grado di allocare e liberare blocchi di memoria. Non si tratta di codice necessariamente dannoso ma che semplicemente non è possibile stabilirne la sicurezza attraverso i canonici processi di verifica.

AdnSpiceKernelRevolution all'interno del proprio costruttore privato, invocabile unicamente da *AdnSettingManager*, come prima azione chiama la funzione `erract_c` in modo da configurare correttamente la gestione errori, dopodiché procede al caricamento del *metakernel*, definito dall'utente all'interno del file di configurazione, attraverso la funzione `furnsh_c`. Effettuate le operazioni preliminari di configurazione e caricamento si pone in attesa di richieste di posizioni da parte di *AdnRevolution*. Ogni qualvolta vengono ricevute viene acquisito un `mutex` in modo da rendere la risorsa mutuamente esclusiva rispetto ad altre richieste parallele e non ricadere in errori imprevedibili.

La richiesta di una o più posizioni è composta di diversi campi:

- `int` *naif_id*, identificativo del corpo celeste (Paragrafo 2.2.2).
- `DateTime` *stert_time*, data della posizione richiesta oppure data di inizio dell'intervallo di posizioni.
- `DateTime` *end_time*, data di fine dell'intervallo di posizioni.
- `int` *step*, intervallo in secondi tra una posizione e quella successiva.

Ricevuti i parametri necessari, *AdnSpiceKernelRevolution* procede a convertire la data nel numero di secondi TDB passati dal *J2000* attraverso la funzione `str2et_c` e salvando il risultato all'interno di un `double`. Nel caso di richiesta di intervallo di posizioni converte le due date e ne calcola i secondi che le separano procedendo ad iterare basandosi sullo *step* ricevuto ed effettuando sequenzialmente tante interrogazioni puntuali sommando via via i secondi alla data di partenza. Tutto ciò viene svolto invocando la funzione `spkpos_c` che risponderà con la terna di coordinate rappresentanti la posizione nel frame *J2000* la quale verrà trasformata portandola nel sistema di riferimento di *Unity*. Terminati i calcoli viene rilasciato il `mutex` rendendo nuovamente la risorsa disponibile per nuove richieste.

3.4.4 AdnPMSRevolution

Sviluppo dell'interfaccia *IAdnServiceRevolution*, *AdnPMSRevolution* è una classe *singleton* istanziabile unicamente da *AdnSettingsManager* con le stesse modalità presentate precedentemente. Esso instaura una connessione con il servizio *PMS* (Paragrafo 2.3) al quale richiede di calcolare posizioni quando necessario.

Generalmente tale soluzione del metodo di rivoluzione viene scelta in situazioni in cui *ADN* deve garantire elevata precisione nel mostrare l'andamento temporale delle posizioni dei corpi celesti ma non si dispone delle risorse locali per farlo sfruttando la dll, come kernel o semplicemente potenza di calcolo e memoria sul dispositivo. Il risultato tra questa soluzione e quella locale è perfettamente identico, variano unicamente le modalità con cui vengono reperite le posizioni.

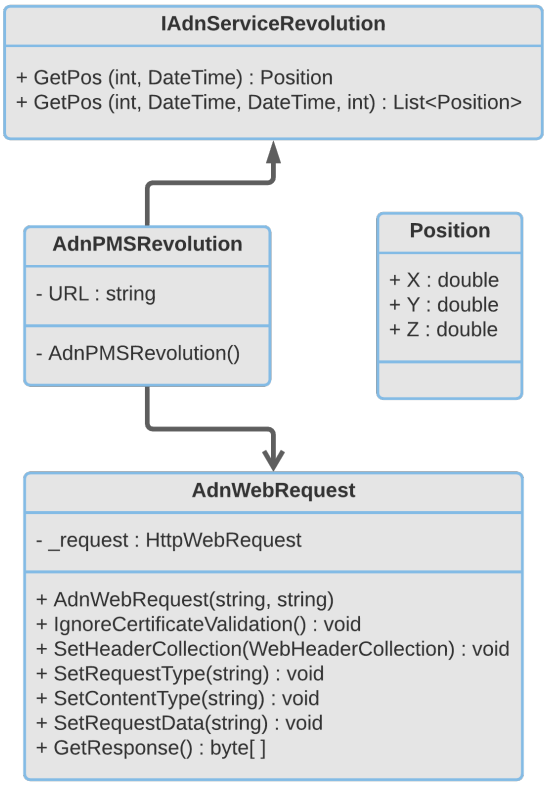


Figura 3.6: Diagramma delle classi di *AdnPMSRevolution*

L'URL del servizio a cui connettersi viene inserito dall'utente all'interno del file di configurazione e memorizzato prima che vengano generati tutti gli oggetti. Dopodiché la classe rimane in attesa delle richieste di posizioni. Ogni qualvolta ne riceve una, provvede a creare un'istanza della classe *AdnWebRequest*, wrapper della classe *WebRequest* di C#, impostandone i parametri necessari all'invio della

richiesta *POST*[3], concatenando l'URL con l'API che si è intenzionati ad utilizzare.

Per prima cosa viene creato un `Dictionary<string, string>` con al suo interno i campi necessari, il quale verrà successivamente serializzato in formato *JSON* per essere contenuto in una sola stringa e salvato dal metodo `SetRequestData` all'interno dell'*AdnWebRequest*. Una volta configurata la richiesta, *AdnPMSRevolution* rimane in attesa della risposta da parte del servizio, essa arriverà codificata in `bytes` e serializzata in *JSON*, per questo motivo occorrerà decodificarla e poi deserializzarla in un oggetto `Position` (contenente la terna di coordinate spaziali) oppure in una `List<Position>` in base al tipo di richiesta effettuata (Appendice A.2).

Nel caso insorgano errori di trasmissioni (o di altro tipo) verrà generata una `WebException` che contrarrà lo stack delle chiamate dei metodi fino a raggiungere *AdnRevolution* che la gestirà adeguatamente e modificherà il proprio comportamento per quell'oggetto.

3.4.5 AdnKeplerRevolution

Realizzazione fittizia, nonché di default, dell'interfaccia *IAdnServiceRevolution* utilizzata unicamente per comunicare agli *AdnRevolution* quale tipo di calcoli effettuare. La classe viene comunque istanziata da *AdnSettingsManager* e mai utilizzata in quanto tutta la logica di calcolo è dislocata in altre parti del codice.

3.4.6 AdnOrbitVisualizer

Quando viene scelto come metodo di rivoluzione *Kepler* è possibile visualizzare, premendo l'apposito bottone situato in basso a destra dell'interfaccia utente, le orbite dei corpi celesti.

Al contrario degli oggetti nella scena, le orbite non sfruttano la tecnica dell'origine flottante, per questo motivo ogni frame in cui devono essere disegnate, occorre ricalcolare il loro punto relativo alla posizione della camera (che invece è un oggetto flottante) attraverso il metodo `CalculatePosition` messo a disposizione da *SGT*.

AdnOrbitVisualizer è un componente da attaccare ad ogni *Prefab* di cui si vuole visualizzarne l'orbita, la direttiva `[RequireComponent(typeof(AdnRevolution))]` posta in cima a questo script assicura la mutua presenza di *AdnRevolution* in cui all'interno vengono calcolati i punti necessari all'orbita.

Mediante il metodo `DrawOrbit` vengono invocate le direttive *OpenGL* con cui è possibile disegnare punti e linee collegandoli tra loro e definendone il colore. Tale funzione viene invocata dalla callback `OnRenderObject` di *Unity* la quale viene eseguita dopo il termine della pipeline di rendering della scena.

[3]metodo di richiesta supportato da *HTTP* nel quale il web server accetta i dati racchiusi nel corpo della richiesta stessa

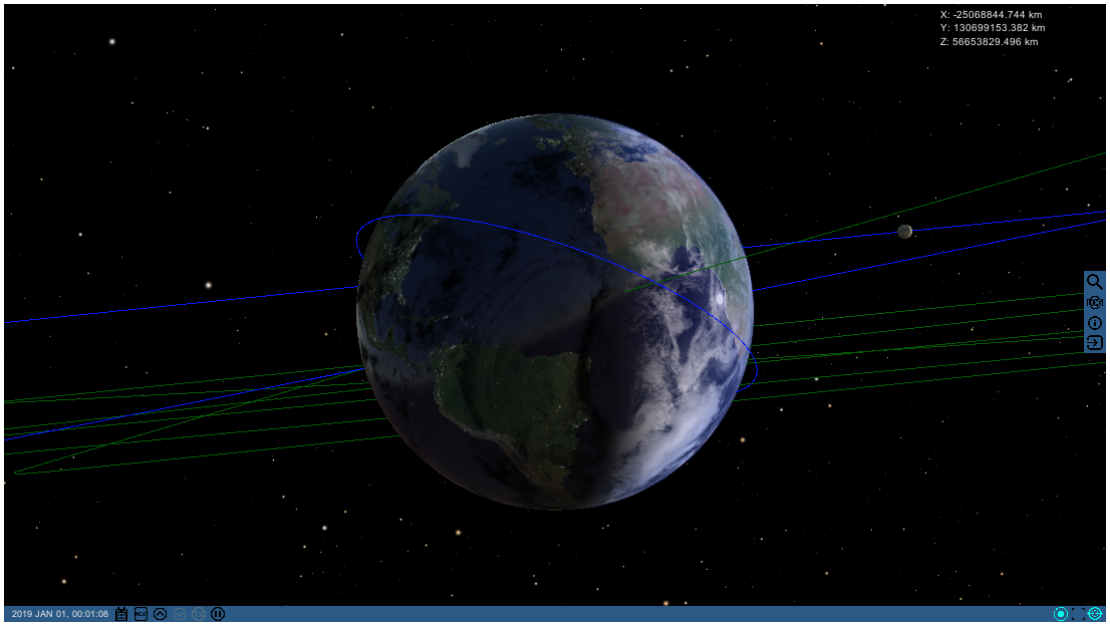


Figura 3.7: Visualizzazione delle orbite tramite l'*AdnOrbitVisualizer*

Occorre prestare molta attenzione all'utilizzo di questa funzione in quanto potrebbe avere grosso impatto sulle prestazioni. Per tale motivo si è deciso di non superare mai i 500 punti per orbita (il valore è dato dal massimo tra 500 ed una formula direttamente proporzionale alla distanza perifocale dell'oggetto) nonostante per i corpi agli antipodi del sistema solare questo potrebbe portare a notare i segmenti di cui è composta l'orbita. Per i satelliti, al contrario, vengono utilizzati costantemente 200 punti poiché non si discostano eccessivamente dall'oggetto attorno a cui orbitano.

La decisione di mostrare questa caratteristica unicamente quando viene utilizzato *Kepler* come metodo di rivoluzione è data dall'impossibilità di calcolare abbastanza punti per le orbite dei pianeti e nano-pianeti eccessivamente distanti dal Sole in quanto non vi è supporto da parte dei kernel. Attualmente per questi corpi non sono stati calcolati sufficienti kernel, non si esclude in futuro di poter accrescere questa caratteristica con il supporto degli *SPICE*.

All'interno di *ADN* per ogni riferimento a pianeti verrà utilizzato il colore verde, mentre per satelliti il colore blu. Nella figura 3.7 si può osservare la Terra con attorno l'orbita dell'ISS e della Luna mentre sullo sfondo si scorgono le orbite degli altri pianeti del sistema solare.

3.5 Interfaccia utente

Lo stile della GUI di *ADN*, nel corso del lavoro di tesi, è stato mantenuto simile a quello della sua precedente versione. Nonostante ciò, sono state apportate numerose modifiche ed aggiunte al fine di migliorare l'esperienza percepita dall'utente (Figura 3.8):

- Poiché l'applicazione in fase di inizializzazione potrebbe richiedere fino a 30 secondi, si è deciso di mostrare una **schermata di caricamento** la quale coprirà completamente la scena finché essa non sarà pronta. Tale schermata è realizzata mediante un componente *Image* di *Unity*, un icona rotante animata ed una scritta aggiornano l'utente riguardo lo stato di inizializzazione in cui si trova l'applicazione. Terminato il calcolo di tutte le posizioni iniziali la schermata viene sfumata fino a diventare completamente trasparente (e successivamente disattivata) utilizzando una *Corutine*.

- Una **schermata di caricamento dinamico**, simile a quella iniziale, viene mostrata nei casi in cui venga modificata la data della simulazione poiché ciò richiede un ricalcolo necessariamente sincrono delle posizioni di tutti i corpi celesti presenti all'interno della scena. È composta da un *Panel* semi-trasparente con un'icona di caricamento statica ed un *Text* in basso a sinistra. Poiché *Unity* aggiorna la UI al termine del frame corrente, la funzione di ricalcolo dovrà essere schedulata al frame successivo oppure l'applicazione si congelerà prima che la **schermata di caricamento dinamico** venga mostrata, ciò viene fatto attraverso la direttiva `yield return null` all'interno della *Corutine*.

- Il **pannello di ricerca** è stato migliorato al fine di lavorare con cataloghi stellari molto ampi senza impattare sulle prestazioni dell'applicazione.

- Viene introdotto un **pannello di modifica della data** al fine di permettere all'utente di inserire manualmente un preciso istante temporale del quale vuole vedere l'universo. Questo pannello è realizzato mediante 6 *InputFiled* e *Placeholder*, in cui inserire la data desiderata, nel caso quest'ultima risultasse errata verrà generata un eccezione.

- È stata sviluppata una gestione personalizzata dei log, ogni record viene salvato all'interno di un file `log.txt` presente all'interno della cartella dell'applicazione. Ne viene fornita una sua versione runtime attraverso un **pannello dei log** all'interno della UI utile a notificare in tempo reale all'utilizzatore eventuali errori o informazioni di carattere generale.

- Un nuovo **pannello informativo** per gli oggetti presenti all'interno della scena dove è possibile trovare tutte le informazioni presenti nei cataloghi rispetto a quel dato corpo, ne sono un esempio i parametri orbitali citati nei

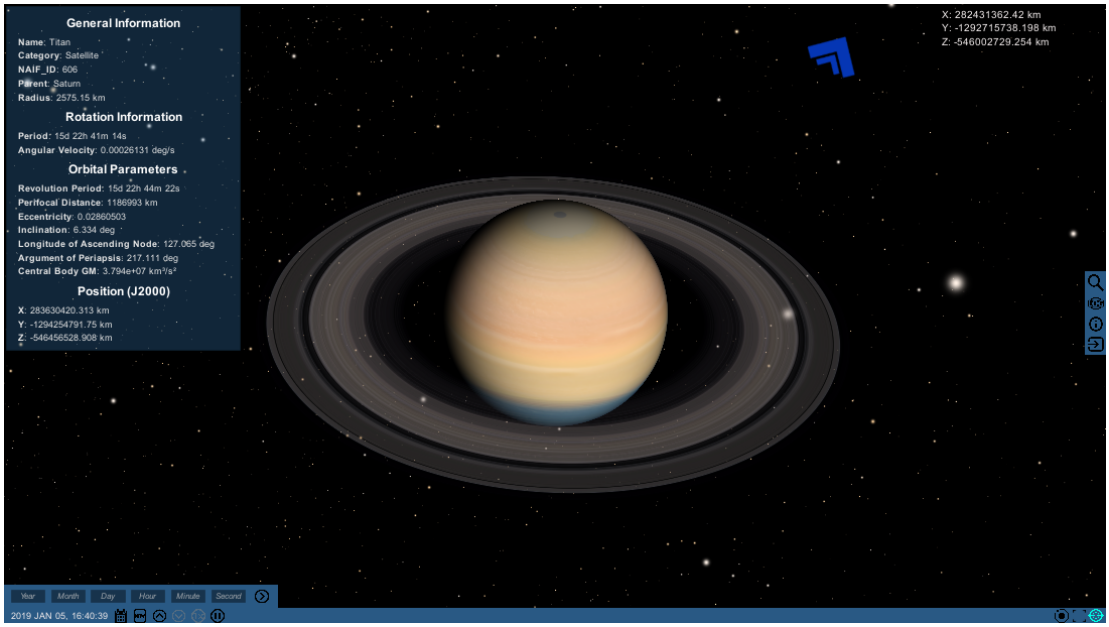


Figura 3.8: Visualizzazione di alcuni componenti della UI di *ADN*

paragrafi precedenti. Tale pannello è altamente modulare in quanto genera ogni entry separatamente l'una dall'altra come *Text* di *Unity* utilizzando la funzione `AddInfoEntry`, risulta quindi facilmente modificabile ed estendibile nell'eventualità che le informazioni riguardanti i corpi celesti aumentino.

- Perdersi all'interno dell'universo è molto semplice, sopratutto quando non si hanno punti di riferimento, per questo motivo è stato sviluppato un **puntatore** che indicherà sempre l'oggetto selezionato nel caso in cui esso dovesse uscire al di fuori del frustum della camera. Tale indicatore viene realizzato utilizzando un componente *Image* ruotato verso il corpo selezionato e fatto slittare sulla cornice del campo visivo dell'applicazione. Esattamente come le orbite, assumerà colori diversi in base alla categoria di oggetto verso il quale punta. In figura 3.8 si può notare Saturno ed il puntatore indicante Titano (una delle sue lune).

3.6 La stereoscopia

Il supporto alla stereoscopia in *ADN* è stato realizzato utilizzando lo *Unity XR Management* plugin installabile direttamente dall'editor e impostato al fine di funzionare con piattaforme *Windows Mixed Reality*. Utilizzando tale plugin non vengono richieste ulteriori configurazioni all'interno di *Unity*. Nel caso in cui venga rilevato uno schermo compatibile l'applicazione verrà automaticamente eseguita in modalità stereoscopica.

La separazione stereoscopica, la quale determina quanto sono differenti l'immagine destinata all'occhio destro rispetto a quella destinata all'occhio sinistro, può essere impostata all'interno del pannello di controllo dei driver grafici (nel caso della macchina su cui è stato progettato *ADN*, *NVIDIA Control Panel*). L'impostazione utilizzata è del valore di 10%, rispetto al 15% di default, poiché parte della UI veniva tagliata fuori dai bordi dello schermo.

Utilizzando la configurazione di default, la stereoscopia incrementava la profondità della scena facendo apparire gli oggetti più lontani rispetto a quanto lo fossero effettivamente, e facendo galleggiare la GUI in primo piano. È tuttavia possibile invertire tale effetto modificando il parametro *Swap eyes* nel pannello di controllo dei driver. Analizzando entrambe le soluzioni si è optato per la prima poiché generava nell'utente un minor affaticamento oculare nonostante l'effetto 3D non fosse marcato a causa degli scarsi punti di riferimento reperibili nello spazio.

Capitolo 4

Risultati e Analisi

4.1 Analisi delle performance

Essendo *ADN* una simulazione realtime le caratteristiche prestazionali giocano un ruolo fondamentale al fine di soddisfare i requisiti richiesti da questo genere di applicazioni. Tempi di caricamento e calcolo troppo elevati, bassi framerate ed eccessivo utilizzo della memoria devono necessariamente essere evitati per poter ambire ad estendere i dataset esistenti senza rischiare di compromettere la fluidità della navigazione.

Nonostante questo fosse un lavoro di modifica di una precedente versione non è possibile confrontare risultati riguardanti il movimento dei corpi celesti poiché esso non era ancora stato inserito, per questo motivo nei paragrafi successivi verrà presentata un'analisi delle prestazioni di calcolo e precisione differenziando tra le varie soluzioni sviluppate.

Il framerate è un dato a cui occorre prestare particolare attenzione, indica il tempo che CPU e GPU necessitano per completare tutti i calcoli per la creazione di un singolo frame, ciò definisce la percezione di fluidità nell'utente. Per ottenere una buona esperienza VR responsiva e che non causi malessere occorre mantenere un framerate di almeno 60 FPS.

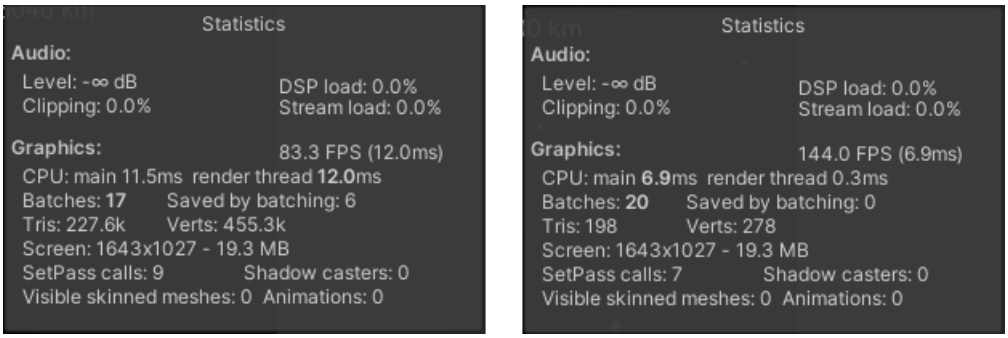


Figura 4.1: Confronto tra l'applicazione originale (sinistra) e la nuova versione di *ADN*

In figura 4.1 è possibile notare una forte riduzione del tempo di utilizzo della CPU ed un sostanziale aumento degli FPS rispetto alla versione originale dell'applicazione.

4.1.1 Performance di calcolo

Come anticipato nei capitoli precedenti, i differenti metodi di rivoluzione comportano differenti tempi di calcolo. Questo è dovuto agli sviluppi interni di ognuna delle soluzioni, utilizzando le equazioni di Keplero ci si aspetta maggior velocità ma minor precisione, con la *cspice.dll*, essendo compilata in C++, si presume di avere precisione e buona velocità ma elevato dispendio di memoria, mentre con il *PMS* si avrà poca velocità ma elevata precisione e quasi per nulla utilizzo di memoria.

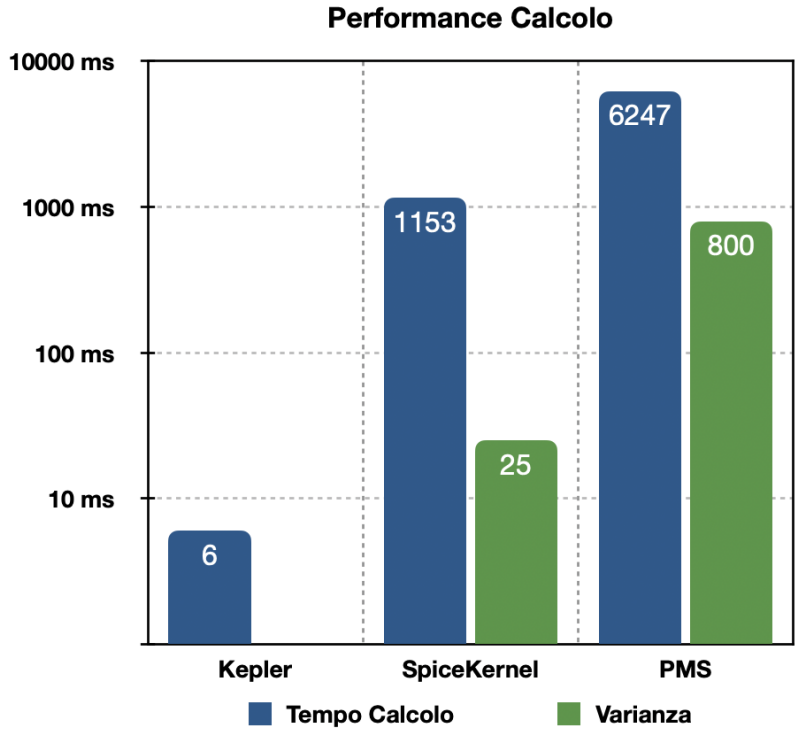


Figura 4.2: Analisi dei tempi di calcolo delle posizioni delle soluzioni sviluppate

Come si può notare dal grafico, i tempi di calcolo risultano esattamente coerenti alle aspettative.

Utilizzando la soluzione basata sulle equazioni di *Keplero* saranno necessari solamente 6 millisecondi in totale per calcolare, per tutti i corpi, i dati necessari alla simulazione. Non è presente varianza del tempo di calcolo poiché essendo il processo estremamente veloce non verrà mai schedulato dalla CPU.

Per quanto riguarda l'uso della *cspice.dll* si ha ancora ottima velocità di calcolo (circa 1 secondo) e bassa varianza dipesa dagli accessi a risorse e memoria nonché dallo scheduling dei processi della CPU.

Infine la soluzione con il *PMS* richiederà oltre 6 secondi di calcolo dovuti soprattutto ai tempi di attesa e trasmissione della rete, l'alta varianza simboleggia la possibilità di congestione di quest'ultima, eventualità ovviamente impredicibile.

4.1.2 Performance di precisione

La precisione è un altro elemento fondamentale per applicazioni di questo genere, garantire all'utente la sicurezza che ciò che sta guardando ed utilizzando rispecchi la realtà è stato uno degli obiettivi principale del progetto di tesi.

I grafici successivi presentano estratti temporali di posizioni di Terra e Luna presi come campioni per pianeti e satelliti.

Tutti i dati riportati vanno letti ed interpretati tenendo a mente le dimensioni e la velocità con cui i copri celesti si muovono all'interno dello spazio.

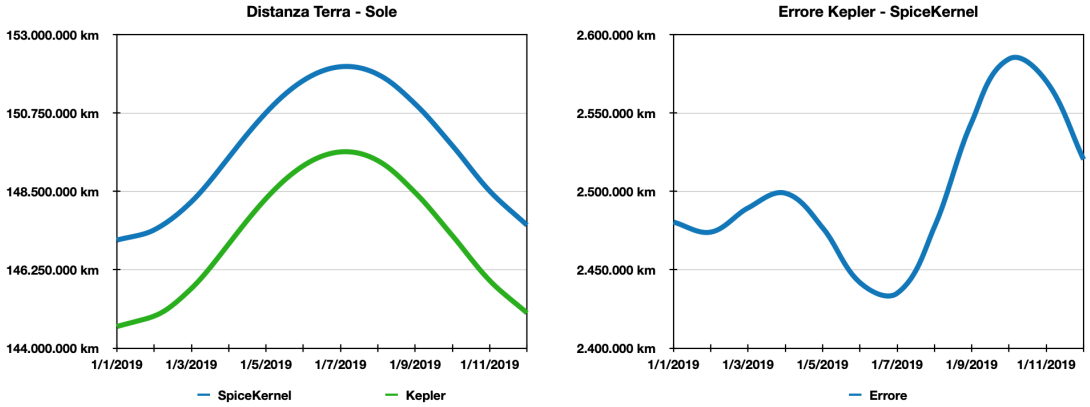


Figura 4.3: Distanza Terra - Sole calcolata mensilmente

Precisione del modello di *Keplero*

Si assume la soluzione basata sugli *SPICE* precisa per definizione. Come si può osservare (Figura 4.3) il modello di *Keplero* è in grado di replicare abbastanza fedelmente quello che dovrebbe essere il corretto andamento temporale di un pianeta, la distanza tra le due curve indica l'errore intrinseco che tale soluzione ha, questo è dato dal fatto che il modello semplificato formulato da *Keplero* ed utilizzato in ADN non tiene conto di eventuali contributi secondari e terziari di attrazione gravitazionale che i corpi celesti ricevono. Si può comunque notare che l'errore non cresce mai oltre i 2.500.000 km, compromesso accettabile per tale soluzione.

Comportamento simile per i satelliti (Figura 4.4), compiendo orbite molto più strette il modello di *Keplero* è in grado di produrre risultati più precisi. In tali casi l'errore non supera mai i 20.000 km.

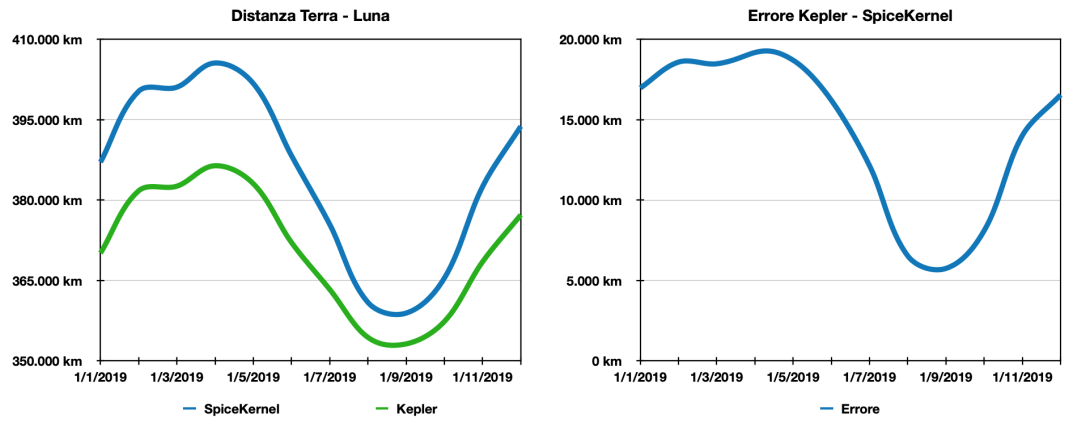


Figura 4.4: Distanza Terra - Luna calcolata mensilmente

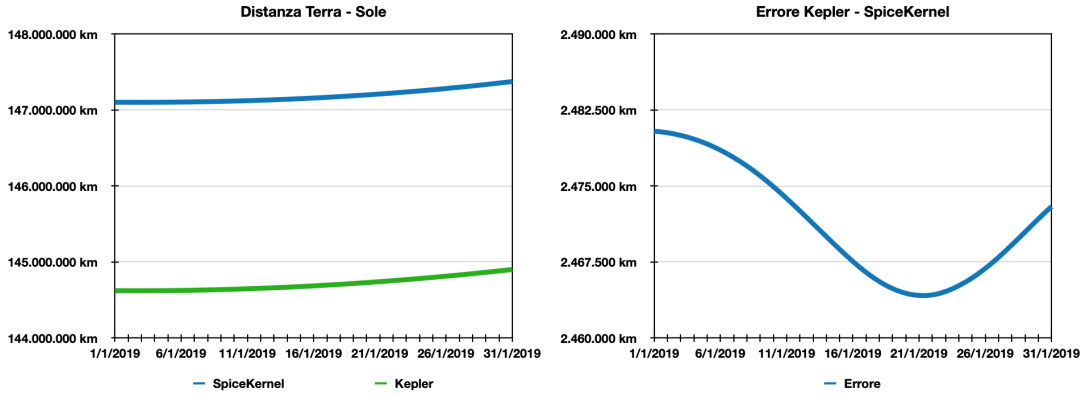


Figura 4.5: Distanza Terra - Sole calcolata giornalmente

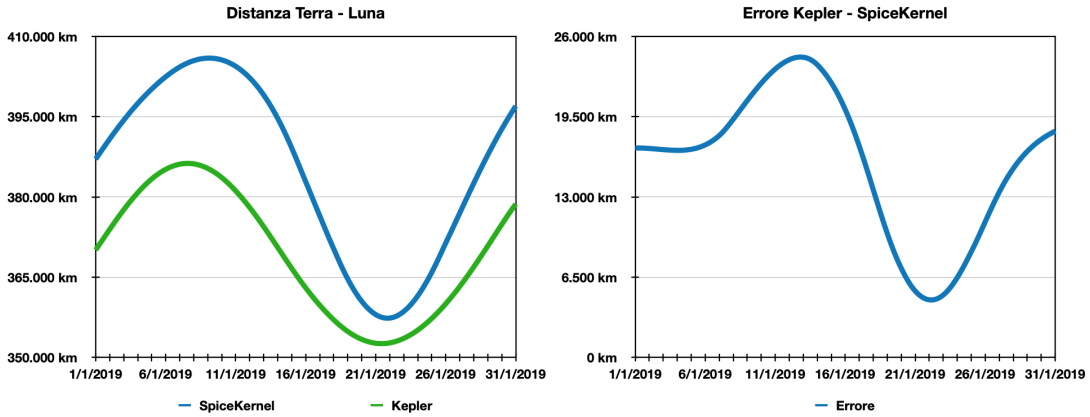


Figura 4.6: Distanza Terra - Luna calcolata giornalmente

Ripetendo l'analisi e calcolando una posizione al giorno per un mese (Figure 4.5 e 4.6) anziché una al mese per un anno si ottengono risultati molto simili. Ciò avviene poiché il comportamento dei corpi celesti risulta essere uniforme nel tempo.

Occorre notare che l'errore prodotto non è mai costante nel tempo, rendendo impossibile applicare una correzione manuale. Sarebbe necessario raffinare il modello matematico utilizzato al fine di ottenere una maggior precisione senza ridurre le prestazioni garantite da tale soluzione.

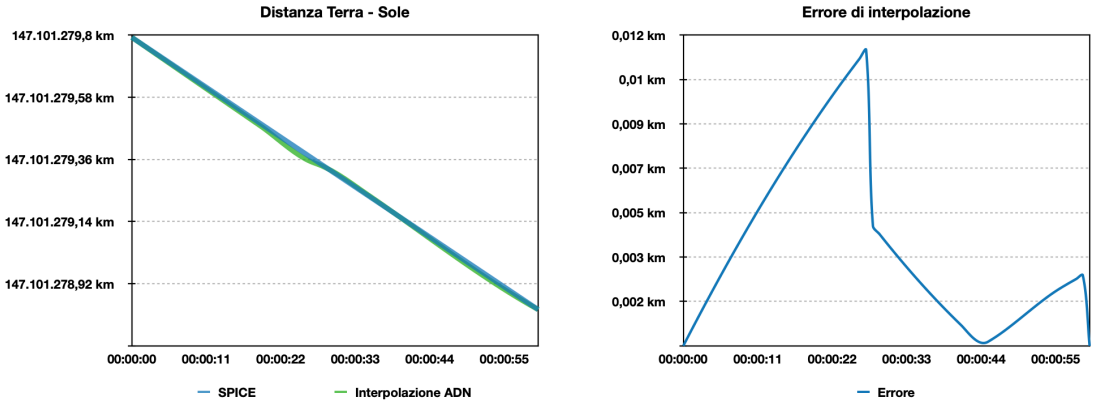


Figura 4.7: Errore di interpolazione della Terra

Precisione della soluzione con *SPICE*

In tal caso non avrebbe alcun senso effettuare un analisi di confronto sul lungo periodo in quanto non si noterebbe mai errore. Piuttosto è stata condotto un'analisi sull'interpolazione che *ADN* compie tra due posizioni calcolate con gli *SPICE*. Come spiegato nei capitoli precedenti, viene calcolata una posizione ogni minuto per poi interpolarne i valori da utilizzare tra esse.

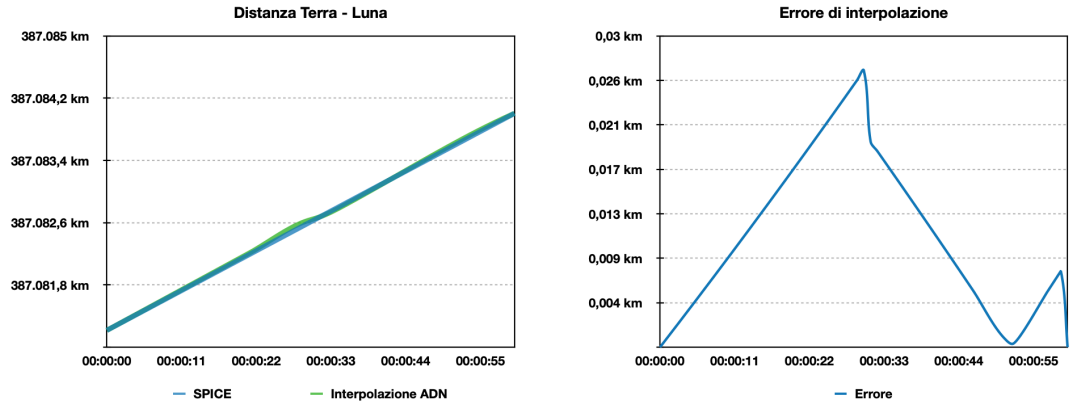


Figura 4.8: Errore di interpolazione della Luna

I risultati ottenuti, come si nota dai grafici in figura 4.7 e 4.8, descrivono come *ADN* non compia mai errori di interpolazione maggiori di circa 12 metri con una media di errore di 3,8 metri per la Terra, e 28 metri con media di 10,7 metri per la Luna. Si nota anche dal fatto che i segmenti riportati nei primi grafici coincidano quasi perfettamente.

4.2 Problemi

Nel corso dello sviluppo di *ADN* si sono incontrate diverse criticità, alcune delle quali non ancora risolte. La lista seguente contiene quelle più rilevanti ancora presenti all'interno di *ADN*:

- **Orbite *SPICE***: allo stato attuale, come spiegato nel paragrafo 3.4.6, risulta difficile calcolare orbite complete per tutti i corpi celesti a causa della mancanza di alcuni kernel. Sarebbe tuttavia possibile attuare una logica in grado di calcolare segmenti di orbita indipendenti al fine di interrompersi in caso di mancanza di dati. Tale soluzione permetterebbe all'utente di osservare le orbite fintanto che vi è supporto da parte dei kernel.

  Per risolvere completamente questo problema occorrerebbe modificare il metodo con cui le orbite vengono disegnate. Una delle possibili soluzioni analizzate sarebbe quella di utilizzare uno *shader*, in grado di velocizzare il ricalcolo dei vertici, ed essere inserito all'interno delle pipeline di rendering in modo da non gravare eccessivamente sul tempo totale con cui viene prodotto un frame. Tale modifica permetterebbe di raffinare ulteriormente i punti con cui vengono calcolate le orbite evitando il più possibile spiacevoli linee spezzate.

- **Limitatezza dei corpi celesti**: grazie all'utilizzo dei kernel sarebbe possibile calcolare le geometrie spaziali di una vastissima quantità di oggetti. Allo stato attuale per visualizzare un oggetto all'interno di *ADN* occorre crearne il suo *Prefab* in cui vengono gestite eventuali texture.

  Per risolvere un simile problema si potrebbe modellare un *Prefab* generico da utilizzare in tutti quei casi in cui non dovesse essere disponibile una sua versione specifica. Ovviamente occorrerebbe tenere conto che eventuali nuove orbite rischierebbero di gravare eccessivamente sulla CPU causando drastici cali di FPS.

- **cspice.dll non parallelizzabile**: fintanto che non verrà prodotta una versione della dll parallelizzabile, risulta difficile pensare di ampliare i cataloghi presenti poiché ciò causerebbe tempi di calcolo troppo elevati per essere effettuati sequenzialmente.

- **Effetto stereoscopico**: il tipo di scena renderizzata da *ADN* non presenta le caratteristiche ideali al fine di ottenere un buon effetto stereo 3D. Ciò è dovuto principalmente da due fattori:

1. Non esiste un vero e proprio background se non lo spazio profondo popolato unicamente da minuscoli puntini raffiguranti le stelle, ciò non fornisce all'utente sufficienti suggerimenti sulla profondità non permettendogli di creare la percezione del 3D.
2. Allo stesso modo non esiste un foreground poiché gli oggetti in "primo piano" che occupano la scena saranno sempre a migliaia di kilometri dalla camera. Per questo motivo verranno comunque considerati da *Unity* come parte del background e appariranno entranti nello schermo anziché uscenti come ci si aspetterebbe dall'effetto 3D.

Tale problema è legato alla natura del contesto e quindi non di facile risoluzione. L'unico elemento che beneficia correttamente dell'effetto tridimensionale è l'interfaccia utente, la quale galleggia di fronte alla scena.

4.3 Test di valutazione soggettiva

Dopo aver terminato lo sviluppo, è stato chiesto ad un gruppo di 12 persone di validare la nuova versione di *Astra Data Navigator* al fine di ottenere una valutazione soggettiva e suggerimenti per possibili futuri miglioramenti e sviluppi.

Il gruppo era composto di 7 soggetti maschi e 5 femmine di età compresa tra 24 e 59 anni, selezionati in modo da avere un'ampia varietà di competenze e capacità professionali.

Ai soggetti è stato presentato un sondaggio (Appendice A.4) il quale li ha guidati nel test dell'applicazione, tale sondaggio è stato progettato al fine di misurare l'usabilità complessiva del sistema, l'intuitività dell'interfaccia utente ed il realismo della simulazione.

Il questionario è principalmente composto di tre parti: la prima consiste nel completamento di informazioni generali riguardanti i soggetti del test quali età, sesso e precedenti esperienze VR; la seconda contiene 3 task che l'utente dovrà eseguire, seguiti da alcune domande prese dal *NASA TLX* tool [20] scelte al fine di valutare il carico di lavoro percepito ed eventuale senso di nausea causato dall'applicazione.

Il primo compito consiste nel lanciare *ADN* in configurazione di default per poi navigare all'interno dell'universo virtuale utilizzando il pannello di ricerca, i selettori attivabili oppure il movimento libero.

Nel secondo test viene richiesto all'utente di alterare la velocità della simulazione ed attivare le orbite attraverso l'apposito bottone al fine di permettergli di osservare evoluzione temporale dei diversi corpi celesti presenti nella scena.

Nel terzo ed ultimo l'utente dovrà chiudere l'applicazione per modificare il file `config.xml` per cambiare il metodo di rivoluzione da *Kepler* a *SpiceKernel* (come spiegato nei capitoli precedenti) tale metodo richiede la configurazione di un *metakernel*, che per i test è stato posto all'interno della cartella dell'applicazione, copiarne ed inserirne il path all'interno dell'apposito campo. Terminato ciò viene

chiesto di riavviare l'applicazione ed eseguire azioni simili a quelle fatte nei primi due compiti. Questa terza richiesta è stata fatta al fine di valutare se la configurazione basata su di un file XML fosse sufficientemente intuitiva per gli utenti, ed anche per far osservare la differenza di prestazioni e precisione delle due differenti soluzioni utilizzate.

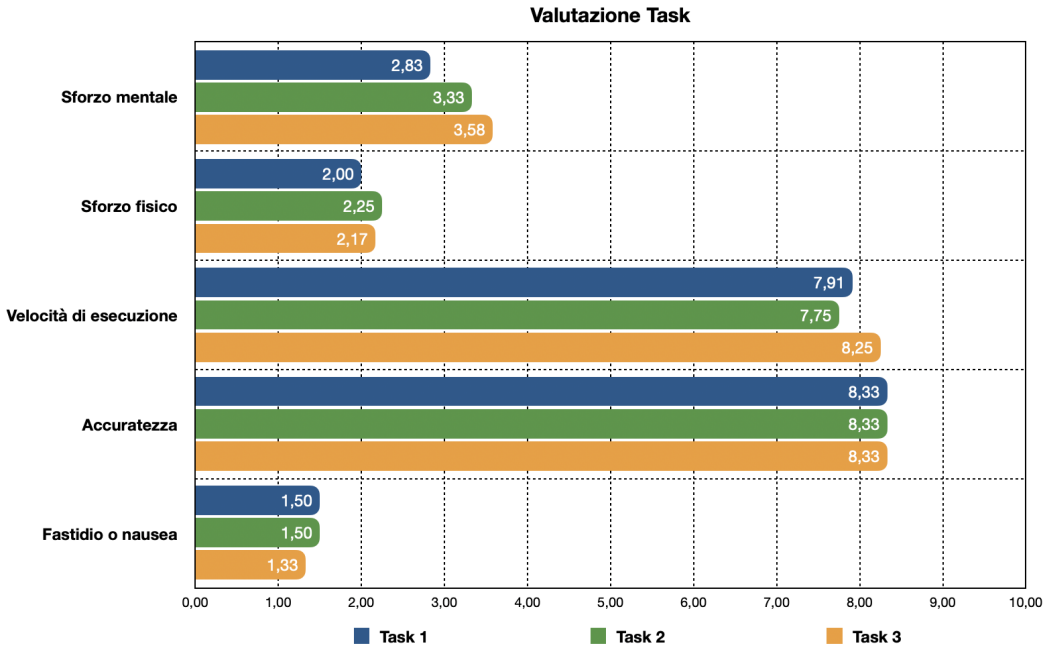


Figura 4.9: Risultati di valutazione dei task

Dopo aver completato ogni incarico venivano poste 5 domande: sforzo mentale, sforzo fisico, velocità di esecuzione, accuratezza e fastidio o nausea percepiti. Ogni riposta era graduata da una scala che partiva da 1 per finire a 10.

Il grafico 4.9 riporta la media delle risposte degli utenti. Come si può notare si sono ottenuti ottimi risultati in ogni domanda, soprattutto non è mai stato percepito alcun fastidio o senso di nausea, molto comune per questo genere di applicazioni stereoscopiche, ciò probabilmente è dovuto al leggerissimo effetto stereo.

Come atteso si può osservare un aumento dello sforzo mentale percepito per l'ultimo compito legittimato dalla non familiarità con configurazioni tramite file XML da parte degli utenti meno esperti.

Nella terza ed ultima sezione del test di valutazione venivano poste all'utente domande di carattere generale da completare successivamente all'utilizzo dell'applicazione su: usabilità complessiva del sistema, realismo della rappresentazione, utilità delle orbite, utilità delle informazioni sui corpi celesti, responsività dell'alterazione temporale, tolleranza dell'imprecisione, utilità dell'applicazione e possibile frequenza di utilizzo. Veniva inoltre data la possibilità di scrivere eventuali suggerimenti su aggiornamenti futuri o consigli per migliorare le caratteristiche attualmente

presenti.

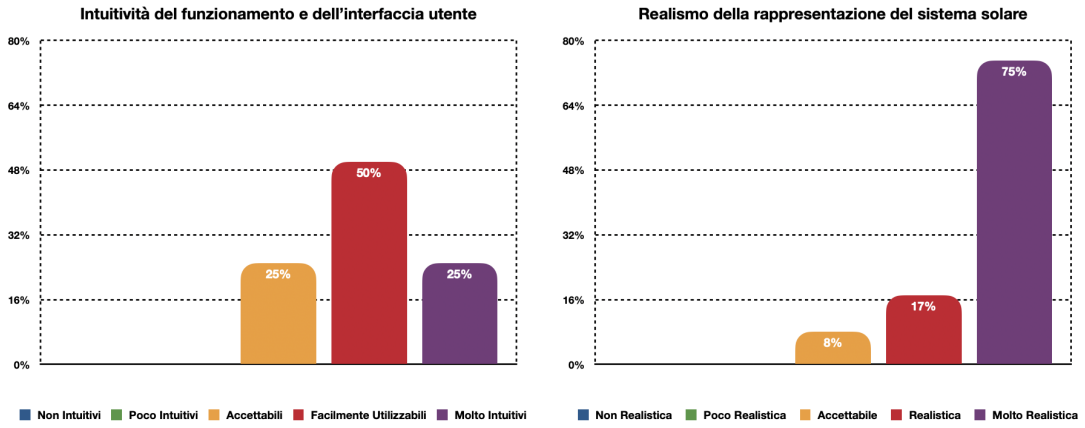


Figura 4.10: Intuitività e realismo dell'applicazione

Il grafico 4.10 illustra i risultati delle risposte degli utenti ad intuitività dell'interfaccia grafica e realismo dell'applicazione. La risposta è stata per lo più positiva, la metà degli utenti considera la UI "Facilmente Utilizzabile" ed il 75% descrive l'applicazione come "Molto Realistica". Ciò comunque non esclude la possibilità che *ADN* possa essere migliorato. La maggior parte dei suggerimenti ricevuti riguardano la UI per rendere controlli e bottoni più grandi e facilmente raggiungibili (2 suggerimenti), semplificare i sistema di configurazione (1 suggerimento) e realizzare un nuovo pannello contente le informazioni riguardo a corpi celesti precedentemente visitati per potervi tornare più facilmente (1 suggerimento).

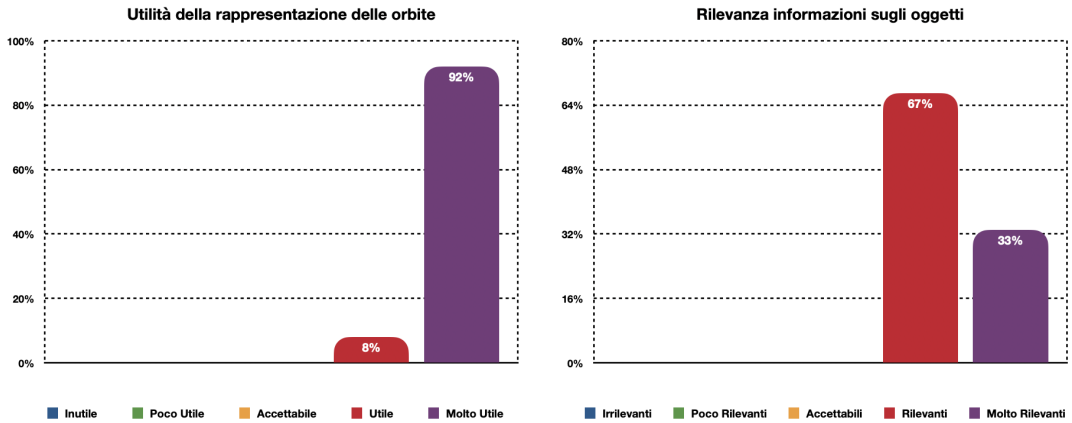


Figura 4.11: Utilità delle orbite e rilevanza delle informazioni fornite

Come si può notare dal grafico 4.11 e come sarebbe corretto aspettarsi la rappresentazione delle orbite permette una comprensione di carattere più generale del movimento dei corpi celesti nonché un utile punto di riferimento spaziale di

questi ultimi. Inoltre, generalmente gli utilizzatori nutrono maggiore interesse se ricevono più informazioni scientifiche possibili rispetto a ciò che stanno osservando.

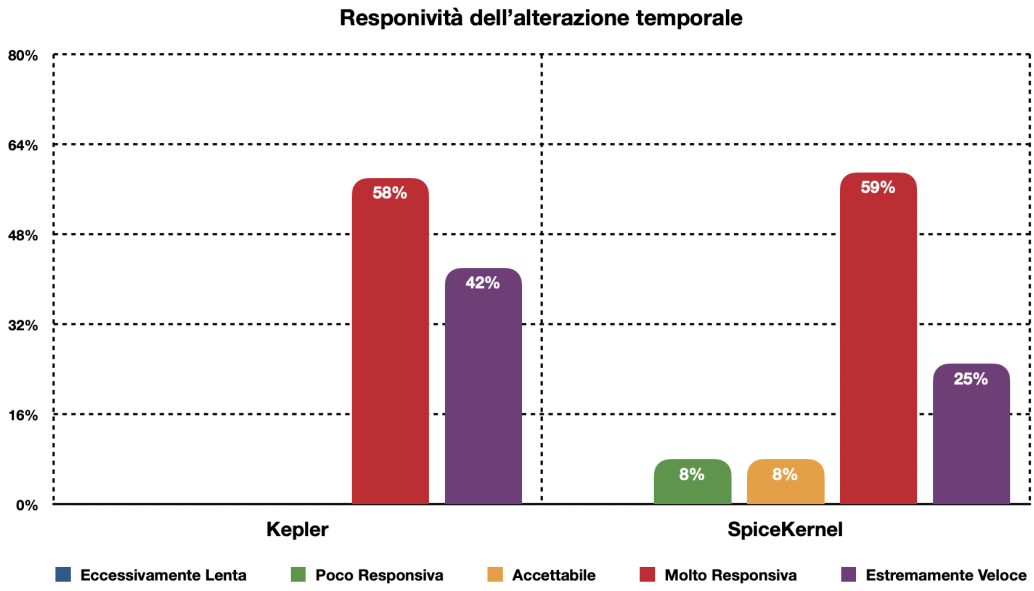


Figura 4.12: Responsività dell'alterazione temporale

Le analisi svolte precedentemente sulla velocità di calcolo, nel grafico 4.12, trovano un confronto rispetto alla percezione di tale velocità da parte degli utenti. Come si osserva dai valori: la totalità degli utenti trova l'alterazione temporale effettuata mediante il metodo di *Keplero* "Molto Responsiva" se non "Estremamente Responsiva"; per ciò che riguarda l'utilizzo dei kernel sono presenti alcuni utenti che preferirebbero un miglioramento (16%), mentre il 59% accetta di attendere brevemente al fine di osservare una simulazione la cui chiave è la precisione.

Dato fortemente soggettivo riguarda la tolleranza dell'imprecisione delle posizioni nel corso della simulazione. Normalmente gli utenti saranno più propensi a tollerare maggiore imprecisione utilizzando l'applicazione con le equazioni di *Keplero* piuttosto che con gli *SPICE*. Il grado di tolleranza, come si osserva nel grafico 4.13, nel primo caso tende ad essere maggiore fino a raggiungere valori di oltre 1.000.000 di *km*, mentre nel secondo è più propenso verso la precisione al di sotto del *km*. Al fine di svolgere i test il più correttamente possibile, non sono stati dati suggerimenti agli utenti riguardo all'effettivo grado di precisione delle diverse soluzioni.

Come ogni applicazione svolta in ambito scientifico, anche per *ADN*, è stato chiesto un riscontro soggettivo di utilità in diversi settori (Grafico 4.14). Il 75% riconosce in *ADN* uno strumento "Estremamente Utile" nel campo dell'educazione, poter visualizzare graficamente le nozioni permetterebbe una più facile comprensione di queste ultime.

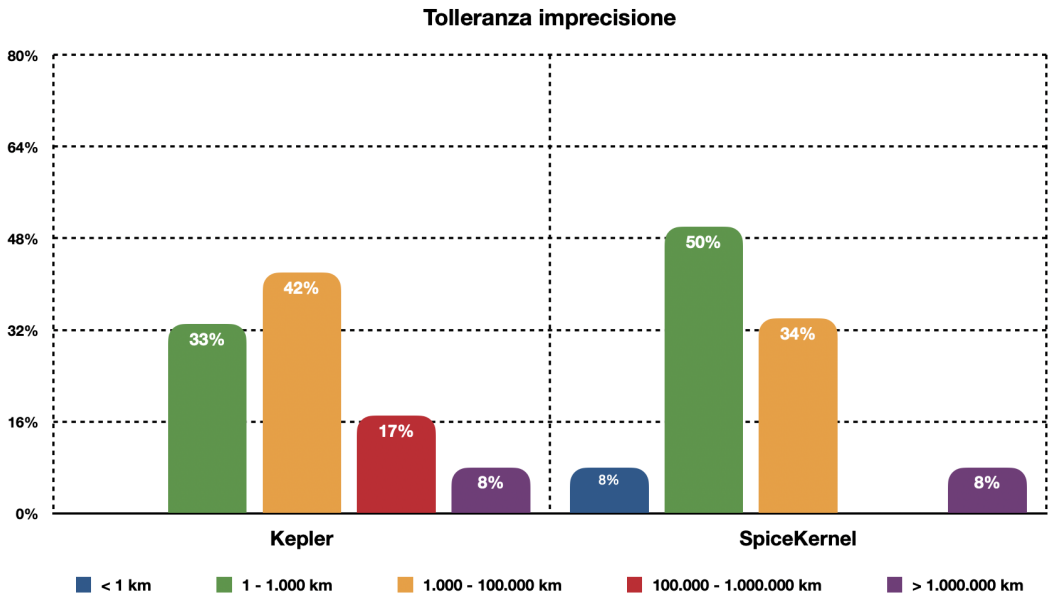


Figura 4.13: Tolleranza di imprecisione delle posizioni

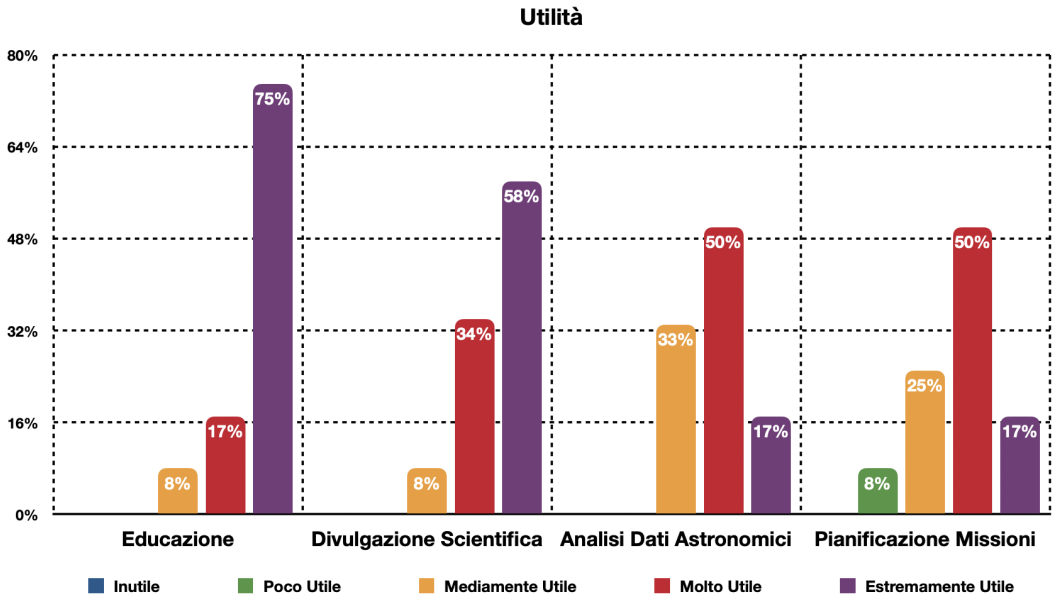


Figura 4.14: Possibili ambiti di utilizzo di *ADN*

La divulgazione scientifica consiste nel fornire ad una vasta scala di utenti la possibilità di esplorare liberamente l'universo al fine di osservare personalmente l'effettivo funzionamento delle dinamiche di interazione dei diversi corpi celesti. In questo caso viene ritenuto *ADN* uno strumento valido per adempiere a tale compito. L'analisi dei dati astronomici viene attualmente effettuata unicamente attraverso

tabelle prive di supporto grafico. Poter visualizzare esplicitamente grandi moli di dati permetterebbe a scienziati ed ingegneri una maggiore facilità di movimento all'interno di essi. Il 92% degli utenti giudica *ADN* più che "Utile" se utilizzato per tale scopo. Infine, la pianificazione missioni risulterebbe attualmente più complessa se non per farsi un'idea di carattere generale delle posizioni degli oggetti nel tempo. Nonostante la grande precisione, per questo tipo di utilizzo non è tollerabile il minimo errore. Se migliorato, *ADN*, potrebbe divenire utile e di supporto anche in questo ambito.

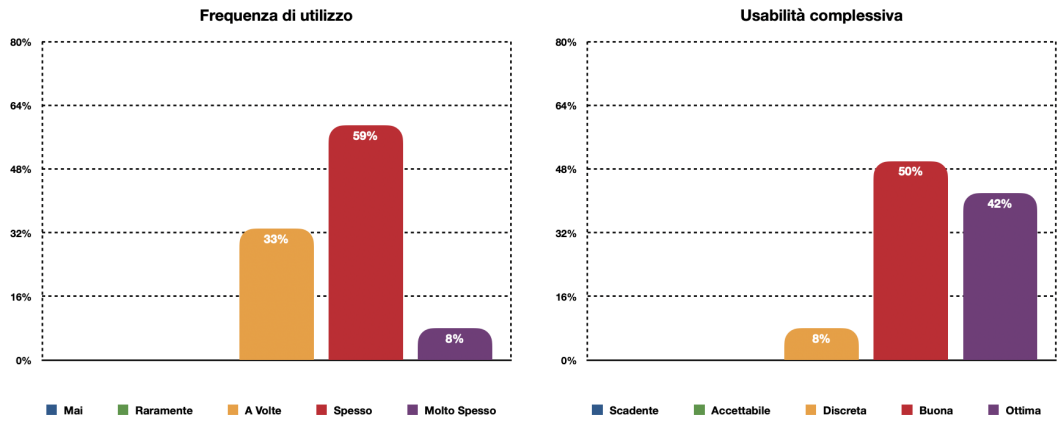


Figura 4.15: Frequenza di utilizzo ed usabilità dell'applicazione

Come ultime domande, è stato chiesto agli utenti, di immaginare la possibile frequenza di utilizzo di *ADN* ed una valutazione di usabilità complessiva. Il 59% dichiara che lo utilizzerebbe "Spesso", mentre il 33% solamente "A Volte". Molto positive le risposte sull'usabilità complessiva, il 50% la ritiene "Buona" ed il 42% "Ottima".

Capitolo 5

Conclusioni

Il progetto di tesi presentato ha portato con successo alla realizzazione di una simulazione in tempo reale nella quale è possibile osservare con buona precisione l'evoluzione temporale di vari corpi celesti del sistema solare. Gli aggiornamenti attuati in *Astra Data Navigator* hanno portato ad un notevole miglioramento della simulazione, offrendo un'esperienza in VR più completa ed accurata. Nella sua versione attuale, *ADN* è in grado di competere con molte delle applicazioni capaci di utilizzare gli *SPICE* come strumento di calcolo di precisione. La facilità con cui è possibile espandere i cataloghi attualmente presenti è uno degli aspetti più rilevanti di questo progetto.

I test soggettivi effettuati sulla versione finale sviluppata hanno evidenziato la cura con cui è stato gestito il realismo nella rappresentazione di un concetto complesso come l'universo, nonché l'intuitività e le prestazioni ottenute. Nonostante ciò sono emerse alcune carenze che in futuro potrebbero essere migliorate al fine di produrre un risultato finale eccellente.

I suggerimenti ricevuti vertono per lo più sulla UI, la quale beneficerebbe di una rielaborazione. Come accennato in precedenza si è scelto di mantenere coerenza con le precedenti versioni di *ADN*, viste le notevoli aggiunte fatte, tale decisione potrebbe non aver giovato all'applicazione.

Dei futuri possibili miglioramenti dell'interfaccia utente, e non solo, potrebbero essere:

- Realizzare un nuovo pannello in grado di mostrare la recente cronologia di navigazione permettendo all'utente di ritornare sui propri passi in maniera immediata e priva di sforzi, per esempio facendo doppio click sul nome di un oggetto precedentemente visitato.

- Introdurre dei tooltip utili all'utente per comprendere lo scopo di ogni elemento visibile della UI.

- Ridisegnare la logica di configurazione, portandola ad essere una scena a se stante che l'utente ha la possibilità di modificare prima che l'applicazione

parta ed a cui è possibile tornare in ogni momento. Renderla un'interfaccia grafica permetterebbe un più facile utilizzo anche ad utilizzatori poco esperti.

- Rendere la disposizione della UI completamente personalizzabile dall'utente, permettendogli di spostare separatamente i pannelli ovunque egli voglia.
- Integrare nuovi sistemi di input quali mouse 3D o motion capture in grado di provvedere ad un più naturale movimento all'interno dell'universo virtuale.

La realizzazione delle orbite calcolate con il supporto degli *SPICE* incrementerebbe notevolmente la valenza scientifica di *ADN* poiché si sarebbe in grado di osservare fisicamente la corretta evoluzione temporale che i diversi corpi celesti hanno. Nonostante ciò occorrerebbe prestare molta attenzione alla tecnica utilizzata per disegnare le orbite in modo da non gravare eccessivamente sul tempo di rendering.

Poter calcolare parallelamente le posizioni degli oggetti presenti nei cataloghi permetterebbe di accrescerne notevolmente il numero diminuendo addirittura il tempo di calcolo. Tale modifica può essere considerata tra le più importati sul fronte dell'espandibilità e versatilità della simulazione. Comporterebbe inoltre, con qualche modifica aggiuntiva, la possibilità per l'utente di caricare cataloghi di corpi celesti personali e costruiti ad hoc basati sulle proprie necessità.

Nonostante il realismo attuale dell'applicazione sia considerato ottimo, vi è sempre margine di miglioramento aumentando la varietà di oggetti presenti nella scena, introducendo asteroidi (di cui per alcuni è possibile calcolare le posizioni esatte attraverso gli *SPICE*), nebulose e galassie.

Con lo studio effettuato viene evidenziata la grande potenzialità che la Realtà Virtuale ha per supportare la ricerca scientifica nel campo dell'astronomia, fornendo un prezioso strumento per scopi educativi e divulgativi in grado di mostrare la corretta evoluzione temporale dei corpi celesti. Migliorare le carenze attualmente riscontrate richiederebbe risorse e tempo aggiuntivo, nonostante ciò i risultati ottenuti si sono dimostrati molto convincenti delineando la grande potenzialità di questa applicazione.

Appendici

A.1 AdnSpiceKernelRevolution - GetPosList

```
1     public unsafe List<Position> GetPosList(int naif_id, DateTime start_time,
      DateTime end_time, int step)
2         {
3             // Mutex necessary because the dll is not multi-thread
4             mutex.WaitOne();
5             double et1, et2, lt = 0;
6             double[] ptarg = new double[3];
7             List<Position> pos = new List<Position>();
8             str2et_c(new StringBuilder(start_time.ToString()), &et1);
9             str2et_c(new StringBuilder(end_time.ToString()), &et2);
10            for (double et = et1; et <= et2; et += step)
11            {
12                // Divide the time span in "step" calls
13                spkpos_c(new StringBuilder(naif_id.ToString()), et, new
      StringBuilder(REVOLUTION_SETTINGS.FRAME), new StringBuilder(
      REVOLUTION_SETTINGS.ABCORR), new StringBuilder(REVOLUTION_SETTINGS.OBSERVER),
      ptarg, &lt);
14                if (ptarg.All(value => value == 0))
15                {
16                    // If something went wrong is necessary to unload and then
      reload Kernels
17                    unload_c(new StringBuilder(AdnSettingsManager.RevolutionSource
      ));
18                    furnsh_c(new StringBuilder(AdnSettingsManager.RevolutionSource
      ));
19                    mutex.ReleaseMutex();
20                    throw new KernelsException();
21                }
22                pos.Add(new Position(ptarg));
23                Array.Clear(ptarg, 0, 3);
24            }
25            mutex.ReleaseMutex();
26            if (pos.Count == 0)
27                throw new KernelsException();
28            return pos;
29        }
```

A.2 AdnPMSRevolution - GetPosList

```
    public List<Position> GetPosList(int naif_id, DateTime start_time, DateTime
    end_time, int step)
        {
            Dictionary<string, string> form = new Dictionary<string, string>
            {
                { "start_time", start_time.ToString() },
                { "end_time", end_time.ToString() },
                { "target", naif_id.ToString() },
                { "observer", REVOLUTION_SETTINGS.OBSERVER },
                { "step", step.ToString() }
            };

            AdnWebRequest wb = new AdnWebRequest(URL, "spkpos");
            wb.IgnoreCertificateValidation();
            wb.SetRequestType(AdnWebRequest.RequestType.POST);
            wb.SetContentType(AdnWebRequest.ContentType.json);
            wb.SetRequestData(JsonSerializer.Serialize(form));

            string json = Encoding.UTF8.GetString(wb.GetResponse());
            if (json == null) throw new WebException();
            List<Position> pos = JsonSerializer.DeserializePositions(json);
            return pos;
        }
```

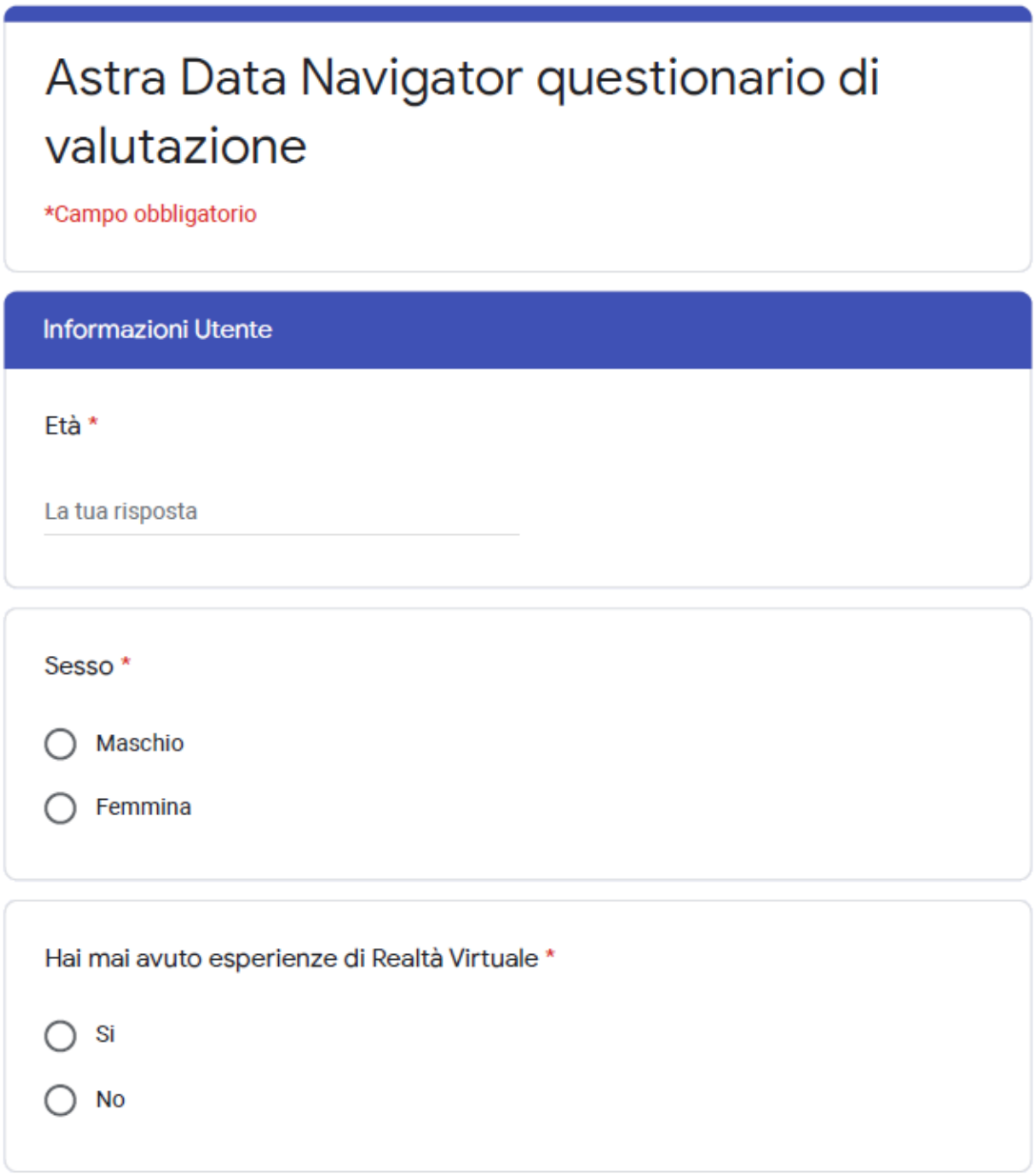
A.3 AdnCatalogueGenerator

```
    FRAME = 'J2000'
    ET = '2019 JAN 01 00:00:00.000'
    ABCORR = 'NONE'
    OBS = 'SUN'

    try:
        spice.furnsh(sys.argv[1])
        if spice.ktotal('ALL') == 0:
            print('Error: 0 Kernels loaded')
    except SpiceNOSUCHFILE:
        print('Error: SpiceNOSUCHFILE.')
    try:
        infile = open(sys.argv[2], "r")
        outfile = open('./Catalogues/' + sys.argv[2], 'w')
        lines = infile.readlines()
        if len(lines[0].split('\t')) == 3:
            outfile.write('Name\tCategory\tParent\tNAIF_ID\tSCALE\tX\tY\tZ\tPHI\
    tTHETA\tPSI\tAV_MAGNITUDE\tPERIOD\tRP\tECC\tINC\tLNODE\tARGP\tNU\tGM')
        if len(lines[0].split('\t')) == 2:
            outfile.write('Name\tParent\tNAIF_ID\tSCALE\tX\tY\tZ\tPHI\tTHETA\tPSI\
    tAV_MAGNITUDE\tPERIOD\tRP\tECC\tINC\tLNODE\tARGP\tNU\tGM')
        for line in lines:
            celestialObject = line.split('\t')
            outfile.write('\n' + celestialObject[0] + '\t')
            if len(celestialObject) == 3:
                outfile.write(celestialObject[2].strip() + '\t')
            outfile.write(celestialObject[1].strip() + '\t')
            try:
                NAIF_ID = spice.bodn2c(celestialObject[0])
            except NotFoundError:
```

```
29                print('ATTENTION: Kernels can\'t retrieve NAIF_ID for ' +
    celestialObject[0].upper() + ', parameter must be completed manually.')
30                NAIF_ID = int(input('NAIF_ID: '))
31            outfile.write('{}\t'.format(NAIF_ID))
32            try:
33                Scale = spice.bodvrd(celestialObject[0], 'RADII', 3)[1][0]
34            except (SpiceNOTRANSLATION, SpiceKERNELVARNOTFOUND):
35                print('ATTENTION: Kernels can\'t retrieve Scale for ' +
    celestialObject[0].upper() + ', parameter must be completed manually.')
36                Scale = float(input('Scale: '))
37            outfile.write('{}\t'.format(Scale * 2))
38            try:
39                Pos = spice.spkpos(celestialObject[0], spice.str2et(ET), FRAME,
    ABCORR, OBS)
40            except SpiceIDCODENOTFOUND:
41                print('ATTENTION: Kernels can\'t retrieve Position for ' +
    celestialObject[0].upper() + ', parameter must be completed manually.')
42                Pos = [[0, 0, 0]]
43                Pos[0][0] = float(input('X: '))
44                Pos[0][1] = float(input('Y: '))
45                Pos[0][2] = float(input('Z: '))
46            outfile.write('{}\t{}\t{}\t'.format(Pos[0][0], Pos[0][1], Pos[0][2]))
47            try:
48                xform = spice.sxform("IAU_" + celestialObject[0].upper(), FRAME,
    spice.str2et(ET))
49                rotation_matrix, angular_velocity = spice.xf2rav(xform)
50                psi, theta, phi = spice.m2eul(rotation_matrix, 3, 1, 3)
51                outfile.write('{0:.4f}\t{1:.4f}\t{2:.4f}\t{3:.8f}\t'.format(np.
    rad2deg(phi), np.rad2deg(theta), np.rad2deg(psi), np.rad2deg(np.linalg.norm(
    angular_velocity))))
52            except SpiceUNKNOWNFRAME:
53                print('ATTENTION: Kernels can\'t retrieve PHI, THETA, PSI for ' +
    celestialObject[0].upper() + ', all values are set to 0.')
54                print('ATTENTION: Kernels can\'t retrieve AV_MAGNITUDE (rotation
    speed deg/s) for ' + celestialObject[0].upper() + ', parameter must be
    completed manually.')
55                angular_velocity = input('AV_MAGNITUDE: ')
56                outfile.write('{0:.4f}\t{1:.4f}\t{2:.4f}\t{3:.8f}\t'.format(0, 0,
    0, float(angular_velocity)))
57            try:
58                GM = spice.bodvrd(celestialObject[1].strip(), 'GM', 1)[1][0]
59                STARG, _ = spice.spkezr(celestialObject[0], spice.str2et(ET),
    FRAME, ABCORR, celestialObject[1].strip())
60                ELTS = spice.oscltx(STARG, spice.str2et(ET), GM)
61                outfile.write('{0:.4f}\t{1:.4f}\t{2:.8f}\t{3:.6f}\t{4:.6f}\t{5:.6f
    }\t{6:.6f}\t{7:.4f}'.format(ELTS[10], ELTS[0], ELTS[1], np.rad2deg(ELTS[2]),
    np.rad2deg(ELTS[3]), np.rad2deg(ELTS[4]), np.rad2deg(ELTS[8]), ELTS[7]))
62            except (SpiceIDCODENOTFOUND, SpiceKERNELVARNOTFOUND):
63                print('ATTENTION: Kernels can\'t retrieve Orbital Parameters for '
     + celestialObject[0].upper() + ', all values are set to 0.')
64                outfile.write('{0:.4f}\t{1:.4f}\t{2:.8f}\t{3:.6f}\t{4:.6f}\t{5:.6f
    }\t{6:.6f}\t{7:.4f}'.format(0, 0, 0, 0, 0, 0, 0, 0))
```

A.4 Questionario di valutazione soggettiva



Astra Data Navigator questionario di valutazione

*Campo obbligatorio

Informazioni Utente

Età *

La tua risposta

Sesso *

- Maschio
- Femmina

Hai mai avuto esperienze di Realtà Virtuale *

- Si
- No

Task 1

Dopo aver lanciato l'applicazione, viaggiare verso un pianeta del Sistema Solare utilizzando il pannello di ricerca o attraverso i selettori attivabili dall'interfaccia. Esplorare i dintorni del pianeta e dei suoi satelliti.

Quanto sforzo mentale ha richiesto questa operazione? *

	1	2	3	4	5	6	7	8	9	10	
Veramente Poco	○	○	○	○	○	○	○	○	○	○	Veramente Tanto

Quanto sforzo fisico ha richiesto questa operazione? *

	1	2	3	4	5	6	7	8	9	10	
Veramente Poco	○	○	○	○	○	○	○	○	○	○	Veramente Tanto

Quanto velocemente hai eseguito questo task? *

	1	2	3	4	5	6	7	8	9	10	
Molto Lentamente	○	○	○	○	○	○	○	○	○	○	Molto Velocemente

Quanto accuratamente sei riuscito/a a completare questo task? *

	1	2	3	4	5	6	7	8	9	10	
Per Nulla	○	○	○	○	○	○	○	○	○	○	Completamente

Quanto fastidio o senso di nausea hai provato? *

	1	2	3	4	5	6	7	8	9	10	
Nessuno	○	○	○	○	○	○	○	○	○	○	Molto

Task 2

Utilizzando l'interfaccia grafica, attivare le orbite ed alterare la velocità della simulazione al fine di osservare il comportamento temporale dei corpi celesti.

Quanto sforzo mentale ha richiesto questa operazione? *

	1	2	3	4	5	6	7	8	9	10	
Veramente Poco	○	○	○	○	○	○	○	○	○	○	Veramente Tanto

Quanto sforzo fisico ha richiesto questa operazione? *

	1	2	3	4	5	6	7	8	9	10	
Veramente Poco	○	○	○	○	○	○	○	○	○	○	Veramente Tanto

Quanto velocemente hai eseguito questo task? *

	1	2	3	4	5	6	7	8	9	10	
Molto Lentamente	○	○	○	○	○	○	○	○	○	○	Molto Velocemente

Quanto accuratamente sei riuscito/a a completare questo task? *

	1	2	3	4	5	6	7	8	9	10	
Per Nulla	○	○	○	○	○	○	○	○	○	○	Completamente

Quanto fastidio o senso di nausea hai provato? *

	1	2	3	4	5	6	7	8	9	10	
Nessuno	○	○	○	○	○	○	○	○	○	○	Molto

Task 3

Chiudere Astra Data Navigator, modificare il file "config.xml" nella cartella dell'applicazione cambiando il metodo di rivoluzione in SpiceKernel e aggiungere il path del meta kernel da utilizzare (che si trova nella cartella dell'applicazione). Dopodiché tornare ad esplorare il Sistema Solare alterando anche lo scorrimento del tempo.

Quanto sforzo mentale ha richiesto questa operazione? *

	1	2	3	4	5	6	7	8	9	10	
Veramente Poco	○	○	○	○	○	○	○	○	○	○	Veramente Tanto

Quanto sforzo fisico ha richiesto questa operazione? *

	1	2	3	4	5	6	7	8	9	10	
Veramente Poco	○	○	○	○	○	○	○	○	○	○	Veramente Tanto

Quanto velocemente hai eseguito questo task? *

	1	2	3	4	5	6	7	8	9	10	
Molto Lentamente	○	○	○	○	○	○	○	○	○	○	Molto Velocemente

Quanto accuratamente sei riuscito/a a completare questo task? *

	1	2	3	4	5	6	7	8	9	10	
Per Nulla	○	○	○	○	○	○	○	○	○	○	Completamente

Quanto fastidio o senso di nausea hai provato? *

	1	2	3	4	5	6	7	8	9	10	
Nessuno	○	○	○	○	○	○	○	○	○	○	Molto

Valutazione dell'usabilità del sistema

Quanto sono intuitivi il funzionamento e l'interfaccia dell'applicazione? *

	1	2	3	4	5	
Non Intuitivi	○	○	○	○	○	Molto Intuitivi

Quanto realistica hai trovato la rappresentazione degli oggetti del Sistema Solare? *

	1	2	3	4	5	
Non Realistica	○	○	○	○	○	Molto Realistica

Quanto hai trovato utile la rappresentazione delle orbite degli oggetti? *

	1	2	3	4	5	
Inutile	○	○	○	○	○	Molto Utile

Quanto hai trovato rilevanti le informazioni fornite riguardo agli oggetti? *

	1	2	3	4	5	
Irrilevanti	○	○	○	○	○	Molto Rilevanti

Quanto hai trovato responsiva l'alterazione della simulazione temporale? *

	Eccessivamente Lenta	Poco Responsiva	Accettabile	Molto Responsiva	Estremamente Veloce
Kepler	○	○	○	○	○
SpiceKernel	○	○	○	○	○

Quale imprecisione sei disposto/a a tollerare riguardo alle posizioni dei corpi celesti nel corso della simulazione? *

	< 1 km	1 - 1.000 km	1.000 - 100.000 km	100.000 - 1.000.000 km	> 1.000.000 km
Kepler	○	○	○	○	○
SpiceKernel	○	○	○	○	○

Quanto pensi che questa applicazione potrebbe essere utile nei seguenti ambiti? *

	Inutile	Poco Utile	Mediamente Utile	Molto Utile	Estremamente Utile
Educazione	○	○	○	○	○
Divulgazione Scientifica	○	○	○	○	○
Analisi Dati Astronomici	○	○	○	○	○
Pianificazione Missioni	○	○	○	○	○

Quanto frequentemente ti piacerebbe usare questo tool? *

	1	2	3	4	5	
Mai	○	○	○	○	○	Molto Spesso

Valutazione usabilità complessiva del sistema *

	1	2	3	4	5	
Scadente	○	○	○	○	○	Ottima

Suggerimenti

La tua risposta

Bibliografia

[1] Sandra Dutra Piovesan, Liliana Maria Passerino e Adriana Soares Pereira. «Virtual Reality as a Tool in the Education.» In: *International Association for Development of the Information Society* (2012). URL: https://academic.microsoft.com/paper/1839171376 (cit. a p. 1).

[2] Chris Christou. «Virtual Reality in Education». In: 2010, pp. 228–243. URL: https://academic.microsoft.com/paper/2477406261 (cit. a p. 1).

[3] YAGV Boas. «Overview of virtual reality technologies». In: *Interactive Multimedia Conference*. Vol. 2013. 2013. URL: https://static1.squarespace.com/static/537bd8c9e4b0c89881877356/t/5383bc16e4b0bc0d91a758a6/1401142294892/yavb1g12_25879847_finalpaper.pdf (cit. a p. 2).

[4] *Space making the virtual a reality*. ESA. URL: https://www.esa.int/Applications/Telecommunications_Integrated_Applications/Space_making_the_virtual_a_reality (cit. a p. 4).

[5] *The Virtual Interface Environment Workstation (VIEW)*. NASA. URL: https://www.nasa.gov/ames/spinoff/new_continent_of_ideas/ (cit. a p. 4).

[6] *NASA Explores Potential of Altered Realities for Space Engineering and Science*. NASA. URL: https://www.nasa.gov/feature/goddard/2017/nasa-explores-potential-of-altered-realities-for-space-engineering-and-science (cit. a p. 5).

[7] *ALTEC website*. URL: https://www.altecspace.it (cit. a p. 5).

[8] *NEANIAS Project website*. URL: https://www.neanias.eu (cit. a p. 5).

[9] *NAIF Website*. URL: https://naif.jpl.nasa.gov/naif/ (cit. a p. 8).

[10] *NASA WebGeocalc*. URL: https://wgc.jpl.nasa.gov:8443/webgeocalc/#NewCalculation (cit. a p. 10).

[11] *ESA WebGeocalc*. URL: http://spice.esac.esa.int/webgeocalc/#NewCalculation (cit. a p. 10).

[12] *Cosmographia*. URL: https://naif.jpl.nasa.gov/naif/cosmographia.html (cit. a p. 10).

[13] *CosmoScout VR repository*. URL: https://github.com/cosmoscout/cosmoscout-vr (cit. a p. 11).

[14] *SpaceEngine Website*. URL: http://spaceengine.org (cit. a p. 12).

[15] *Celestia Website*. URL: https://celestia.space (cit. a p. 13).

[16] *Celestia Website*. URL: https://unity.com (cit. a p. 17).

[17] *Space Graphics Toolkit in the Unity Asset Store*. URL: https://assetstore.unity.com/packages/tools/level-design/space-graphics-toolkit-4160 (cit. a p. 19).

[18] Andrew M. Annex et al. «SpiceyPy: a Pythonic Wrapper for the SPICE Toolkit». In: *Journal of Open Source Software* 5.46 (2020), p. 2050. DOI: 10.21105/joss.02050. URL: https://doi.org/10.21105/joss.02050 (cit. a p. 32).

[19] *JuliaPackaging/Yggdrasil GitHub*. URL: https://github.com/JuliaPackaging/Yggdrasil/tree/master/C/CSPICE (cit. a p. 37).

[20] *NASA Task Load Index (TLX) Website*. NASA. URL: https://humansystems.arc.nasa.gov/groups/TLX/ (cit. a p. 53).