UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY FYZIKY A INFORMATIKY

Bakalárska práca

INTERAKTÍVNA SLNEČNÁ SÚSTAVA NA PLATFORME MARS

Martin Slavkovský

Vedúci bakalárskej práce: Mgr. Matej Novotný, PhD.

Bratislava 2014

Čestné vyhlásenie

Čestne prehlasujem, že som túto prácu vypracoval samostatne s použitím uvedenej literatúry a zdrojov.

V Bratislave 30.5.2014

.....

vlastnoruční podpis

Poďakovanie

Chcel by som týmto poďakovať vedúcemu mojej bakalárskej práce Mgr. Matejovi Novotnému PhD. za cenné rady a pripomienky pri písaní tejto práce. Tiež RNDr. Martinovi Samuelčíkovi a spoločnosti VIS GRAVIS s.r.o za poskytnutie zariadenia na vývoj a otestovanie mojej práce.

Abstrakt

Cieľom bakalárskej práce bolo vyvinúť použiteľnú demonštračú aplikáciu využivajúcu platformu MARS pre zobrazovanie v dvoch kontextoch a v rozšírenej realite. MARS je proprietárna platforma vyvinutá spoločnosťou VIS GRAVIS. Systém využíva dotykový stôl pre interakciu a zobrazovanie dát v primárnom 2D kontexte a rozšírenú reality pre zobrazovanie dát v sekundárnom 3D kontexte. Výsledkom práce je výuková aplikácia, ktorá vizualizuje slnečnú sústavu. 2D kontext slúžiaci na interakciu a používateľské rozhranie, 3D kontext je animovaný trojrozmerný model slnečnej sústavy reagujúci na interakciu s dvojrozmerným kontextom. Práca je realizovaná v populárnom renderovacom engine Unity.

Kľúčová slová:

Slnečná sústava, MARS, Unity, 3D model, Keplerove zákony, Rozšírená realita

Abstract

The goal of thesis was to create a demonstration application for MARS platform using augmented reality with two contexts. MARS is proprietary platform developed by VIS GRAVIC company. In MARS multitouch table is used for interaction and rendering of primary 2D context and augmented reality in secundary 3D context. Result of thesis is visualization of Solar system which can be used also as educational application. 2D context is used for interaction and grafical user interface. 3D context is animated 3D model of our Solar System , which reacts on input form 2D context. The application is created in popular Unity rendering engine.

Kľúčová slová:

Solar System, MARS, Unity, 3D model, Kepler laws , Augmented reality

Obsah

ÚVOD

V súčasnosti ako aj posledných desať rokov speje svet technológii k čoraz menším a šikovnejším prístrojom. Snažíme sa urobiť naše počítače čo najmenšie a čo najrýchlejšie. S rozvojom týchto technológií prišla firma Apple s dotykovým telefónom, ktorý prevalcoval všetky vtedajšie konkurenčné produkty podobnéhot typu. S príchodom iPhonu a neskôr iPadu nastal „boom" s dotykovými zariadeniami. Neskôr sa pridal Google so svojím Androidom.

S Androidom a iPhonom prišlo tísice aplikácii pre bežných používateľôv. Jeden z hlavných cieľov pre vývojára je prehľadné a intuitívne používateľské rozhranie. To znamená, aby človek, ktorý si aplikáciu stiahol vedel bez rôznych návodov a lúštenia program použit na to, na čo bol určený.

Rozšírená realita (niekedy tiež augmentovaná realita) je označenie používané pre reálny obraz sveta doplnený digitálne vytvorenými objektami. Táto technológia ulahčuje človeku spojiť si reálny a virtuálny svet, a tým aj rýchlejšie pochopiť na čo je aplikácia určená. Človek sa preto snaží stále viac a viac doplnit si prirodzené videnie sveta o dodatočné informácie uložené v našich digitálnych zariadeniach. Táto technológia môže mať veľmi prospešné využitie v mnohých oblastiach nášho života. Lekári, záchranári alebo hasiči by si v budúcnosti mohli pomáhať so zariadeniami s rozšírenou realitou pri vykonávaní verejnej služby. V neposlednom rade by táto rozšírená realita mala veľké využitie vo výučbe. Zoberme si príklad, keď sa deti učia na základnej škole o slnečnej sústave. Keď ukážeme dieťaťu planéty v trojrozmernom priestore(resp. ilúziu 3D hologramu), s ktorou môže interagovať, je oveľa pravdepodobnejšie, že si z hodiny astronómie školák niečo zapamätá.

Dôvod

Jeden z dôvodov, prečo som si vybral vizualizáciu slnečnej sústavy na platforme MARS ako bakalársku prácu je môj záujem o astronómiu. Ďalej by som spomenul záujem o nové technológie a obavu o slovenské školstvo. Keď sa zamyslím nad tým, akým spôsobom je u nás vyučovaná fyzika, matematika ale aj ostatné predmety na základných školách, cítim potrebu aspoň trochu prispieť k zlepšeniu súčasného stavu. Memorovanie suchých informácii a nemoderná výučba spôsobuje to, že sa Slovenská republika rok čo

rok prepadá v rebríčku kvality školstva. Preto je podľa mojho názoru aplikácia rozšírenej reality dobrou demonštráciou toho, akým spôsobom by sa naše vzdelávanie mohlo respektíve malo uberať ďalej.

Cieľ práce

Cieľom práce je oboznámiť sa s platformou MARS(multi-touch augmented reality system) a vytvoriť demonštračnú aplikáciou pre toto zariadenie. Aplikácia bude obsahovať trojrozmernú interaktívnu vizualizáciu slnečnej sústavy a výsledný program by mal byť ukážkou toho ako sa MARS a Unity renderovaci engine dá použiť pri interaktívne výučbe na základných a stredných školách. Aplikácia bude obsahovať všetky planéty a veľke mesiace slnečnej sústavy, ktoré sa budú pohybovať po svojich dráhach podľa platných (ale zjednodušených) fyzikálnych zákonov. Dôraz je kladený na intuitívnosť používateľského rozhrania a na grafickú stránku. Tiež môže byť využitá ako šablóna pre iných študentov prípadne vývojárov, ktorí by chceli niečo podobné vytvoriť buď v Unity samotnom alebo v kombinácii Unity a MARSu.

Obsah práce

V prvej časti práce vysvetlím čo je a ako funguje platforma MARS. Tiež sa budem venovať podobným už existujúcim riešeniam so sytémom rozšírenej reality. Takisto si priblížime Unity engine renderovací engine, jeho základné pojmy, funkcie a vývojové prostredie.

V druhej časti práce si priblížime a vysvetlíme fyziku, ktorú som potreboval na implementáciu simulácie pohybu planéty, konkrétne si prejdeme Keplerovými zákonmi.

V tretej časti práce si prejdeme štruktúrou scény. Ako je navrhnutý Unity projekt , jeho scéna , akým spôsobom je zakomponovaný MARS do aplikácie vizualiácie slnečnej sústavy. Tiež si ukážeme aký spôsobom funguje dotyková interakcia so systémom.

V závere zhodnotím celú prácu. Čo sa nám podarilo dosiahnuť, a kde vidím priestor na prípadné vylepšenia.

1. PREHĽAD V PROBLEMATIKE

V tejto kapitole si priblížime problematiku aplikácii rozšírenej reality. Rozšírena reality (po anglicky augmented reality) je označenie používané pre reálny obraz sveta doplnený počítačom. Inak povedané ide o zobrazenie skutočného sveta a následné pridanie digitálnych prvkov. Napríklad sa može jednať o navigáciu do auta, ktorá by snímala ulice, budovy, cesty a pridávala by k nim názvy respektíve iné informácie. Priblížim platformu MARS, ktorú vyvinula spoločnosť VIS GRAVIS a na ktorej je moja práca postavená. Tiež si priblížime enginy podobného typu a uvediem príklady existujúcich riešení podobných MARSu. Kedže na vývoj demonštračnej aplikácie som použil renderovací engine Unity, budem sa mu v tejto kapitole venovať. Tiež sa budem venovať podobným edukatívnym aplikáciam, ktoré vizualizujú slnečnú sústavu.

1.1 MARS (Multi-touch Augmented Reality System)

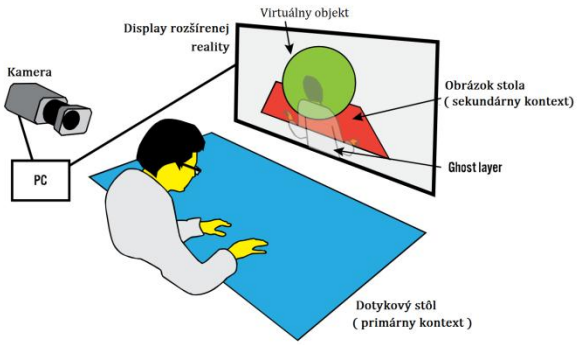
MARS je dotykový systém rozšírenej reality vytvorený spoločnosťou VIS GRAVIS. Bol vytvorený primárne pre edukačné účely a môže byť použitý ako doplnok vo výučbe dejepisu, geografie, fyziky a ďalších predmetov. Prvá demonštračná aplikácia bola určené ako potenciálny doplnok pre výučbu histórie. (Obrázok 2.) [2]

Systém MARS pozostáva z dvoch monitorov a kamery ako je možné vidieť na *obrázku 1*. Na oboch monitoroch sa zobrazuje tá istá scéna (napr. Mapa Bratislavy), kde primárnym kontextom je dotykový stôl. (Monitor č.1) Tento stôl slúži zároveň ako zariadenia na interakciu so systémom. Všetky zmeny scény ako je posunutie, rotácia, škálovanie sú uskutočňované pomocou dotykových giest. Scéna je v primárnom kontexte projektovaná ortogonálne a všetky interakcie s MARSom sú uskutočnované prostredníctvom tohto dotykového stola, teda primárneho kontextu. Sekundárnym kontextom je druhý monitor, v ktorom (Monitor č.2) je už scéna projektovaná perspektívne a spája virtuálne objektu (planéty , 3D model budy) s obrazom z kamery respektíve z reálneho sveta. Ako môžeme vidiet na *Obrázku 2* vytvára sa nám ilúzia ako keby objekt vystupoval zo stola , inak povedané vytvárame ilúzia hologramu.

1.1.1 **Ako to funguje**?

Platforma MARS je vyvinutá v renderovacom engine UNITY. Celá aplikácia je natiahnutá na dva monitory. Ak má jeden monitor rozlíšenie 1920x1080 naša aplikácia bude mať rozlíšenie 3840x1080. V scéne sa nachádzajú dve virtuálne kamery, pričom obe snímaju tú istú scénu a tie isté objekty. Ak pohnem s objektom v rámci prvej kamery, ten istý pohyb objektu zaznamenám aj v rámci druhej kamery. Scéna je akokeby rozdelená na dve časti, ale v skutočnosti je to tá istá scéna a tie isté objekty, ktoré sú duplicitne projektované s inými parametrami a vrstvami do obidvoch monitorov. Prvá kamera projektuje scénu na dotykový stôl ortogonálne a zobrazovanie istej časti objektov môže byť úplne vypnuté pre túto kameru. To už záleží od aplikácie a od toho akú ilúzie chceme vytvoriť. Ako príklad môžeme vidieť na *Obrázku 2* model Dómu svätého Martina, ktorý na primárnom kontexte rederovaný nie je. Obraz druhej virtuálnej kamery je zobrazovaný na monitore č.2. Tento monitor zobrazuje virtuálne objekty (planéty , budovy) spôsobom , že používateľovi sa zdá ako keby vystupovali zo stola pričom reálna kamera sníma dotykový stôl spolu s človekom a premieta tento obraz na druhý monitor. Obraz z kamery prekrývajú virtuálne objekty a ďalšia vrstva, ktorá predstavuje stôl. Na obrázku ju môžeme vidieť označenú červenou farbou. Jedná sa o otextúrovaný obdlžník alebo rovinu prekrývajúcu obraz z kamery. [2]

Pri spustení aplikácie, ktorá funguje na platforme MARS je dôležité dostať druhú virtuálnu kameru (kamera monitora č 2.) do pozície, ktorá korešponduje s reálnym postavením kamery voči dotykovému stolu. Na to nám slúži kalibrácia. MARS má podporu manuálnej a automatickej kalibrácie. Červenú rovinu sa snažíme dostať do pozície, ktorá bude prekrývať obraz stola z kamery. Keď si predstavíme, že sme vypli zobrazovanie červenej roviny na *Obrázku 1*, videli by sme obraz stola nasnímaný kamerou. Kalibrácia kamery nám však zabezpečí, že stôl je dokonale prekrytý červenou rovinou.



Obrázok 1 - Náčrt systému MARS

Obrázok 2 - MARS , mapa Bratislavy

1.2 Unity engine

Unity je komplexný renderovaći herný engine vyvinutý spoločnosťou *Unity Technologies.* Zvyčajne je používaný na výrobu hier pre webové prehliadače, stolové počítače, konzoly a mobilné zariadenia. Pôvodne to bolo prostredie na vytváranie hier na OS X, ale postupne prešiel k iným platformám ako Windows, Linux, Android atď. Výhodou Unity je jeho jednoduchosť a intuitívnosť. Obsahuje kompletnú sadu nástrojov na vytváranie plnohodnotnej trojrozmernej hry. Aj keď Unity mierne zaostáva za svojimi enginami ako sú *Cry Engine*, *Source*, *Unreal Engine,* jeho výhodou oproti konkurentom je otvorenosť a dostupnosť. Unity je dostupné v dvoch verziách. Unity Free je voľne dostupná pre každého a na jej stiahnutie sa stačí vývojarovi zaregistrovať na webstránke. Profesionálnejšia verzia pre tvorbu komerčných aplikácii je Unity Pro. Tento balík obsahuje pokročilejšie funkcie tieňovania, postprocessingu (HDR, Bloom), debugovacích nástrojov a mnoho ďalších. Cena Pro verzie je okolo 1500 dolárov.

V nasledujúcej časti si priblížime niektoré komponenty Unity enginu, kedže ich použitie bolo kľúčove pri tvorbe mojej práce.[1]

Obrázok 3 Ukážka unity enginu

1.2.1 Projekt

Zjednodušene je projekt adresárová štruktúra, ktorú si Unity dokáže spracovať podľa určitých pravidiel. V zásade sa jeden projekt rovná jednej hre respektíve aplikácii. Zvuky, textúry, trojrozmerné modely, *assety*, *scény*, *skripty* a v zásade všetko čo potrebujeme na vytvorenie hry obaľuje projekt. Adresárovú štruktúra projektu v rámci Unity sa rovná štruktúre v rámci systému, čo znamená, že keď vymažeme nejaký súbor v Unity Editore tento súbor sa tiež vymaže zo systému a to isté platí aj naopak. Dôležitou súčasťou projektu sú projektové subory ku skriptom. V prípade C# to bude .sln súbor s projektovým súborom k Visual Studiu. Netreba si však zamienať tieto dva pojmy, pretože Visual Studio projekt bude v tomto prípade len súčasťou Unity projektu. [1]

1.2.2 Scéna

Je trojdimenzionálny priestor obsahujúci všetky *Herné Objekt*. Jedným slovným spojením môžno scénu charakterizovať ako úroveň hry. (level po anglicky) Scénu možno použiť na vytvorenie hlavného menu hry, jednotlivých úrovní hry alebo čokoľvek iného čo potrebujeme pre našu aplikáciu. Dôvod prečo scéna nie je nazývaná úrovňou ako je to

zvykom v hrách je ten, že jedna scéna môže byť fakticky zdielaná viacerými úrovňami. Napríklad hlavné menu bude zdielané všetkými úrovňami aplikácie. V rámci adresárovej štruktúry je scéna súbor s príponou *.unity*. [1]

1.2.3 Balíčky (Packages)

Balíčky sú komprimované súbory obsahujúce modely, textúry, zvuky, skripty atď. Môžeme si pod tým predstaviť zip alebo rar súbor, ktorý je špeciálne určený pre Unity. Hlavná výhoda balíčkov je tá, že umožňujú zabaliť všetky *assety* do jedného súboru kompatibilného s Unity editorom, ktorý môžeme jednoducho importovať do iného projektu. [1]

1.2.4 Predpripravené objekty (prefabs)

Je to kontajner, ktorý umožňuje zoskupovať *assety* a ich nastavenia do templatu, ktorý môžeme neskôr použiť na vytvorenie inštancií viacerých objektov tej istej skupiny *assetov*. Predpripravené objekty môžeme vytvárať dvoma hlavnými spôsobmi. Ako dizajnér budeme vyrába v rámci scény viaceré kópie toho istého predpripraveného objektu a to tak, že si jednoducho naklikáme čo potrebujeme. Druhý spôsob je cez skripty. Ako príklad si môžeme predstaviť planétu v slnečnej sústave. Tento objekt by mal obsahovať otextúrovaný model gule a skript, ktorý zabezpečuje, aby sa nám planéta točila okolo vlastnej osi a okolo Slnka. Aby sme si zakaždým nemuseli vytvárať nový model gule a nastavovať mu tieto parametre, uložíme si prvú planétu do predpripraveného objektu (prefab), ktorý budeme neskôr len kopírovať. [1]

1.2.5 Herný objekt (Game object)

V podstate všetko čo sa nachádza v scéne alebo hre môžeme nazvať herným objektom. Herný objekt môže byť prázdny alebo to môže byť model, textúra , svetlo v zásade čokoľvek. Je to transformovatelný bod v priestore, ktorý môžeme posúvať , rotovať alebo škálovať. *GameObject* má svoje meno a môže byť súčasťou hierarchie objektov. To znamená , že môže byť potomkom alebo rodičom iného herného objektu. [1]

1.2.6 Komponenty

Komponent je zoskupenie parametrov a funkcionality, ktoré definuje *herný objekt.* Každý objekt sa môže skladať z viacerých kompontentov, ktoré definujú jeho vlastnosti. Keď si zoberieme ako príklad herný objekt, ktorý predstavuje planétu slnečnej sústavy, tento herný objekt obsahuje textúru, shader, model, nejaký mesh poprípade skript alebo čokoľvek iné. Všetko tieto veci sú samostatné komponenty herného objektu. Ako ďalši príklad si môžeme uviesť bodové svetlo v Unity. Je to prázdny objekt obsahujúci komponent svetla. [1]

1.2.7 Assety

Je to aspekt unity aplikácie, ktorý bude referenciou v rámci *komponentu* alebo na další *asset.* Textúry, modely, zvuky sa radia medzi externé assety pričom materiály, shadery, prefaby sú internými assetami. Firma Unity Technologies poskytuje v rámci Unity editora, ale aj na webe databázu assetov nazvanú *Asset Store.* [1]

1.2.8 Skripty

Skripty sú dôležitou súčasťou Unity, pretože na vytvorenie plnohodnotnej interaktívnej 3D aplikácie potrebujeme definovať jej správanie. Unity má podporu viacerých programovacích jazykov. Skripty môžu byť písané v *C#, JavaScripte* alebo *Boo*. Spolu s editorom je možné nainštalovať program MonoDevelop, ktorý je plnohodnotným vývojovým prostredím pre C# s debuggerom nakonfigurovaným pre potreby Unity. Pre porovnanie, integrácia Visual Studia s Unity je zdĺhavý proces s pochybnými výsledkami.

Skripty v mojej práci sú napísané v C# a ako IDE som použil MonoDevelop. [1]

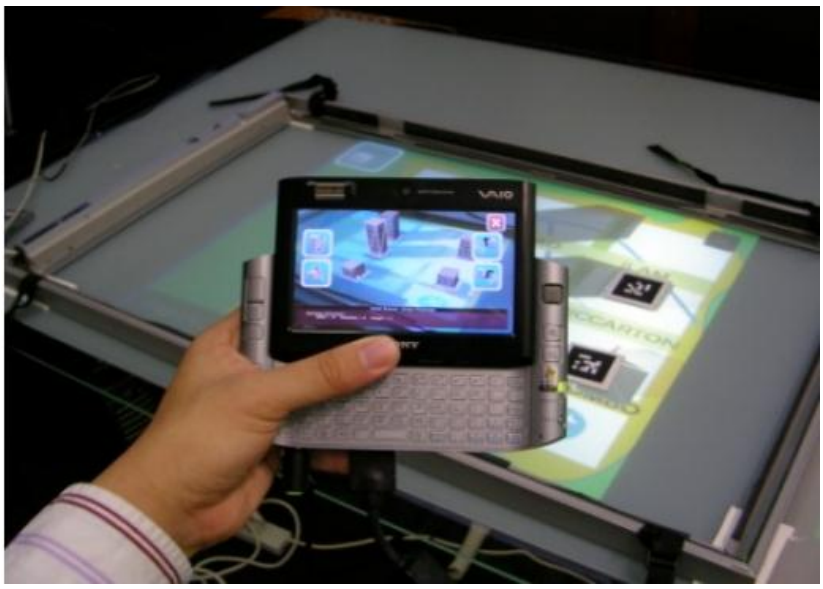
1.3 Podobné riešenia

V nasledujúcej kapitole priblížim niekoľko projektov, ktoré pracujú s rozšírenou realitou. Tiež si spomenieme edukačnú aplikáciu vizualizácie slnečnej sústavy , ktorou som sa v mnohom inšpiroval.

1.3.1 TMAR - Extension of a Tabletop Interface using Mobile Augmented Reality

TMAR je systém rozšírenej reality podobný MARSu vyvinutý Canterburskou univerzitou na Novom Zélande pozostávajúci z dvoch rozhraní : zo špeciálneho zariadenia

slúžiaceho na interakciu, ktoré autori nazývajú ARTable. ARTable sa skladá z priesvitného dotykového framu, pod ktorým sa nachádza projektor a kamera slúžiace na priemietanie scény na stôl alebo plátno. Interakcia s ARTable je možná podobne ako pri MARSe pomocou dotykového stola alebo pomocou reálnych objektov, ktoré sú položené na tomto stole. Tieto objekty majú tvar štvorca a sníma ich kamera nachádzajúca sa pod plátnom projektora. Druhé rozhranie je mobilné zariadenia, ktoré umožňuje paralelnú interakciu so systémom. Na *Obrázku* 4 môžeme vidieť ako systém TMAR vyzerá v reálnom nasadení demonštračnej aplikácie s návrhom developerského projektu. Ako môžeme vidieť na stole sú malé štvorčeky, ktoré sú detekované kamerou a neskôr nahradené digitálnym obsahom (v tomto prípade budovou). Keď štvorec posunieme na stole, zrotujeme ho alebo úplne odstránime, to isté sa stane s virtuálnym objektom na mobilnom zariadení. [3]



Obrázok 4 - TMAR systém

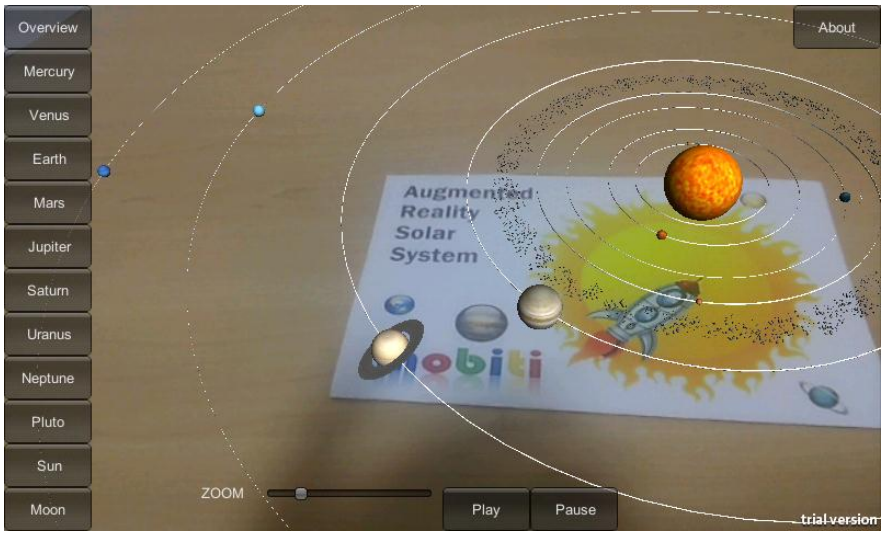
1.3.2 PapArt: interactive 3D graphics and multi-touch augmented paper for artistic creation.

Systém rozšírenej reality, ktorý umožňuje presnú projekciu scény do špeciálneho papierého hárka. Projekcia je vykonávaná projektorom, na ktorom je pripevnená kamera snímajúca papier. Projektor je umiestnený nad hárkom papiera. Dotyková interakcia je vykonávaná pomocou Kinectu, ktorý sa nachádza naľavo alebo napravo od projektora. Zorné pole kamery je obmedzené, preto tento priestor autori nazývajú *display space.*

Interakcia so systémom prebieha dotykovými gestami, ale aj zmenou reálnej pozície alebo natočenia papierového hárka. [4]

1.3.3 Augmented Reality Solar System

Androidová vizualizácia slnečnej sústavy veľmi podobná mojej práci. Edukačná aplikácia postavená na *Unity Engine* v tomto prípade používa ako vstup z reálneho sveta kameru a trackovací obrázok. Kamera sa snaží nájsť o nasnímanom obraze tvar Slnka z konkrétneho trackovacieho obrázka. Ak sa tento tvar nájde slnečná sústava je centrovaná do špecifického bodu tohto obrázka. Interakcia prebieha pomocou mobilného telefónu a pri pohybe telefónu okolo papiera máme pocit ako by sme v priestore pozorovali hologram slnečnej sústavy. Tá je v tomto prípade ozaj len vizualizáciou, pretože sa planéty hýbu lineárne a po kružniciach . V aplikácii sa tiež nedá centrovať pozícia na vesmírne objekty. [5]



Obrázok 5 - Môžeme vidiet obrázok s detskou kresbou slnka

1.3.4 Solar Systen Scope

Nejedná sa o systém rozširenej reality, ale je to edukačná aplikácia vizualizujúca Slnečná sústavu a je naprogramovaná vo flashi. Zároveň je aj pomerne presnou fyzikálnou simuláciou. Aplikáciu je možné spustiť aj v prehliadači alebo ako flash program na počítači. Obsahuje všetky planéty, pás asteroidov, všetky významnejšie mesiace a planétky, kómety, veľké asteroidy atď, ktor. Planéty a kométy sa hýbu podľa platných

fyzikálnych zákonov. Orbitálne dráhy planét sú naklonené a postupom času sa mení aj *excentricita* orbitálnych dráh ako aj ich natočenie vzhľadom na Slnko. Interakcia prebieha pomocou myši. Používateľ môže čas zrýchliť, nastaviť konkrétny dátum, nastaviť realistické veľkosťi a orbity planét. V mnohom som sa toutou aplikáciou inšpiroval, aj keď som nešiel do takých detailov, čo sa týka fyzikálnej časti mojej práce. [6]

2. FYZIKA POHYBU PLANÉT

Kedže moja práca je súčasne aj simuláciou pohybu vesmírnych telies v rámci slnečnej sústavy, v tejto kapitole si priblížime niektoré fyzikálne zákony, ktoré som potreboval na implementáciu. Konkrétne sa jedná a tri keplerove zákony, ktoré definujú pohyb planéty okolo slnka resp. pohyb okolo materského telesa. Kedže témou mojej práce nebola fyzikálna simulácia Slnečnej sústave, nejdem do takej hĺbky ako *System Solar Scope.* Preto som v aplikácii nepoužil Newtonove zákony, ktoré sú potrebné na presnú simuláciu, ale obmedzil som sa len na Keplerove. Planéty v mojej práci teda nebudú meniť napríklad excentricitu svojej orbitálnej dráhy a tiež sa nebudú gravitačne ovplyvňovať. Na vzdialenejšie planéty ako Neptún a Urán tiež nebude aplikovaný Tretí Keplerov zákon.

2.1 Keplerove zákony

Johannes Kepler bol významný nemecký matematik, astrológ a astronóm. Bol jednou z kľúčových postáv vedeckej revolúcie v 17. storočí. Najznámejší bol kvôli svojim prácam *Astronomia nova, Harmonices Mundi* a *Epitome astronomiae Copernicanaea* práve v diele *Astronomia nova* Kepler opísal pohyb nebeských telies pomocou troch zákonov.

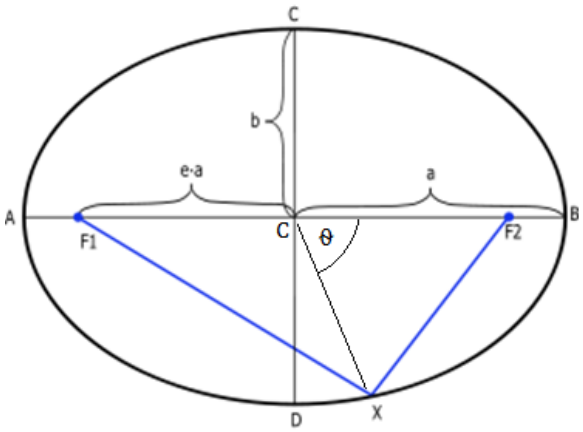
2.1.1 Prvý Keplerov zákon

„Planéty obiehajú okolo Slnka po eklipticých dráhach, pričom Slnko je v jednom z ich spoločných ohnísk.“ [9]

Na *obrázku 6.* môžeme vidieť elipsu, kde Slnko sa bude nachádzať v bode F1 a planéta v bode X. Planéta sa bude pohybovať po eklipticej dráhe podľa stredovej rovnice elipsy v rovine.

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$$

Kde ***a*** je veľkosť hlavnej polosi elipsy a ***b*** je veľkosť vedľajšej polosi.



Obrázok 6 – Elipsa

Elipsu je tiež určená parametrickou rovnicou

$$x = a\,\sin(\theta)$$
$$y = b\,\cos(\theta)$$

ϴ je v tomto prípade uhol medzi hlavnou polosou vektorom $\overrightarrow{CX}$.

Kedže táto rovnica nám určuje len pozíciu planéty (bod **X** na *obrázku 6*) v rovine , musíme rovnicu rozšíriť aj na tretí priestor. To znamená, že musíme bod (p_x, p_y) vypočítaný parametrickou rovnicou pretansformovať do roviny ρ v priestore, ktorá je určená dvoma normalizovanými smerovými vektormi $\vec{u}, \vec{v}$.

Bod ***p*** z kartezíanskej sústavy transformujeme do bodu v priestore nasledovne.

$$x = \vec{u}_x p_x + v_x p_y$$
$$y = \vec{u}_y p_x + v_y p_y$$
$$z = \vec{u}_z p_x + v_z p_y$$

Teda finálna rovnica elipsy v priestore, ktorá leží na rovine ρ bude

$$x = \vec{u}_x\, a\,\sin(\theta) + v_x\, b\,\cos(\theta)$$
$$y = \vec{u}_y\, a\,\sin(\theta) + v_y\, b\,\cos(\theta)$$
$$z = \vec{u}_z\, a\,\sin(\theta) + v_z\, b\,\cos(\theta)$$

Excentricita ***e*** dráhy alebo výstrednosť vyjadruje kruhovosť respektíve nekruhovosť dráhy. Tiež určuje vzdialenosť medzi ohniskami ako ja pomer medzi hlavnou a vedľajšou

polosou. Keď je ***e*** rovné nula tak sa orbitálna dráha rovná kružnici. Pre **0 < e < 1** sa planéta pohybuje po eklipticke j dráhe. Pre **e > 1** bude dráha hyperbolická.

Excentricitu môžeme formálne vyjadriť ako pre eklipticku dráhu ako

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

Keď poznáme excentriciu, môžeme vypočítať pozíciu vzdialenosť ohnísk od stredu elipsy.

$$|F1| = \sqrt{a^2 + b^2}$$

$$|F2\,| = \sqrt{a^2 - b^2}$$

Transformácia do trojrozmerného priestoru je v tomto prípade veľmi jednoduchá. Stačí vynásobiť vektor vektor $\vec{u}$ týmito veľkosťami, kde $\vec{u}$ je jednotkový vektor hlavnej polosi. Teda body F1 a F2 v priestore budú :

$$F1 = |F1|\ \vec{u}$$
$$F2 = |F2|\ \vec{u}$$

2.1.1.1 Apsida

Je bod dráhy kozmického telesa (v našom prípade planéty), v ktorom sa nachádza najbližšie alebo najďalej od centrálneho telesa napríklad od Slnka. Najvzdialenejší bod od centrálneho telesa sa nazýva ***apoapsida*** alebo ***apofokus.*** Naopak najbližší bod k centrálnemu telesu sa volá ***periapsida*** alebo ***perifokus.***

Ak poznáme excentricitu dráhy a veľkosť hlavnej polosi ***periapsidu*** môžeme vypočítať ako

$$r_{per} = (\,1 - e\,)\,a$$

A ***apoapsidu***

$$r_{ap} = (\,1 + e\,)\,a$$

V mojej implementácie majú telesá svoju začiatočnú práve v ***apoapside.*** Vstupné údaje pre orbitálnu dráhu sú v tomto prípade excentricita ***e*** a vzdialenosť centrálneho telesa od ***apoapsidu*** r_{ap} **.** Preto je potrebné vedieť vypočítať dĺžku hlavnej polosi a vedľajšej polosi z týchto dvoch údajov.

Nasledovná formula nám ukazuje ako

$$a = \frac{r_{ap}}{1+e}$$

$$b = a\sqrt{1-e^2}$$

2.1.1.2 Ekliptika a sklon orbitu

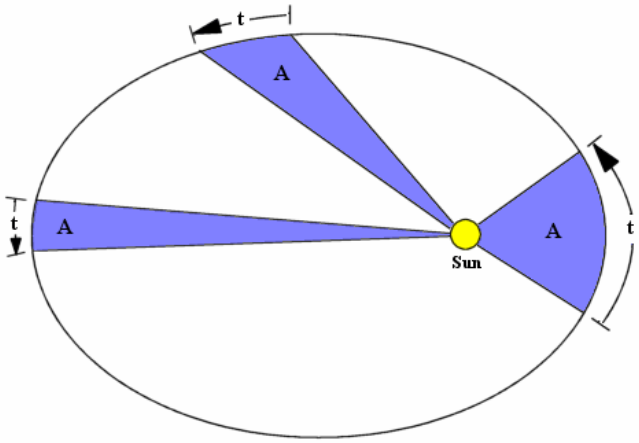
Sklon dráhy vyjadruje uhol planéty alebo mesiaca, ktorý zviera rovina dráhy so základnou rovinou príslušnej sústavy. Pre telesá slnečnej sústavy obiehajúce okolo Slnka je touto rovinou rovina ***ekliptiky.*** Pre kozmické telesá obiehajúce okolo planét (mesiace alebo umele družice) je touto rovinou rovina rovníku príslušnej planéty.

V práci udávam sklon pre jednotlivé planéty v uhloch. Pretože karteziánske súranice sa tranformujú do roviny v priestore, je potrebné si ju najprv vytvoriť tak, že pôvodné smerové vektor $\vec{u}(1,0,0)$, $\vec{v}(0,1,0)$ zrotujeme o zadaný uhol. Vybérame buď medzi $\vec{u}$ alebo $\vec{v}$, v závislosti od toho ako máme orietovanú scénu. [10]

2.1.2 Druhý Keplerov zákon

„Sprievodič (spojnica Slnka a planéty) opíše za rovnaký čas rovnakú plochu.“ [9]

Priamym dôsledkom tohoto zákona je zmena uhlovej rýchlosti resp. Veľkosti vektora rýchlosti v závislosti vzdialenosti planéty od centrálneho telesa. Ako môžeme vidieť na *obrázku 7* planéta opíše sa ten istý čas tú istú modro vyfarbenú plochu teda obsah všetkých troch plôch je ten istý. Čím je planéta bližšie k centrálnemu telesu, tým väčší uhol musí opísať, aby prešla za čas ***t*** *obsah* ***A.*** Najvyššiu orbitálnu rýchlosť dosiahne planéta v ***periapside***, najnižšiu zase v ***apoapside.***



Obrázok 7 - Modré plochy sú opísané za rovnaký času

Druhý keplerov zákon je možné implementovať dvoma spôsobmi.

1) Vypočítame si začiatočnú rýchlosť v apoapside. Tú môžeme dostať nasledovným spôsobom. $\boldsymbol{\mu}$ je gravitačný parameter centrálneho telesa.

$$v_{ap} = \frac{\mu\sqrt{(1-e}}{a\,(1+e\,)}$$

Z minimálnej rýchlosti si vypočítame konštatný obsah A, o ktorý sa budeme posúvať po elipse.

$$\frac{dA}{dt} = \frac{v_{ap}\ r_{ap}}{2}$$

Nasledujúcim spôsobom potom dostaneme rýchlosť planéty v bode X

$$v = \frac{2A}{r\sin(\varphi)}$$

Kde ***r*** je spojnica medzi bodom **X** a centrálnym telesom**.** **φ** vypočítame uhol medzi ***r*** a dotyčnicou v bode **X** , ktorá je určuje smer vektora rýchlosti. Túto dotyčnicu vypočítame pomocou derivácie parametrickej rovnice elipsy. Z vektora rýchlosti v bode X vypočítame jeho uhlovú rýchlosť takto

$$\omega = \frac{v\sin(\varphi\,)}{r}$$

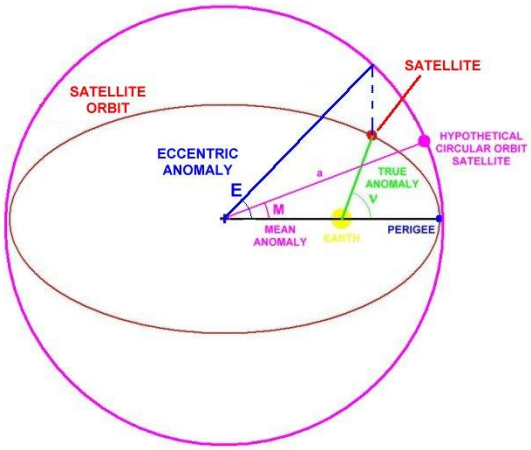
V každom kroku algoritmu pričítame k súčasnému celkovému uhlu ϴ jeho uhlovú rýchlosť ***ω***. ϴ nám vypočíta bod ***X*** v podľa parametrickej rovnice elipsy. Uhol ϴ je v prvom kroku rovný nule. Spočiatku som sa snažil pohyb planét naimplementovať touto metódou, ale pretože nie je možné pomocou tohoto spôsobu určit presnú polohu planéty v ľubovoľnom čase *t*, rozhodol vo finálnej verzii algoritmu použit Keplerovu rovnicu.

2.1.2.1 Keplerova rovnica

Pre určenie polohy telesa na ekliptickej dráhe si môžeme pomôcť opísanou kružnicou, ktorá ma ten istý polomer ako hlavná polos našej elipsy a oba útvary zdieľaju ten istý stred. Na pomocnej kruhovej dráhe sa pohybuje pomocné teleso rovnomerným pohybom čo znamená, že môžeme ľahko vypočítať jeho polohu v akomkoľvek okamžiku. Uhol medzi hlavnou polosou a telesom sa nazýva ***stredná anomália M.*** (ma *obrázku 8* označené ako Mean anomally). Keplerova rovnica má potom tvar. [8]

$$M = E - \varepsilon\sin(E)x$$

Kde $\boldsymbol{E}$ je excentrická anomália, ktorú na *obrázku 8* môžeme vidieť znázornenú modrou farbou a $\boldsymbol{\varepsilon}$ je v tomto prípade excentricita. Kedže stredná anomália sa mení rovnomerne, udáva nám akú časť z periódy prešla planéta okolo Slnka. Z toho vyplýva, že tento údaj vieme veľmi jednoducho prepojiť s časom. Napríklad ak ma obiehajúce teleso strednú anomáliu 90° vieme, že teleso sa nachádza v ¼ svojej periódy.



Obrázok 8 - Náčrt Keplerovej rovnice

Našou úlohou bude potom pomocou tejto rovnice nájsť *excetrickú anomáliu* pre dané $\boldsymbol{M}$. Na výpočet musíme použit Newtonovu iteračnú metódu, pretože ide o transcendetnú rovnicu a tú nie je možné ju vyriešiť algebraicky. Nasledujúci algoritmus ukazuje ako môžeme vypočítať $\boldsymbol{E}$. Vstupným parametrom bude presnosť.

$$E_i = M - \varepsilon \sin(M)\,(1 + \varepsilon \cos(M))$$

$$E_{i+1} = E_i - \frac{E_i * \varepsilon\, sin(E_i) - M}{1 - \varepsilon\, cos(E_i)}$$

Ak rozdiel E_i a E_{i+1} je väčší ako zadaná presnosť tak pokračujeme v iterovaní, ak je rozdiel menší bude výsledok rovný poslednému E_{i+1}.

Po vypočítaní excentricke anomálie musíme ešte získať polohu obiehajúceho telesa. Priamka vedená zo stredu elipsy a odchýlená o $\boldsymbol{E}$ pretne dráhu pomocnej kružnice v určitom bode, z ktorého urobíme kolmý priemet na hlavnú polos. Naša výsledná pozícia v karteziánskej sústave bude

$$x = a\ (\,cos(E) - \varepsilon\,)$$

$$y = b\ sin(E)$$

Do roviny v priestore transforumuje tým istým spôsobom ako iný bod a teda výsledná Keplerova rovnica bude vyzerať takto

$$x = \vec{u}_x\, a\ (\, cos(E) - \varepsilon\,) + \ v_x\, b\ sin(E)$$
$$y = \vec{u}_y\, a\ (\, cos(E) - \varepsilon\,) + \ v_y\, b\ sin(E)$$
$$z = \vec{u}_z\, a\ (\, cos(E) - \varepsilon\,) + \ v_z\, b\ sin(E)$$

2.1.3 Tretí keplerov zákon

„Pomer druhe mocniny obežnej dopy planéty a tretej mocniny jej strednej vzdialenosti od Slnka má pre všetky planéty rovnakú hodnotu“ [9]

Na to, aby sme pomocu Keplerovej rovnice mohli určiť pozíciu obiehajúceho telesa v ľubovoľnom čase nám ešte chýba perióda. Matematicky tretí zákon hovorí, že výráz $\boldsymbol{T^2/a^3}$ je pre všetky planéty slnečnej sústavy alebo v rámci systému (ako samostatný systém rátame aj npr. Jupiter s mesiacmi) je konštatný. Modernejšia formula je

$$\frac{T^3}{a^3} = \frac{4\,\pi^2}{GM}$$

Kde ***T*** je orbitálna perióda, ***M*** je hmotnosť telesa , ***G*** je univerzálna gravitačná konštanta ***a*** a je hlavná polos. V kompletnej Newtonovej formulácii vyzerá rovnica takto

$$\frac{T^3}{a^3} = \frac{4\,\pi^2}{G(M1 + M2)}$$

M1 a M2 sú hmotnosti dvoch obiehajúcich telies okolo seba respektíve dvoch objektov, ktoré sa navzájom gravitačne ovplyvňujú. Hmotnosť Zeme oproti Slnku je tak zanedbateľná , že pre potreby našej aplikácie nám stačí použit prvú formulu. ***GM*** budeme odteraz označovať ako ***μ.*** -

Keplerov tretí zákon používam v rámci aplikácie dvoma spôsobmi. Buď planéte ako Neptún nastavím periódu, pretože je v scéne bližšie k Slnku ako v skutočnosti, alebo nastavím gravitačný parameter. V oboch prípadoch sa druhá veličina dopočíta a to nasledovne.

$$\mu = \ 4\,\pi^2 \frac{a^3}{T^2}$$

Alebo pre periódu platí

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

3. IMPLEMENTÁCIA

V tejto kapitole si podrobne prejdeme konkrétnu implementáciu mojej bakalárskej práce. Budeme sa venovať štruktúre scény a implementácii fyzikálnych zákonov. Vysvetlím podrobne ako som intergroval systém MARS do mojej aplikácie a spôsob interakcie so systémom.

3.1 Štruktúra scény

Celá aplikácia sa skladá z jednej Unity scény, o ktorej by sa dalo povedať, že je vlastne jedninou úrovňou vesmírnej hry. V tejto podkapitole si priblížime jej jednotlivé komponenty, povieme si ako a prečo sú usporiadané tak ako sú usporiadané.

3.1.1 Slnko, Planéty, mesiace, svetlá

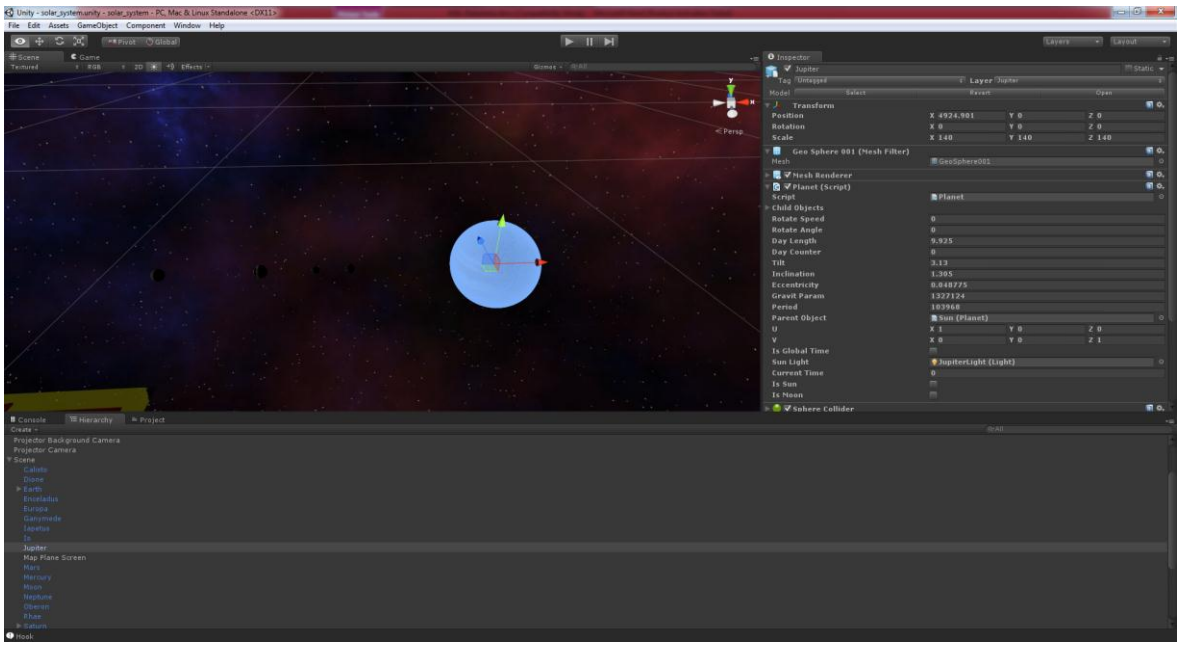
Všetky objekty slnečnej sústavy sú hierarchicky usporiadané v rámci objektu nazvaným *Scene.* Notáciu som prebral z predošlej demonštračnej aplikácie MARSu a netreba si tento pojem zamienať s pojmom *scény* v Unity. *Scene* je v tomto prípade len *herný objekt,* ktorý obsahuje všetky telesá solárneho systému ako jeho potomkov. Preto budem kvôli prehľadnosti nazývať tento herný objekt *SpaceScene. SpaceScene* obsahuje všetky planéty slnečnej sústavy od Zeme po Neptún, Slnko, príslušné svetlá a väčšie mesiace. Tiež je jeho súčasťou špeciálna rovina slúžiaca na interakciu s dotykovým stolom s názvom *MapPlaneScreen,* ku ktorej sa dostaneme neskôr. Všetky planéty a mesiace sú herným objektom, ktorý sa skladá z otextúrovaného modelu sféry, a ktorý obsahuje komponent Planet.cs, čo je skript definujúci základné vlastnosti planéty a jeho pohyb okolo materského telesa. V skutočnosti Planet.cs je súčasťou aj všetkých mesiacov dokonca so špeciálnym prepínačom aj súčasťou Slnka. Planet.cs obsahuje nasledujúce parametre.

- **Rotačná perióda** definuje čas, za ktorý sa planéta alebo mesiac otočí okolo svojej osi. Podľa tohoto parametra sa vypočíta uhol, o ktorý sa má teleso zrotovať po každom tiku časovača.
- **Sklon osi rotácie** definuje natočenie vermírneho telesa k materskému objektu. Pre väčšinu planét a mesiacov je tento parameter takmer rovný nule, ale napríklad pri

Zemi, ktorá ma natočenie okolo 22 stupňov, to spôsobí to, že severný pól je nakolenený počas leta k Slnku.

- **Sklon dráhy** je udávaný v stupňoch a určuje rovinu elipsy, po ktorej sa má vesmírne teleso pohybovať. Skript obsahuje dva smerové vektory $\vec{u}(1,0,0)$, $\vec{v}(0,0,1)$ určujúce rovinu elipsy. Vektor $\vec{u}$ za zrotuje okolo osi x v prípade, že je sklon väčší ako 0.
- **Excentricita** definuje výstrednosť dráhy. Podpora je pre excetricity od 0 po 1. V práci nepočítam s hyperbolickými alebo parablickými dráhami.
- **Gravitačný parameter** slúži ako jeden zo vstupov pre Keplerovu rovincou, ktorou sa vypočítava pozícia planéty v čase. Ak je parameter periódy väčší ako nula, gravitačný parameter sa vypočíta podľa tretieho Keplerovho zákona. V opačnom prípade sa dostaneme periódu práve podľa gravitačného parametra. Dôvod tejto implementácie je ten, že nie všetky vzdialenosti planét od Slnka sú realisticky škálované. Napríklad Jupiter alebo Neptún by mali byť v scéne oveľa ďalej než sú, ale kedže sá má jednať o vizualizáciu, je potrebné niektorým telesám nastaviť nerealistický čas obehu .
- **Orbitálna perióda** definuje čas, za ktorý opíše teleso svoju orbitálnu dráhu. V mojej práci je udávaná v hodinách a vypočítava sa podľa Tretieho Keplerovho zákona. Previazaná je s gravitačným parametrom
- **Rodičovský objekt** je referencia na herný objekt, okolo ktorého bude naše teleso obiehať. Pre planéty je to Slnko, ktoré sa nachádza v strede scény. Pre mesiace je to ich príslušná planéta. K vypočítanej pozícií v rámci elipsy je potom pričítaná pozícia rodičovského objektu, čím zabezpečíme, že planéta alebo mesiac okolo neho skutočne obiehajú.
- **Svetlo** je referencia na herný objekt smerového svetla, ktoré je unikátne pre každú planetárnu sústavu. Napríklad Jupiter a jeho mesiace zdieľajú to isté svetlo, ale Zem má už referenciu na iné. Dokopy je použitých v celej aplikácii osem smerových svetiel pre planetárne systémy Slnečnej sústavy. Dôvod prečo som to takto urobil, bol ten, že Unity nezvládalo správnym spôsobom tieňovanie, ak sa v strede Slnka nachádzalo jedno bodové svetlo, ktoré by bolo zdieľané všetkými planétami. Riešením bolo vytvoriť smerové svetlo, ktoré bude aplikované iba na príslušnú planétu a jeho mesiace. Pri každom tiku časovača bude tohoto svetla nastavený na vektor idúci zo stredu Slnka do planéty, čím zabezpečíme permanentné osvetlenie vesmírneho telesa. Ak chceme aplikovať svetlo iba na konkrétne herné objekty musíme v Unity zadefinovať *vrstvy* a musíme nastaviť jeho *culling mask.* Zoberme si ako príklad Jupiter. Najprv si musíme vyrobiť novú vrstvu, ktorú si napríklad nazveme

JupiterLayer. Potom herného objektu *Jupiter* nastavíme, že spadá práve pod vrstvu *JupiterLayer* a svetlu (pomenujme ho *JupiterLight*) nastavíme jeho *culling mask* opäť iba na *JupiterLayer*. Tento proces zopakujeme pre jeho mesiace, s tým, že nebudeme vyrábať vrstvy a svetlá pre každý zvlášť, ale použijeme *JupiterLayer* a *JupiterLight.*



Obrázok 9 - Obrázok z Unity editora s nastaveniami pre Jupiter

3.1.1.1 Slnko

Slnko je špeciálny typ objektu, ktorý sa nachádza v strede celej scény. Jeho správanie je definované pomocou skriptu Sun.cs, ale aj pomocou skriptu Planet.cs, z ktorého sa používa len rotácia okolo vlastne osi. Herný objekt Slnka tiež obsahuje všetky smerové svetlá pre planetárne systémy a dve špeciálne bodové svetlá, ktoré majú zapnutý efekt koróny. Samotný skript Sun.cs rieši časť interakcie s aplikáciou ako nastavovanie dátumu, zrýchlenie a spomalienie času alebo *kvadratický easing.*

3.1.2 Kamery v scéne

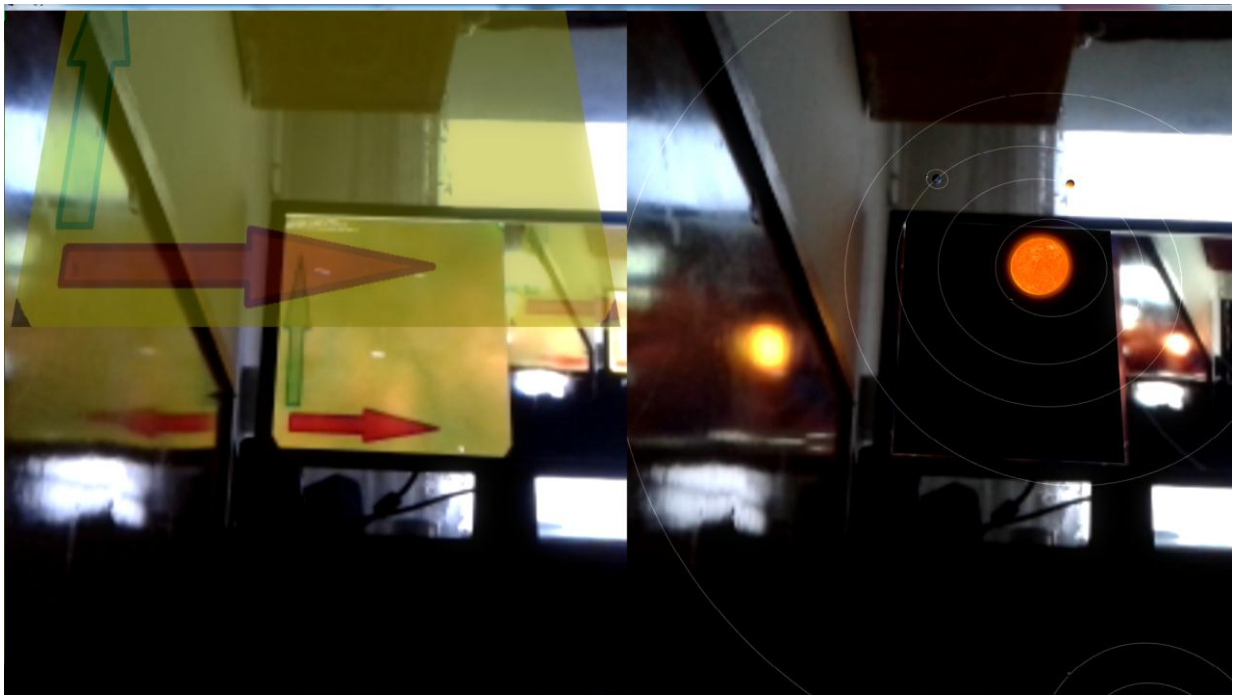
Ako už bolo spomínané MARS funguje na princípe dvoch monitorov, kde do jedného monitora projektujeme scénu ortogonálne a do druhého perpektívne. Virtuálne kamery majú rozdelemú viewport na dve polovice, kde každej kamere pripadá práve jedna.

Dva monitory v skutočnosti predstavujú len rozšírenú pracovnú plochu natiahnutú cez oba displeje. Ak bude mať každý monitor rozlíšenie 1920x1080, potom okno aplikácie bude mať rozlíšenie 3840x1080, čo zabezpečí, že každá virtúalna kamera si bude projektovať scénu na samostaný monitor. V MARSe používame konkrétne práve konfiguráciu s rozlíšením 3840x1080. Herný objekt kamery, ktorá projektuje scénu na dotykový stôl sa nazýva *Screen Camera* a zároveň tento objekt obsahuje skript TouchControl.cs na detekciu dotykov

Herný objekt druhej virtuálnej kamery je *Projector Camera* a projektuje scénu na druhý monitor. Jej pozícia by mala korešpondovať s pozíciou reálnej kamery, ktorá je niekde za dotykovým stolom a sníma ho spolu používáteľom. Na to aby sme to dosiahli nám slúži kalibrácia , ktorá mení pozíciu virtuálnej kamery v scéne. Momentálne MARS podporuje len manuálna kalibráciu pomocou klávesových skratiek, ktoré podrobnejšie vysvetlím v ďalšej podkapitole. Ako pomôcka nám slúži *kalibračná rovina (*v projekte nazvaná ako *Calibration Plane),* ktorú sa snažíme dostať do takej pozície, aby zakrývala primárny kontext (zobrazovaciu časť dotykové monitora). Inak povedané snažíme sa o to, aby sa hrany kalibračnej roviny stretli s hranami displeja.

Na ľavej časti *obrázka 10* môžeme vidieť ako vyzerá sekundárny kontext počas kalibrácie. Vidíme, že na druhom kontexte máme žltú priesvitnú rovinu, ktorej pozícia nekorešponduje s obrazom z kamery respektíve virtuálny objekt kalibračnej roviny nie je „napasovaný" do svojho obrazu. Na pravej časti obrázka máme sekundárny kontext po úspešnej kalibrácii. Jedná sa o pohľad kde sme vypli skybox a obraz dotykového stôla zakrývame čiernou rovinou (názov ma *BlackPlane*), ktorej rodičovským herným objektom je práve kalibračná rovina. Kedže potomkovia v hierarchii herných objektov dedia od svojho rodiča tranformačnú maticu, pozícia oboch rovín je rovnaká.

V scéne sa nachádza ešte jedna virtuálna kamera (*Projector Background Camera),* ktorá obsahuje skripty snímajúce obraz z reálnej kamery. Ten sa potom miesto skyboxu renderuje do sekundárneho kontextu aplikácie.



Obrázok 10 - Aplikácia počas kalibrácie a po nej

3.1.3 Skybox

Skybox je otextúrované pozadie v scéne, ktoré sa nám snaží vytvárať dojem, že je väčšie než v skutočnosti. V Unity má skybox tvar kocky, teda aj požítá textúra musí byť mapovaná ako rozložená kocka. Pre moju prácu používam šesť textúr, kde každá má rozlíšenie 1024x1024 pixelov, a ktoré treba pomocou Unity nástrojov poskladať do jednotného materíalu.

Problém, na ktorý som narazil bol, že štandardný shader pre skyboxy (*RenderFX/Skybox*) nie je možné zrotovat , čo je v mojej aplikácii veľmi dôležité. Preto som tento shader modifikoval, tak že do jeho vstupu pribudol parameter *_Rotation,* ktorý nám umožnuje zo skriptov pomocou matice meniť tomuto skyboxu natočenie a to nasledovne

```
GameObject sceneObjects = GameObject.Find("Scene");
Vector3 rot =
Quaternion rot = Quaternion.Euler( new Vector3(-
      90+sceneObjects.transform.eulerAngles.x,
      sceneObjects.transform.eulerAngles.y,
      sceneObjects.transform.eulerAngles.z));
Matrix4x4 m = Matrix4x4.TRS (Vector3.zero, rot , new Vector3(1,1,1) );
RenderSettings.skybox.SetMatrix ("_Rotation", m);
```

3.1.4 Kreslenie orbitov pomocou komponentu LineRenderer

Orbity, okolo ktorých obiehajú planéty sa kreslia pomocou LineRenderera. Tento komponent obsajuje list N bodov(v unity Vector3 trieda) pospájaných medzi sebou dvojrozmernými obdĺžnikmi, ktorým môžeme nastaviť ľubovoľnú farbu, alfa kanál, textúru a dokonca aj shader. Tento komponent nie je inicializovaný pred spustením aplikácie, inak povedané, je generovaný skriptom Planet.cs až pri spustení aplikácie respektíve pri konštruovaní herného objektu planéty.

Ako môžeme vidieť vo funkcii drawAroundPoint renderer si pri každom tiku časovača vyčistí existujúci list bodov a nastaví jeho veľkosť na 361. V cykle sa potom posúvame po jednom stupni a počítame pozíciu v rámci elipsy podľa uhla. Získanú pozíciu následne musíme pretransformovať do lokálnej pozície v rámci *SpaceScene*, na čo slúži v Unity funkcia TransformPoint

```
public void drawAroundPoint( Vector3 point,LineRenderer renderer ,Transform t )
   if (renderer != null){
     renderer.SetVertexCount(361);
     if (isSelected){
      renderer.SetColors(new Color(1,0.3f,0.3f,0.3f), new Color(1,0.3f,0.3f,0.3f));
        renderer.SetWidth(12,12);
      }else{
        renderer.SetColors(new Color(1,1,1,0.3f), new Color(1,1,1,0.3f));
        renderer.SetWidth(7,7);
      }
      for (int i = 0 ; i <= 360 ; i++ ){
        Vector3 p1 = point + getPosition((double)i,  1.0f , 0 );
        renderer.SetPosition(i, t.parent.transform.TransformPoint(  p1)  );
      }
    }
}
```

3.2 Interakcia s aplikáciou

V tejto podkapitole popíšem akým spôsobom vieme interagovať s aplikáciou. Priblížime si spôsob akým fungujú dotykové gestá, kvadratický easing alebo nastavovanie dátumu.

3.2.1 TUIO a multitouch

MARS používa špeciálnu technológiu na detekciu dotykov nazvanú TUIO. TUIO je opensource framework , ktorý definuje štandardizovaný protokol a API pre multittouch zariadenia. Tento protokol enkóduje pomocou TUIO trackera informácie o dotyku získané priamo z hardvéru a posiela ich do akehokoľvek klienta, ktorý je schopný tieto dáta dekódovať. Technológia bola pôvodne navrhnutá ako abstrakcia pre interaktívne displeje, ale postupne našla využitie v mnohých ďalších oblastiach. Dôvodom použitia TUIO v mojej práci bola neschopnosť UNITY spracovať viacej dotykov naraz. Windows multitouch nebolo v tomto prípade možné integrovať do MARSu , spôsobom akým sme to potrebovali, preto používame konkrétnu implementáciu nazvanú uniTUIO, ktorá práve pridáva podporu pre multitouch. [11]

UniTUIO je sada skriptov, v ktorých je naimplementovaný TUIO tracker spolu s klientom. Aby sme tento plugin mohli použiť v projekte, musíme si v scéne vyrobiť prázdny herný objekt (nazvyme ho tuioStarter), ktorému pridáme ako komponenty skripty BBTouch Manager Starter a BBCrosshair Controller. Tento postup by nam mal zabezpečiť, že po spustení aplikácie budeme schopní spracovať viac ako jeden dotyk. Aplikácia zároveň interpretuje aj dotyky myšou, ale toto sa snažíme ignorovať pri interakcii.

3.2.2 Translácia scény

Na posunutie pohľadu dolava, doprava alebo hore používame transláciu scény. Pretože nechceme meniť pozíciu nakalibrovanej kamery na to, aby sme sa posunuli musíme zmeniť pozíciu herného objektu *SpaceScene* zastrešujúceho všetky objekty, ktorým chceme meniť polohu. Na transláciu scény nám stačí jeden dotyk, ktorý je implementovaný v *uniTUIO* ako *iPhoneTouch* tieda.

Vo funkcii *Update*, ktorú *Unity* volá pred každý vykreslením, sa snažime detekovať koľko *iPhoneTouch-ov* nám vrátilo *uniTUIO*. Ak je tento počet rovný jednej, vieme, že sa

jedná o gesto posunutia. Testujeme či sme už nazačali nejaký pohyb predtým. Ak nie uložíme si do premennej *m_MovePrevPosition* súčasnú pozíciu do *iPhoneTouch-u*. Pri každom ďalšom volaní funkcie *Update* sa nová pozícia pre *SpaceScene* vypočíta nasledovným spôsobom.

$$x = -\text{MoveMultiplier} + (iPhoneTouch.x - MovePrevPosition.\text{x})$$
$$y = -0.5625 * \text{MoveMultiplier}(iPhoneTouch.y - MovePrevPosition.\text{y})$$

Pri translácii scény by malo byť zabezbečené, že posúvaný objekt (máme na ňom prst) nám ostáva pod prstom počas celého priebehu posúvania a po pustení bude mať pozíciu tam, kde sme naposledy mali prst.

3.2.3 Zoom scény

Podobne ako pri translácii ani pri priblížení alebo oddialení scény nechceme meniť pozíciu projekčnej kamery. Rozdiel je v tom, že pri zoome nemeníme ani pozíciu *SpaceScene* smerom po osi y ako by sa mohlo najprv zdať. Ak by sme posúvali scénu, nevideli by sme pri ortogonálnej projekcii, že sa nám planéty zväčšujú. Preto treba objekty škálovať.

Tentokrát testujeme, či nám neprišli v Update metóde dve inštancie triedy iPhoneTouch. Zoomujeme tak, že k sebe postupne približujeme prsty. Opačným pohybom prstov zase odzoomujeme. Ako prvé testujeme, či sme už nemali začatý nejaký predošlý zoom. (tentokrát uložené v premennej *m_ZoomDistance*). Škálu vypočítame tak, že vzdialenosť medzi oboma bodmi dotykov vydelíme premennou *m_ZoomDistance*. Pretože chceme vycetrovať pozíciu scény medzi dva prsty, musíme si najprv vypočítať stred medzi bodmi dotykov, do ktorého potom vypustíme lúč smerujúci z kamery (*Screen Camera*).

Vzorec pre pozíciu bude potom vyzerať nasledovne.

*moveX = -(luc.x - scene.x * newScale - scene.x) / scene.oldScale.x;*
*moveZ = -(luc.z - scene.z * newScale - scene.x) / scene.oldScale.x;*

Obrázok 11 - Nalavo rotujeme scénu, napravo zoomujeme

3.2.4 Rotácia scény

Rotácia podobne ako zoom funguje pri dvoch dotykoch a tomto prípade sa mení natočenie scény. Opäť sa snažíme scénu vycetrovať na stred medzi dvoma bodmi dotykov. V skutočnosti sa v metóde *Update* po detekovaní dvoch iPhoneToucho-v vykoná najprv zoom a potom rotácia, teda môžeme povedať, že obe gestá sa vykonávajú naraz. Keď budeme približovať prsty k sebe a zároveň nám pojde jeden nahor a druhý nadol, budeme vidieť, že sa planéty zväčšujú a zároveň sa mení natočenie scény.

Rotácia tiež používa ten istý lúč, ale má vlastnú premennú na uloženie predošleho uhla. (m_PrevAngle). Natočenie potom vypočítame nasledovne

Angle = -360.0f(Acos(Vector(touch2.x - touch1.x, touch2.y - touch1.y)) - m_PrevAngle)/2/Mathf.PI*

Potom scénu otočíme okolo osi y o tento uhol tak, že rotujeme okolo začiatočného bodu lúča.

Na ľavej casti *obrázka 11* môžeme vidieť akým spôsobom musíme hýbať prsty aby nám scéna zmenila natočenie. V tomto prípade budeme scénu otáčat v smere šípok teda doprava. Na pravej časti zase vidíme akým spôsobom budeme scénu zoomovať.

3.2.5 Podpora pre myš

Kedže dotykový hardvér nebol vždy dostupný, bol som nútený urobiť aj podporu pre myš pri zoomovaní, škálovaní a posunutí. Posunie pre s myšou funguje tým istým spôsobom ako pre dotyk, keď stačí len kliknúť a pohybovať myšou. Rotácia funguje pomocou kláves Q, E a škálovať vieme scrolovaním. Podpora pre myš sa dá počas behu aplikácie vypnúť a zapnúť. Pri nasadení na MARS je lepšie mať myš vypnutú, aby nám náhodou neinterferovala s dotykovými gestami.

3.2.6 Dotykové tlačidlá

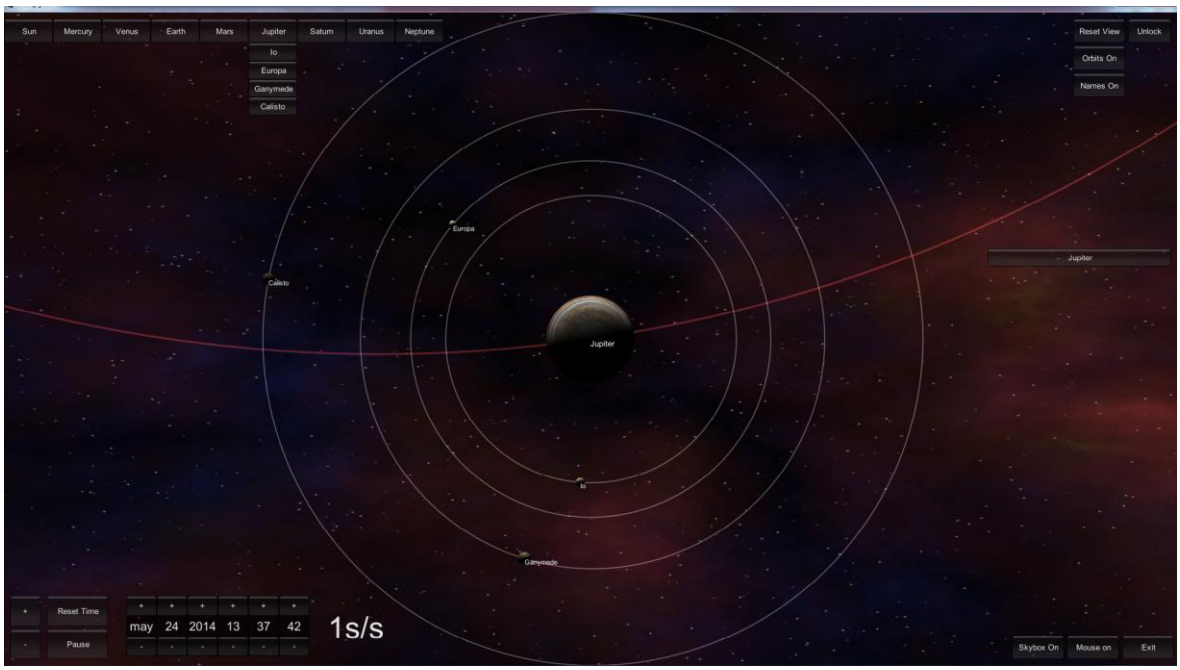
Aj keď Unity má plnú podporu pre GUI (*graphical user interface*) tlačidlá, tvorcovia enginu veľmi nemysleli na dotykové zariadenia. GUI komponenty sa s naším zariadením fungujúcim s *windows mutltitouch-om* správa veľmi neštandardne, keď napríklad tlačidlo ostane označené niekedy aj dve sekundy po pustení. Preto bolo potrebné naprogramovať používateľské rozhranie, ktoré by bolo prispôsobené na dotykový stôl MARSu respektíve TUIO technológiu. Na toto nám slúži trieda KeyButton.

KeyButton obaľuje pôvodné Unity komponenty, konkrétne GUI.Button, s tým, že ignoruje pôvodné akcie. Spôsob akým sa pracuje s odpoveďou na kliknutie tlačidla je úplne zmenený a kontrolovaný triedou KeyButton. Pri implementácii bolo potrebné dosiahnuť to, aby sme mohli vykonávať akciu počas celej doby stlačenia a aby sme zároveň zachovali aj možnosť ju vykonať jednorazovo. Napríklad pri zrýchlovaní času chceme držať tlačidlo stlačené tak, aby nám hodnota rýchlosti času stúpala dovtedy, kým tlačidlo nepustíme. Zároveň však potrebujeme zastaviť čas len jedným stlačením tlačidla. V podstate potrebujeme naimplementovať klasické *OnButtonDown* a *OnButtonPressed*.

Inicializácia KeyButtonu kopíruje spôsob inilizácie GUI.Buttonu z Unity. V skutočnosti sa GUI.Button používa na vykreslovanie nášho nového KeyButtonu, pretože nemalo veľký zmysel prerábať aj vykreslovanie. Prerobená je detekcia a odpoveď na dotyk, kde si uložíme obdĺžnik, ktorý definuje veľkosť tlačidla a snažime sa detekovať, či sa pozícia dotyku nachádza v tomto obdĺžniku. Ak áno poznačíme si , že pri najbližšom voláni funkcie *update* sa má zavolať funkcia definujúca akciu. Akciu môžeme nastaviť pri konštruovaní KeyButtonu pomocou delegátu, čo je v C# vlastne smerník na funckiu. Stlačené tlačidlo zároveň prekreslí na seba tmavý priesvitný obdĺžnik, aby používateľ videl čo má stlačené.

3.2.7 Centrovanie pohľadu na planétu , kvadratický easing

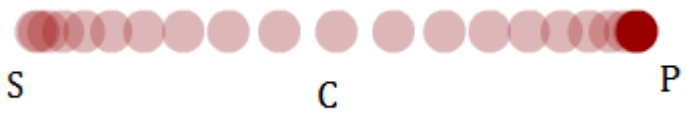
Aby mohol používatel sledovať vesmírne teleso počas jeho obehu okolo orbitu sme sa rozhodli naimplemenovať centrovanie pohľadu na konkrétne teleso pomocou tlačidla s jeho menom.



Obrázok 12 - pohľad vycentrovaný na Jupiter

Na *obrázku 12* môžeme vidieť príklad, kde máme pohľad vycentrovaný na Jupiter. Ten sa náchádza v strede scény a pri zmene pozície Jupitera sa posunieme vždy do takého bodu, aby Jupiter ostal v centre. Pôvodne bolo vycentrovanie na planétu riešené len okamžitým nastavením globálnej pozície na pozíciu planéty, čo bolo však mätúce pre používateľa, ktorý ľahko stratil prehľad o tom kde je.

Na plynulú zmenu pozície k planéte používame kvadratický in/out easing. Vysvetlíme si ho na príklade z *obrázka 13*. Zoberme si, že pozícia scény je v bode S a naša planéta je v bode P a stred medzi úsečkou $\overrightarrow{SP}$ je bod C. Pri easingu bude rýchlosť zmeny pozície stúpať na úsečke $\overrightarrow{SC}$ a klesať o tú istú hodnotu v $\overrightarrow{CP}$. .



Obrázok 13 - kvadratický in/out easing

Funkcia pre easing vyzerá takto [12]

```
double easeInOutQuad(double t, double b, double c, double d){
      t /= d/2;
      if (t < 1) return c/2*t*t + b;
      t--;
      return -c/2 * (t*(t-2) - 1) + b;
}
```

Kde b je začiatočná hodnota, c je zmena v hodnote a d je dĺžka trvania easingu. Parameter t priebežne meníme a jeho možné hodnoty by mali byť v intervale **< 0,d >.** Pri našej konkrétnej implementácii easingu si vypočítame najprv jednotkový vektor smeru z $\overrightarrow{SP}$, nazvyme ho $\vec{d}$. Dĺžku trvania máme nastavenú na 100 framov (v Unity 2 sekundy) a v cykle postupne vypočítavame pozície takto

$$S + easiInOutQuad(i,0\ ,|\overrightarrow{SP}|, 100) * \vec{d};\ i \in <0,100>$$

Stále nám však pretrváva problém pri veľkých zoomoch, pri ktorých človek takisto môže stratiť orientáciu. Preto sa pri veľkých hodnotách priblíženia najprv oddialime, urobíme plynulý prechod k planéte a zase sa priblížime na pôvodnú hodnotu. V skutočnosti používame dva easingy, kde jeden je pre zmeny pozície a druhý pre zoom. Výsledok je zoznam pozícií a zoomov, kde prvých 50 hodnôt sú pozície štartu so zmenšujúcim sa zoomom, ďalších 100 sú rôzne pozície so stálym zoom a posledných 50 sú pozície planéty so zvačšujúcim sa zoomom.

Pri veľmi rýchlom plynutí času by sa mohlo stať, že kým prejdeme k nášmu telesu tak sa stratí z obrazu. Preto musíme výslednú pozíciu planéty alebo mesiaca predpočítať a použiť ju ako parameter výslednej polohu pre funkciu easingu. Na to použime druhý Keplerov zákon, ktorý nám vracia pozíciu vesmírneho telesa v čase.

3.2.8 Dátum a čas

Jednou z dôležitých funkcíi moje aplikácie je nastavenie konkrétneho dátumu alebo zmena rýchlosti plynutia času. Používateľ môže pomocou grafického komponentu kopirujúci androidový DatePicker nastaviť dátum na hodiny, dni, roky, dokonca zobrazuje aj sekundy, ale tie nemenia pozíciu planéty keď ich nastavíme. Súčasťou Unity nebol žiadny komponent, ktorým by sa dal nastavovať dátum, preto potrebné si naprogramovať vlastný. DatePicker sa skladá z 6 menších komponentov nazvaných SinglePicker. Tie dokážu pomocou dvoch KeyButtonov (+ a -) prepínať medzi listom nejakých objektov(čísla , stringy , prakticky hocičo) s tým, že sa vieme dostať z poslednej hodnoty k prvej.

Napríklad z decembra pri stlačení plusu sa dostaneme na január. DatePicker rieši aj priestupné roky a správne dni pri mesiacoch. Ak mame nastavený napríklad dátum na 31. januára a zrazu zmeníme mesiac na február, DatePicker si zistí, že maximálny deň pre február je 28 a nastaví teda dátum na 28. februára. Pre priestupný rok to bude zase 29. Februára.

Na nastavenie dátumu pre planéty sa používa druhý Keplerov zákon. Všetky herné objekty obsahujúce komponent Planet.cs majú premennú CurrentTime, ktorá predstavuje počet uplynutých hodín od 1.1.1904. Maximálny dátum je zase posledný deň v roku 2099. Pri nastavovaní jednotlivých SinglePickerov sa zároveň mení aj CurrentTime pre planéty, čim sa menia aj ich pozície v rámci orbitov.

Čas je možné aj zrýchliť, spomaliť, zastaviť alebo nastaviť na pôvodnú rýchlosť plynutia. Aplikácia používa statickú premennú Sun.TimeConstant, ktorá je pri spustení nastavená na reálne plynutie času. V scéne máme dva KeyButtony, ktoré nám vedia zmeniť premennú Sun.TimeConstant. Rýchlosť plynutia času sa mení lineárne s tým, že akcelerácia postupne narastá v závislosti od toho ako držíme tlačidlo.

Dátum môžeme nastaviť len keď je čas zastavený. Zrýchlovať alebo spomalovať môžeme zase len, keď sme ho nezastavili. Tiež môžeme nastaviť reverzný tok času, čo znamená, že sa planéty budú pohybovať opačným smerom.

3.2.9 Ostatné komponenty používateľského rozhrania

Aplikácia mimo spomínaných komponentov obsahuje aj ďalšie. V krátkosti si ich vymenujeme

- **Vypnutie ovládania myšou :** používateľ má môžnosť úplne vypnúť ovládanie myšou. V tom prípade bude všetky tlačidlá a gestá reagovať len na dotyky. Prepínač môžeme nájsť v dolnej časti rozhrania.
- **Vypnutie vykresľovania orbitov :** ak nechceme, aby sa orbity planét vykresľovali, môžeme použit na to prepínač nachádzajúci sa v hornej časti GUI
- **Názvy pre telesá :** všetky telesá majú pri sebe aj svoje názvy, tak ako ostatné komponenty aj toto je možné vypnúť
- **Vykresľovanie skyboxu :** môžeme prepínať medzi pohľadom rozšírenej reality a medzi vykreslovaním skyboxu. Používateľovi sme chceli zachovať možnosť prepnúť si medzi oboma kontextami.

- **Popis pre planétu :** po vycentrovaní sa na planétu sa nám zjaví v pravej časti obrazovky rozlikávacie okno s základnými informáciami o planéte. V rámci tohoto okna sa tiež nachádza *Unlock* tlačidlo, ktoré nám dokáže zrušit vycentrovanie.

- **Kalibrácia kamery :** manuálna kalibrácia kamery sa spustí stlačením klávesy c. Vykreslovanie virtuálnych objektov a skyboxu sa vypne a objaví sa priestivná žltá rovina. Kombináciou kláves Ľavý Shift + Šípky, Právy Shift + Šípky a Ctrl+Šípky vieme projekčnú kameru rotovať, škálovať a meniť jej zorné pole. Po skončení kalibrácie si ju môžeme uložiť klávesou X a prepnúť sa späť na pôvodné zobrazovanie klávesou C.

ZÁVER

V práci sa nám podarilo úspešne vytvoriť demonštračnú aplikáciu pre MARS platformu a zároveň čiastočnú a graficky atrakívnu vizualizáciu Slnečnej sústavy. Výsledná aplikacia dokáže zobrazovať Slnečnú sústavu systémom rozšírenej reality zároveň môže slúžiť ako edukatívna pomôcka pri výučbe astronómie na základných alebo stredných školách, čím sa nám podarilo splniť väčšinu stanovených cieľov.

Vo výslednom programe je možné urobiť pár vylepšení , ktoré nebolo pre krátkosť času možné naimplementovať. Fyzikálna simulácia planét by mohla byť lepšia, chýba implementácia Newtonových zákonov. Tiež by sa mohol vyriešit problém s tieňovaním, keď zatmenie Slnka a Mesiaca nesedí s reálnymi dátumi a v neposlednom rade dotiahnuť platoformu MARS, tak aby bola možná automatická kalibrácia kamery a lepšie splynutie obrazu z kamery s virtuálnymi objektami.

ZOZNAM POUŽITEJ LITERATURY

1. **Unity fundamentals** [online], z dňa [1.5.2013]
   http://www.3dbuzz.com/training/view/unity-fundamentals

2. **Multi-Touch Augmented Reality System and Methods of Interaction with It**
   Matej Novotný, Ján Lacko, Martin Samuelčík Information Technology Applications, 2012(2), ISSN 1338-6468, pp. 30-36
   http://udv.ull.es/vare/data/vare2013_ID_33_short%20PAPER.pdf

3. **TMAR: Extension of a Tabletop Interface using Mobile Augmented Reality**
   Sewon Na, Mark Billinghurst, Woo Woontack
   Transactions on Edutainmen, 2008, pp. 96-106
   http://www.hitlabnz.org/administrator/components/com_jresearch/files/publications/2008-TMARExtensionofaTabletopInterfaceUsingMobileAugmentedReality.pdf

4. **PapARt: interactive 3D graphics and multi-touch augmented paper for artistic creation.**
   Jeremy Laviole, Martin Hachet
   3DUI - IEEE Virtual Reality Conference, March 4-8 2012, California. (2012)
   http://www.labri.fr/perso/laviole/publis/laviole-3dui.pdf

5. **Augmented Reality Solar System 4.2** [online]], z dňa [1.5.2013]
   http://www.1mobile.com/augmented-reality-solar-system-671880.html

6. **Solar System Scope** [online] , z dňa [1.5.2013]
   http://www.solarsystemscope.com/

7. **J. Giesen** It is often erroneously claimed that Kepler's equation "cannot be solved analytically".
   http://www.jgiesen.de/kepler/kepler.html

8. **Swerdlow, N. M. (2000).** "Kepler's Iterative Solution to Kepler's Equation". Journal for the History of Astronomy 31: 339–341. Bibcode:2000JHA....31..339S.
   http://adsabs.harvard.edu/full/2000JHA....31..339S

9. **Bruce Stephenson** (1994). *Kepler's Physical Astronomy*. Princeton University Press. p. 170. ISBN 0-691-03652-7.

10. **Astronomy Picture of the Day** [online]. NASA, 1996-9-21, [cit. 2010-04-25]. http://apod.nasa.gov/apod/ap960921.html

11. **TUIO protocol** [online] , z dňa [1.5.2013]
    http://www.tuio.org/?specification

12. **Easing equations by Robert Penner** [online] , z dňa [1.5.2013]
    http://gizma.com/easing/

Zoznam príloh

Príloha č. 1 : DVD-R médium s elektronickou formou tohto dokumentu spolu s výslednou aplikáciou a zdrojovými kódmi v podobe *unity* projektu.

ZOZNAM PRÍLOH