Simulacija planetarnog sustava u 3D okruženju

Benčević, Bruno

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

SIMULACIJA PLANETARNOG SUSTAVA U 3D OKRUŽENJU

Završni rad

Bruno Benčević

Osijek, 2021.

SADRŽAJ

1. UVOD

Računalna grafika je doživjela brz napredak posljednjih nekoliko desetljeća. Više nije problem prikazivati jednostavno grafičko sučelje, 2D video igre ili hiperrealističke simulacije u trodimenzionalnom okruženju jer moderna računala sadrže vrlo jake grafičke procesorske jedinice. Kako je čovječanstvo oduvijek bilo zadivljeno veličinom i kompleksnošću svemira, napretkom u istraživanju svemira sve je veća potreba za simuliranjem svemira i planetarnih sustava. Jedna od najpotrebnijih simulacija za ljude je simulacija Sunčevog sustava, odnosno predviđanje putanja planeta za svemirske misije, a najbitnije – predviđanje putanja asteroida koji se nasumično kreću Sunčevim sustavom. U ovom radu se opisuje postupak razvoja simulacije planetarnog sustava kroz koji je moguće detaljno vidjeti planete koji kruže oko glavne zvijezde čiji su odnosi veličina i udaljenosti od Sunca proporcionalni realnim veličinama te asteroide koji se nasumično kreću sustavom djelovanjem gravitacijskih sila planeta.

1.1. Zadatak završnog rada

Zadatak završnog rada je napraviti aplikaciju za simuliranje planetarnog sustava u trodimenzionalnom okruženju koji sadržava glavnu zvijezdu, planete i asteroide. Potrebno je omogućiti korisniku kretanje kroz planetarni sustav, dodavanje i brisanje planeta i asteroida te kontroliranje brzine simulacije.

2. PREGLED PODRUČJA

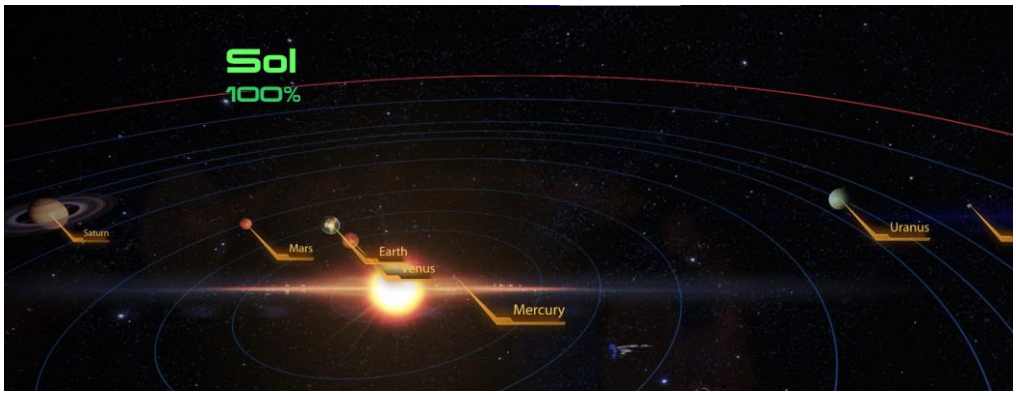
Prilikom izrade modela planetarnog sustava, glavni problem je proporcionalno umanjivanje veličina i udaljenosti planeta od središnje zvijezde kako bi se planet mogao prikazati dovoljno velikim na zaslonu unutar planetarnog sustava. Rješenje ovog problema je uglavnom zanemarivanje omjera stvarne udaljenost planeta od središnje zvijezde i njegove veličine. Najčešće se implementira korištenjem realnih vrijednosti veličina i udaljenosti planeta, a pri prikazivanju planeta na zaslonu koristi se nekoliko magnituda veći model planeta. Time simulacija pri računanju gravitacijske sile, koristeći Newtonov zakon gravitacije, između planeta koristi realne vrijednosti neovisno o prikazanoj veličini planeta.

Na skali planetarnog sustava, svi asteroidi i planeti se vrlo sporo kreću te se korisniku u većini slučajeva omogućuje ubrzavanje kretanja asteroida i planeta na način da se ubrza provođenje simulacije. Također je potrebno omogućiti zaustavljanje simulacije ako korisnik želi detaljno promotriti njeno stanje ili stanje nekog planeta.

Simuliranje planetarnog sustava, osim u znanstvene svrhe poput predviđanja putanje asteroida ili položaja planeta, se također koristi u video igrama tematike istraživanja svemira.

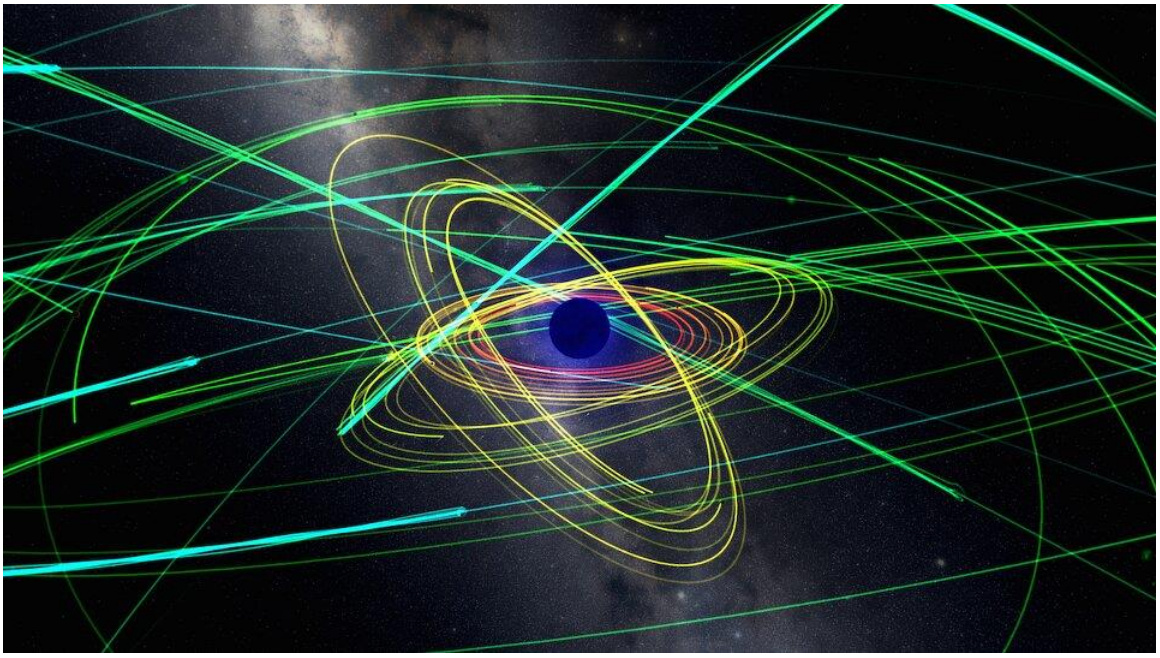
2.1. Postojeća rješenja

Mass Effect 2 je akcijska video igra koju je razvio BioWare 2010. godine čija se radnja odvija u 22. stoljeću u cijeloj Mliječnoj stazi. Kao dio misija koje igrač mora obaviti, igračima je omogućeno putovanje kroz više planetarnih sustava diljem Mliječne staze, uključujući i Sunčev sustav kroz koji je moguće kontrolirati svemirsku letjelicu te detaljno istražiti svaki planet. Sunčev sustav prikazan u igri se kontinuirano simulira te prolaskom vremena u igri se položaj planeta mijenja [1].



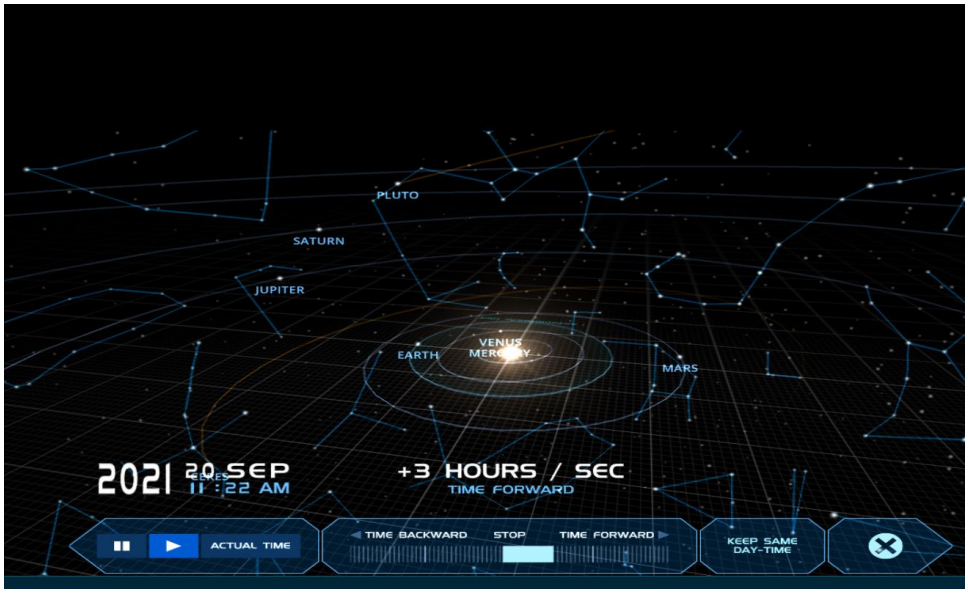
Slika 2.1. *Prikaz Sunčevog sustava u igri Mass Effect 2*

Universe Sandbox je interaktivan simulator svemira koji omogućuje simuliranje planetarnih sustava, galaksija i crnih rupa. Na sve objekte unutar simulacije djeluje gravitacija, a sudari dvaju tijela daju kompleksne rezultate poput raspadanja većih asteroida ili izbacivanja planeta iz njihovih putanja. Korisnik također može dodavati na proizvoljnoj poziciji nove asteroide, planete, zvijezde i crne rupe koje imaju utjecaj na daljnji razvoj simulacije. Osim simulacije gravitacije, Universe Sandbox omogućuje detaljnu simulaciju klime na Zemlji koja ovisi o Zemljinoj udaljenosti od Sunca i godini simulacije. Pozadina simulacije se također simulira te je moguće vidjeti kretanje crnih rupa ili supernova. Universe Sandbox jer razvila tvrtka Giant Army 2012. godine [2].



Slika 2.2. *Prikaz orbita planeta planetarnog sustava unutar programa Universe Sandbox*

Solar System Scope je interaktivna web stranica koja pruža aplikaciju za simuliranje Sunčevog sustava. Omogućeno je postavljanje brzine simulacije koja se može odvijati u realnom vremenu, ubrzano ili unatrag. Korisnik može vidjeti putanje koje planeti prate te postaviti kameru da prati označeni planet čiji se podaci ispisuju na zaslon. U pozadini simulacije se nalazi statična 3D karta poznatih zviježđa. Solar System Scope je također izrađen za mobilne platforme i za osobna računala pri čemu nije potrebna internetska veza [3].



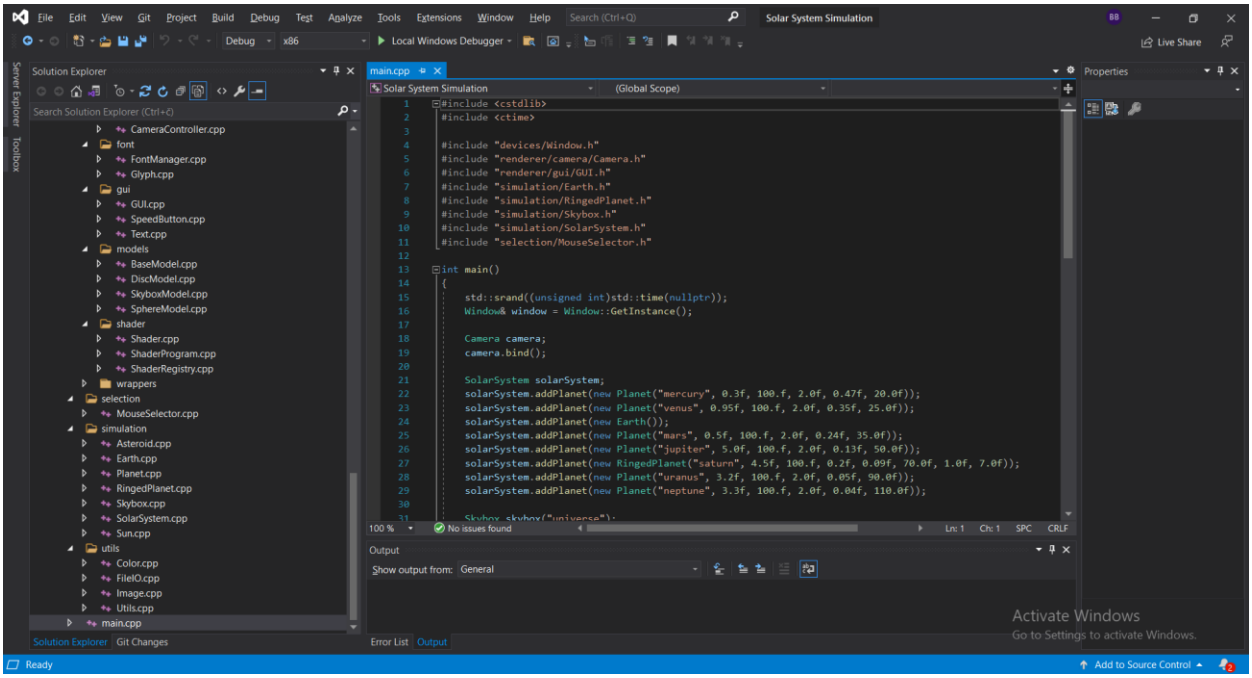
Slika 2.3. *Prikaz korisničkog sučelja Solar System Scope simulatora*

3. KORIŠTENI ALATI I TEHNOLOGIJE

Izrada simulacije planetarnog sustava zahtjeva kompleksniju strukturu projekta za koju je potrebno napredno integrirano razvojno okruženje te API (engl. *Application Programming Interface*) za kreiranje prozora, prikazivanje 3D okruženja na zaslon i učitavanje komprimiranih slika koje će se koristiti za prikaz planeta i asteroida. U sljedećim poglavljima objašnjeni su alati i tehnologije korištene za razvoj simulacije.

3.1. Microsoft Visual Studio

Microsoft Visual Studio 2019 je integrirano razvojno okruženje koje je razvio Microsoft. Omogućuje razvoj programskih rješenja u većini najkorištenijih programskih jezika poput C, C++, C#, Java, JavaScript, Python i drugih. Sadrži alate za uređivanje koda (engl. *code editor*), ispravljanje pogrešaka (engl. *debugging*), uređivanje postavki (engl. *properties editor*) i navigiranje projektom (engl. *solution explorer*) [4].



Slika 3.1. *Microsoft Visual Studio 2019 korisničko sučelje*

3.2. Microsoft Visual C++ 2019

Programski jezik C++ je jezik opće namjere kojega je razvio Bjarne Stroustrup. Nastao je kao nadogradnja na C programski jezik dodajući mogućnost korištenja klasa i objekata, a samim time korištenje polimorfizma i hijerarhija klasa koristeći svojstvo nasljeđivanja klasa. Microsoft Visual C++ je kompajler (engl. *compiler*) za C i C++ programske jezike kojeg je razvio Microsoft. Korištena implementacija C++ kompajlera podržava C++17 standard. Za korištenje Microsoft Visual C++-a potrebno je imati instalirane C++ redistributivne biblioteke za učitavanje pri pokretanju programa.

3.3. OpenGL i GLSL

OpenGL je aplikacijsko programsko sučelje (engl. *API*) za razvoj programa koji koriste 2D ili 3D grafiku. Implementira se kao standardizirani set funkcija za generiranje međuspremnika podataka (engl. *buffers*), tekstura (engl. *textures*), programa za sjenčanje (engl. *shader program*) te postavljanje podataka unutar programa – uniformi (engl. *uniforms*). Prilikom korištenja OpenGL API-a, koriste se funkcije za generiranje i postavljanja objekata, dok je proces prenošenja podataka na grafičko procesorsku jedinicu i na zaslon apstraktiran od programera [5]. U nastavku su objašnjeni najčešće korišteni OpenGL objekti.

VAO (engl. *Vertex Array Object*) predstavlja niz vrhova modela spremljenih u međuspremnike. Pomoću VAO-a se definira raspored podataka unutar međuspremnika, te je moguće definirati najviše četiri vrste podataka za pojedini vrh – koordinate vrha, normala vrha, koordinate teksture vrha te boja vrha.

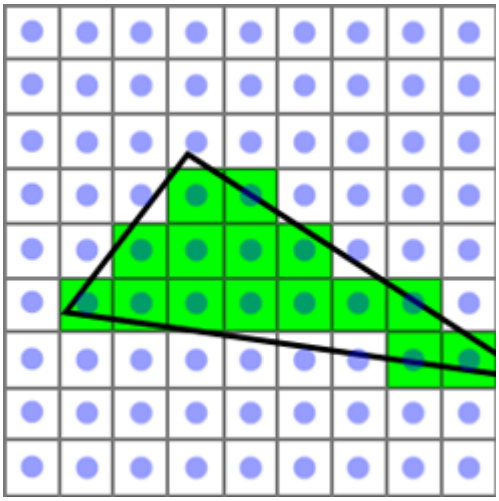
VBO (engl. *Vertex Buffer Object*) i EBO (engl. *Element Buffer Object*) predstavljaju međuspremnike podataka za model. VBO je međuspremnik koji sadrži sve podatke o svim vrhovima modela, dok EBO sadrži indekse vrhova unutar VBO-a. Svi modeli prikazani na zaslonu se sastoje od seta trokuta definiranih s tri vrha koji mogu definirati više od jednog trokuta. Kako se ne bi duplicirali podaci o trokutima, koristi se EBO kako bi se ponovno mogli iskoristi.

Tekstura (engl. *Texture*) predstavlja sliku koja je spremljena u memoriji grafičke procesorske jedinice te se identificira se jedinstvenim brojem (engl. *ID*). Za definiranje dijela teksture koji se želi prikazati na određenom dijelu modela, koriste se koordinate teksture koje se također nazivaju UV koordinate (engl. *UV coordinates*). One su uređeni par čije komponente mogu poprimiti sve vrijednosti u intervalu [0.0, 1.0], gdje 0.0 označava donji lijevi kut teksture, a 1.0 gornji desni kut

teksture. Prilikom prikazivanja teksturi na zaslonu, potrebno je postaviti način na koji će se interpolirati boja piksela s teksture na zaslon koristeći zadane UV koordinate. Najčešće korištene tehnike za interpolaciju piksela su GL_NEAREST i GL_LINEAR (slika 2.2.). GL_NEAREST metoda ne koristi linearnu interpolaciju boje piksela, već uzima najbliži odgovarajući piksel teksture za piksel na zaslonu dok GL_LINEAR metoda koristi interpoliranu vrijednost piksela teksture pri čemu se interpolira vrijednost susjednih piksela.

Slika 3.2. *Najčešće metode interpolacije piksela teksture u OpenGL-u [6]*

Programi za sjenčanje (engl. *shader programs*) su jednostavni programi kojima se upravlja podacima dobivenih iz niza vrhova modela te se određuje što će se na zaslonu prikazati. Sastoje se od dva glavna dijela: dio koji procesira podatke o vrhovima modela (engl. *vertex shader*) i dio koji procesira podatke za svaki piksel na zaslonu individualno (engl. *fragment shader*). Vertex shader kao ulazne podatke prima sve podatke o vrhovima modela te može njima manipulirati na bilo koji način. Uglavnom se koristi za primjenjivanje osnovnih transformacija lokalnih koordinata virtualnog okruženja u normalizirane koordinate na zaslonu. Zatim se iz niza vrhova modela, koristeći podatke iz međuspremnika indeksa, tvore trokuti čije su koordinate relativne koordinatama zaslona. Rasterizacijom (slika 3.3.) se dobiva boja za svaki piksel koja je određena ovisno o položaju trokuta. Ti se podaci šalju fragment shaderu koji za svaki piksel pojedinačno određuje boju primjenjujući metode osvjetljenja poput Phongovog modela ovisno o broju izvora svjetlosti u virtualnom okruženju.



Slika 3.3. *Vizualni prikaz procesa rasterizacije [7]*

Programi za sjenčanje se programiraju koristeći GLSL jezik (engl. *OpenGL Shading Language*). Sintaksa GLSL jezika je slična C programskom jeziku, ali omogućuje razvoj programa za sjenčanje neovisno o platformi i grafičko procesorskoj jedinici na kojoj je namijenjeno korištenje programa. GLSL jezik omogućuje korištenje funkcija i operatora koji se također nalaze u C i C++ programskim jezicima te sadrži ugrađene trigonometrijske, vektorske, eksponencijalne i logaritamske matematičke funkcije.

```
#version 330 core

layout (location = 0) in vec3 in_position;

out vec3 texture_coordinates;

uniform mat4 projection;
uniform mat4 view;

void main()
{
    // view - discarding camera translation
    vec4 position = projection * mat4(mat3(view)) * vec4(in_position, 1.0);
    gl_Position = position.xyww;
    texture_coordinates = in_position;
}
```

Slika 3.4. *Primjer programa za sjenčanje napisan GLSL jezikom*

3.4. GLEW i GLFW

GLEW (engl. *OpenGL Extension Wrangler Library*) je višejezična biblioteka javno dostupnog koda namijenjena korištenju na svim platformama. Služi za jednostavno učitavanje temeljnih funkcija i ekstenzija OpenGL-a dostupnih na korištenoj platformi. OpenGL standard definira funkcije koje moraju biti implementirane u upravljačkim programima grafičkih kartica, dok je GLEW zaslužan za učitavanje i inicijalizaciju pokazivača na te funkcije u memoriju [8].

GLFW (engl. *Graphics Library Framework*) je višejezična biblioteka razvijena u C programskom jeziku. Namijenjena je razvoju projekata koristeći OpenGL, OpenGL ES ili Vulkan aplikacijsko programskim sučeljima na Windows i macOS platformama. GLFW omogućava jednostavno stvaranje prozora, upravljanje kontekstima OpenGL-a, upravljanje unosima s vanjskih uređaja poput tipkovnice, miša i joysticka te kreiranje slušatelja događaja s vanjskih uređaja [9].

3.5. STB Image biblioteka

STB Image Biblioteka (pravi naziv stb_image.h) je biblioteka s jednom zaglavnom datotekom koju je razvio Sean Barrett te je dostupna pod MIT licencom [10]. Napisana je za projekte razvijene u C ili C++ programskim jezicima te je namijenjena za jednostavno učitavanje slika iz komprimiranih formata poput PNG, JPG, BMP i sličnih u memoriju.

3.6. Paint.NET

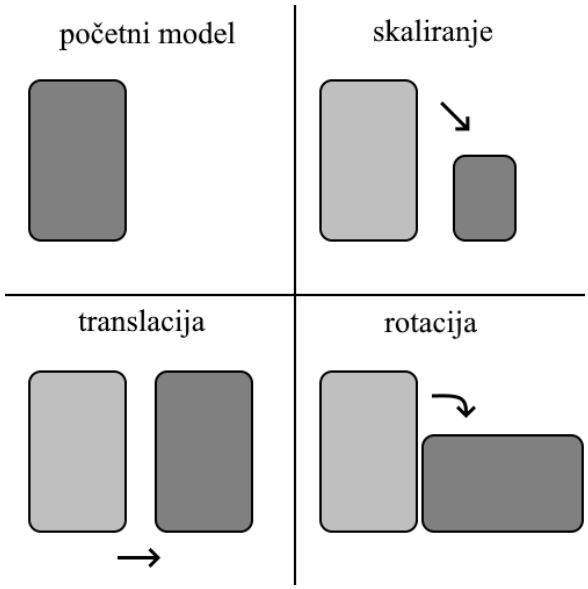
Paint.net je besplatan program za uređivanje slika kojeg je razvio Rick Brewster za Microsoftovo .NET okruženje. Podržava razvoj i dodavanje dodataka (engl. *plugins*) te omogućuje korisniku dodavanje slojeva na sliku. Podržava sve popularne formate slika koje može učitati i zapisivati [11]. Za ovaj projekt je korišten za smanjivanje veličine slika te za kreiranje testnih slika korištenih prilikom razvoja programa.

4. IMPLEMENTACIJA GLAVNIH FUNKCIONALNOSTI

Za izvedbu programa za prikaz 3D okruženja, potrebno je poznavati osnovne koncepte transformacije i projiciranja modela na zaslon koristeći virtualnu kameru te određenu metodu osvjetljavanja modela. Simulacija putanja asteroida zahtijeva implementaciju Newtonovog zakona za gravitaciju, a odabir planeta na zaslonu naprednije vještine korištenja linearne algebre. Implementacija navedenih funkcionalnosti se nalazi u nastavku ovog poglavlja.

4.1. Osnovne transformacije i virtualna kamera

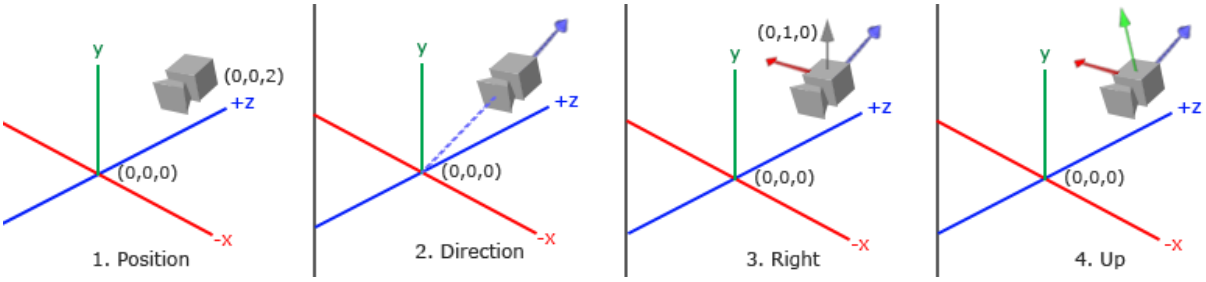
Bilo kakva transformacija modela u trodimenzionalnom okruženju je kombinacija triju osnovnih transformacija: skaliranja, translacije i rotacije. Skaliranje modela podrazumijeva množenje pozicija svih vrhova nekom konstantom, pri čemu se i dužina, visina i širina modela neovisno mijenjaju. Translacija predstavlja pomicanje modela u okruženju, a rotacija rotiranje modela oko zadane osi. Rotiranje je moguće oko tri glavne osi Kartezijveog koordinatnog sustava, ali je moguće i oko proizvoljne osi. Osnovne transformacije su prikazane na slici 4.1.



Slika 4.1. *Osnovne transformacije modela*

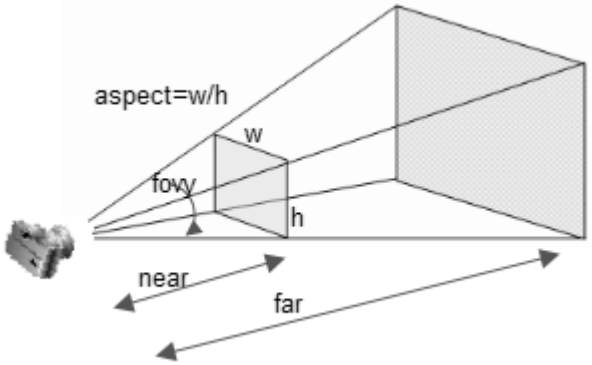
Virtualna kamera je temeljni objekt kojim se služi u razvoju 3D okruženja. Predstavlja poziciju u 3D okruženju iz koje se gleda na ostale objekte te upravo ono što se nalazi u gledištu kamere je ono što se prikazuje na zaslonu. Matematički gledano, virtualna kamera služi za transformaciju koordinata vrhova modela iz globalnih koordinata 3D okruženja u koordinate relativne kameri što se izvodi množenjem vektora pozicije vrha matricom pogleda (engl. *view matrix*). Komponente

matrice pogleda su tri međusobno okomita vektora koja definiraju koordinatni sustav čije je ishodište pozicija kamere te služe za transformaciju baze koordinatnog sustava 3D okruženja u bazu koordinatnog sustava virtualne kamere. Druga komponenta je položaj kamere koji služi za translaciju objekata relativno kameri.



Slika 4.2. *Vizualni prikaz procesa definiranja koordinatnog sustava relativnog virtualnoj kameri [6]*

Za prebacivanje koordinata vrhova modela iz 3D koordinatnog sustava virtualne kamere u 2D koordinatni sustav zaslona, potrebna je transformacija matricom projekcije (engl. *projection matrix*). Matrica projekcije uzima u obzir veličinu zaslona ili prozora na koji će se prikazati model, kut gledanja koji definira veličinu prostora koji se nalazi u gledištu virtualne kamere te udaljenosti od kamere između kojih se nalaze vrhovi modela koje želimo prikazati. Takva metoda projekcije se naziva perspektivna projekcija koja modele blizu virtualne kamere prikazuje većima na zaslonu od onih koji su dalje od kamere.



Slika 4.3. *Vizualni prikaz parametara koji definiraju perspektivnu projekciju [12]*

4.2. Generiranje modela sfere

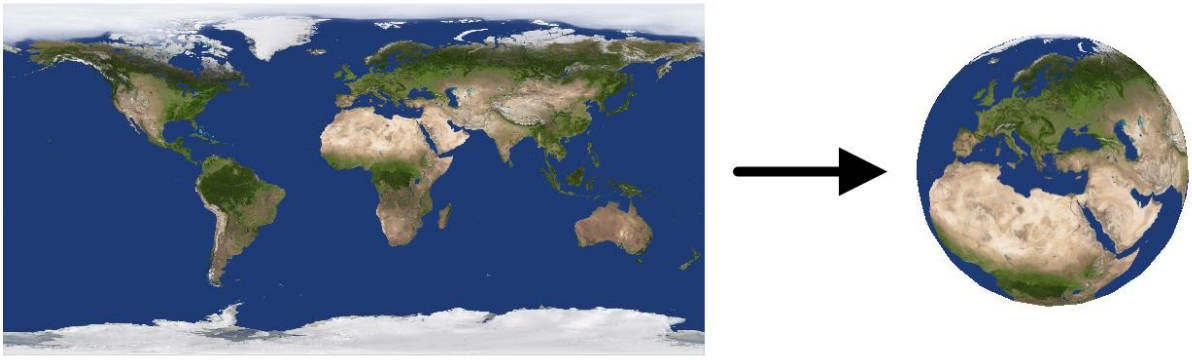
Za prikaz Sunca, planeta i asteroida, u projektu se koristiti isti model – model sfere. Umjesto učitavanja gotovih modela sfere, implementirano je generiranje sfere s četiri parametra: radijus, broj koraka (engl. *step count*), nepravilnost (engl. *irregularity*) i hrapavost (engl. *roughness*).

Proces generiranja sfere, odnosno njenih vrhova (engl. *vertices*), se odvija preko polarnih koordinata koje se kasnije pretvaraju u Kartezijeve koordinate. Umjesto for petlji za x, y i z koordinate, koriste se for petlje za horizontalni i vertikalni kut. Horizontalni kut obuhvaća sve vrijednosti u intervalu [0°, 360°], a vertikalni kut obuhvaća vrijednosti intervala ⟨-90°, 90°⟩. Veličina sfere određena je radijusom, a broj uređenih parova kutova, odnosno detaljnost modela, određeni su parametrom broja koraka. Za generiranje modela planeta, parametar hrapavosti se koristi kako bi se dodalo odstupanje normale vrha od njene normalne vrijednosti dodavajući nasumični pomak na kutove čime se dobije efekt grbave površine planeta bez korištenja tekstura za isti efekt. Modeli asteroida koriste sve parametre, a parametar nepravilnost se koristi kako bi površina sfere izgledala grbavo s oštrim rubovima dodavanjem nasumične vrijednosti na radijus.

```
void SphereModel::setupVertexBuffer()
{
    std::vector<float> vertices;
    int verticalStepCount = getVerticalStepCount();

    for (int j = -verticalStepCount; j <= verticalStepCount; j++)
    {
        float pitch = (float)j / verticalStepCount * 90.0f;

        for (unsigned int i = 0; i <= stepCount; i++)
        {
            float yaw = (float)i / stepCount * 360.0f;

            Vector normal = getVertexNormal(yaw, pitch);

            float u = (360.0f - yaw) / 360.0f;
            float v = (pitch + 90.0f) / 180.0f;

            Vector position = getVertexPosition(yaw, pitch);

            vertices.insert(vertices.end(), {
                    position.x, position.y, position.z,
                    u, v,
                    normal.x, normal.y, normal.z
                });
        }
    }

    vertexBuffer = Buffer::CreateVertex(vertices);
    vertexBuffer.bind();
}

Vector SphereModel::getVertexPosition(float yaw, float pitch) const
{
    if (irregularity == 0.0f) return Vector::FromPolar(radius, yaw, pitch);

    float radiusOffset = (90.0f - std::abs(pitch)) / 90.0f * irregularity * Math::CreateRandom(0.2f, 0.8f);

    return Vector::FromPolar(radius + radiusOffset, yaw, pitch);
}

Vector SphereModel::getVertexNormal(float yaw, float pitch) const
{
    if (roughness == 0.0f) return Vector::FromPolar(1.0f, yaw, pitch);

    float yawOffset = Math::CreateRandom(-roughness, roughness);
    float pitchOffset = Math::CreateRandom(-roughness, roughness);

    return Vector::FromPolar(1.0f, yaw + yawOffset, pitch + pitchOffset);
}
```

Slika 4.4. *Implementacija algoritma za generiranje modela sfere*

Za postavljanje teksture na model sfere, koristi se inverzna Mercatorova projekcija pri čemu se slika pravokutnog oblika projicira na sferu čija je površina zakrivljena. Iako je ovakva projekcija jednostavna za implementirati, glavna mana takve projekcije se očituje pri generiranju vrhova modela gdje količina vrhova na najširem dijelu sfere jednaka na polovima sfere, odnosno, na polovima sfere se u jednoj točki gomila veliki broj vrhova.



Slika 4.5. *Inverzna Mercatorova projekcija*

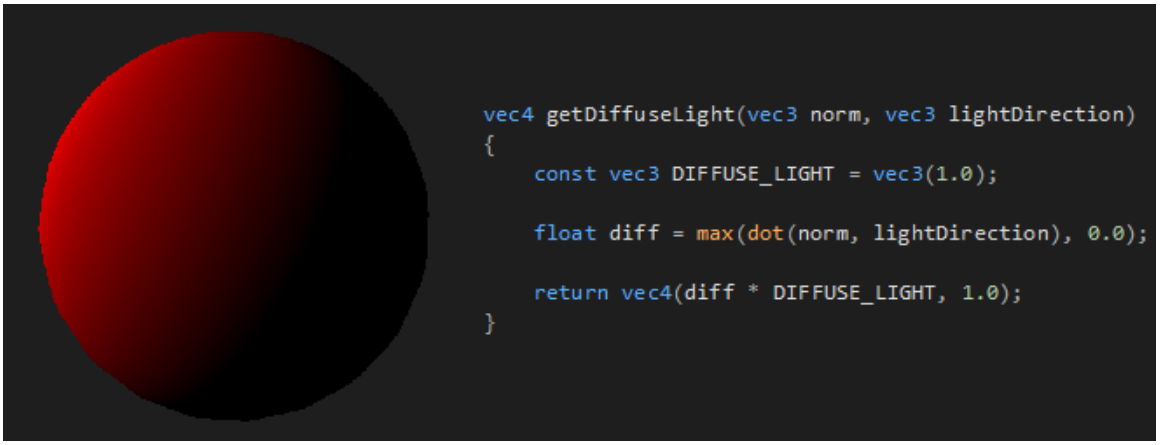
4.3. Phongov model osvjetljenja

Kako bi svaki model asteroida ili planeta bio prikazan na zaslonu kao prirodno osvijetljen, korišten je Phongov model osvjetljenja koji se sastoji od tri komponente: ambijentalno osvjetljenje, difuzno osvjetljenje i spekularno osvjetljenje. Iako se Phongov model osvjetljenja može implementirati u shaderu verteksa, implementiran je u shaderu fragmenata kako bi osvjetljenje izgledalo točnije i prirodnije.

Ambijentalno osvjetljenje (engl. *ambient lighting*) se koristi kako dijelovi modela koji nisu pod nikakvim svijetlom ne bi bili prikazani kompletno crni. Za potrebe ovog projekta koristi se tamnoplava boja.

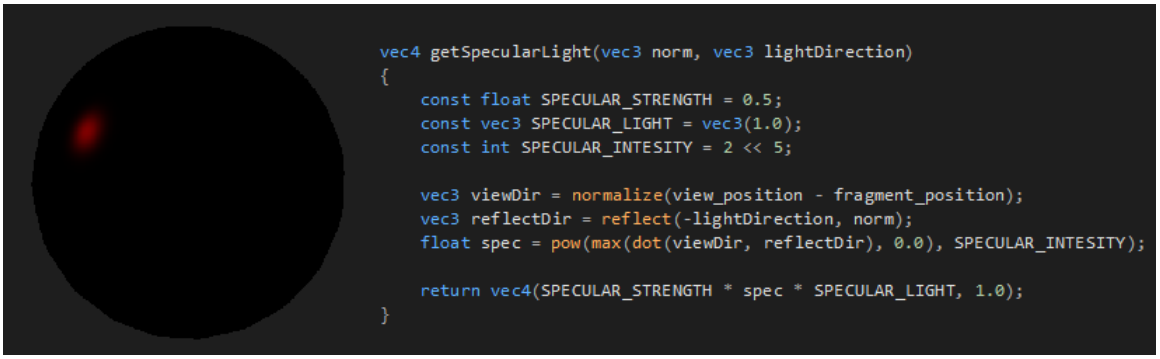
Slika 4.6. *Prikaz i implementacija ambijentalnog osvjetljenja na modelu sfere*

Difuzno osvjetljenje (engl. *diffuse lighting*) daje najprirodnije osvjetljenje objekata. Stranice modela koje su okrenute prema izvoru svjetlosti su najsvjjetlije osjenčane. Stranica će se tamnije prikazati što je više zakrenuta od izvora svjetlosti. Matematički gledano, osvjetljenje stranice modela obrnuto je proporcionalno kutu kojega zatvaraju normala stranice i normala smjera izvora svjetlosti koje je u ovom projektu Sunce koje se nalazi u ishodištu koordinatnog sustava. U shaderu fragmenata se koristi skalarni produkt vektora normale stranice i smjera izvora svjetlosti, te se uzimaju samo vrijednosti veće od nule kako bi rezultat skalarnog produkta bio u intervalu [0.0, 1.0].



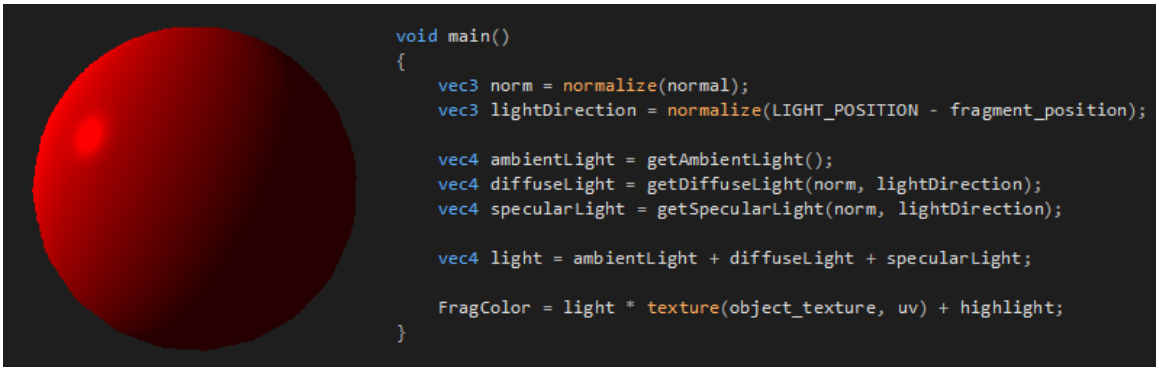
Slika 4.7. *Prikaz i implementacija difuzijskog osvjetljenja na modelu sfere*

Spekularno osvjetljenje (engl. *specular lighting*) se koristi kako bi se dočarala refleksivnost površine stranice modela. Računa se slično difuznom osvjetljenju, ali se umjesto normale izvora svjetlosti uzima normala smjera gledišta virtualne kamere. Isti skalarni produkt se zatim potencira proizvoljnim brojem pri čemu će male vrijednosti potencije prikazivati stranice modela čije površine jako malo svjetlosti reflektiraju.



Slika 4.8. *Prikaz i implementacija spekularnog osvjetljenja na modelu sfere*

Konačno osvjetljenje modela je zbroj ambijentalnog, difuzijskog i spekularnog osvjetljenja za svaki piksel. Za korištenje vrijednosti osvjetljenja, potrebno je pomnožiti vektor osvjetljenja s pikselom teksture za zadani fragment predstavljen također vektorom pri čemu se množenje vrši za svaku komponentu vektora zasebno.



Slika 4.9. *Prikaz i implementacija potpunog Phongovog modela osvjetljenja na modelu sfere.*

4.4. Simulacija gravitacije

Kako bi bilo moguće simulirati prave putanje asteroida kroz planetarni sustavi, potrebno je za svaki planet sustava znati poziciju i njegovu masu, a ovisnost kretanja asteroida o planetima se dobije iz 2. Newtonovog zakona i Newtonovog zakon gravitacije. Prema 2. Newtonovom zakonu, ukupna sila koja djeluje na asteroid je jednaka zbroju Gravitacijskih sila svih planeta koji utječu na asteroid što je zapisano sljedećom jednadžbom:

$$\vec{F}_{uk} = \sum_{i=1}^{n} \vec{F}_{g_i} \tag{4.1.}$$

gdje je:

- $\vec{F}_{uk}$ – ukupna gravitacijska sila na asteroid
- n – ukupan broj planeta u planetarnom sustavu
- $\vec{F}_{g_i}$ – gravitacijska sila $i - tog$ planeta na asteroid

Prema Newtonovom zakonu gravitacije, gravitacijska sila koja djeluje između dva tijela je proporcionalna umnošku masa oba tijela i obrnuto proporcionalna kvadratu njihove udaljenosti što je prikazano sljedećom jednadžbom:

$$\vec{F}_g = G\,\frac{m_1 m_2}{\left|r_{1,2}\right|^2}\,\hat{r}_{1,2} \tag{4.2.}$$

gdje je:

- $\vec{F}_g$ – gravitacijska sila između dva tijela
- G – gravitacijska konstanta
- m_1, m_2 – mase tijela
- $r_{1,2}$ – udaljenost između tijela
- $\hat{r}_{1,2}$ – jedinični vektor između tijela

Kada se jednadžba 4.2. uvrsti u jednadžbu 3.1., dobije se sljedeći izraz:

$$\vec{F}_{uk} = \sum_{i=2}^{n} G\,\frac{m_1 m_i}{\left|r_{1,i}\right|^2}\,\hat{r}_{1,i} \tag{4.3.}$$

Daljnjim raspisivanjem jednadžbe 3.3. moguće je izvesti jednadžbu za trenutnu akceleraciju tijela u ovisnosti o drugim tijelima (4.5).

$$\vec{F}_{uk} = G \cdot m_1 \cdot \sum_{i=2}^{n} \frac{m_i}{\left|r_{1,i}\right|^2}\,\hat{r}_{1,i} \tag{4.4.}$$

$$m_1 \cdot \vec{a}_1 = G \cdot m_1 \cdot \sum_{i=2}^{n} \frac{m_i}{\left|r_{1,i}\right|^2}\,\hat{r}_{1,i}$$

$$\vec{a}_1 = G \cdot \sum_{i=2}^{n} \frac{m_i}{\left|r_{1,i}\right|^2}\,\hat{r}_{1,i} \tag{4.5.}$$

Prilikom implementacije algoritma za dohvaćanje akceleracije tijela uzrokovane gravitacijskom silom ostalih planeta, potrebno je pripaziti da oba tijela ne budu preblizu. U slučaju da je udaljenost dva tijela jako mala, recipročna vrijednost kvadrata njihove udaljenosti postaje vrlo velika što daje netočan rezultat. Stoga, prilikom računanja gravitacijske sile, potrebno je provjeriti da dva tijela nisu preblizu jedan drugom, odnosno da je njihova udaljenost veća od neke male vrijednosti, u ovom slučaju konstante Math::EPSILON koja predstavlja vrlo malu vrijednost.

```
Vector SolarSystem::getGravitationalForceFor(Asteroid* asteroid)
{
    const float GRAVITATIONAL_CONSTANT = 0.001f;

    Vector totalForce;

    for (const auto planet : this->planets)
    {
        Vector direction = planet->getPosition() - asteroid->getPosition();
        float distanceSquared = direction.length2();
        direction.normalize();

        if (distanceSquared < Math::EPSILON) continue;

        Vector force = direction * planet->getMass() / distanceSquared;
        totalForce += force;
    }

    return totalForce * GRAVITATIONAL_CONSTANT;
}
```

Slika 4.10. *Implementacija algoritma za dohvaćanje akceleracije tijela uzrokovane gravitacijskom silom*

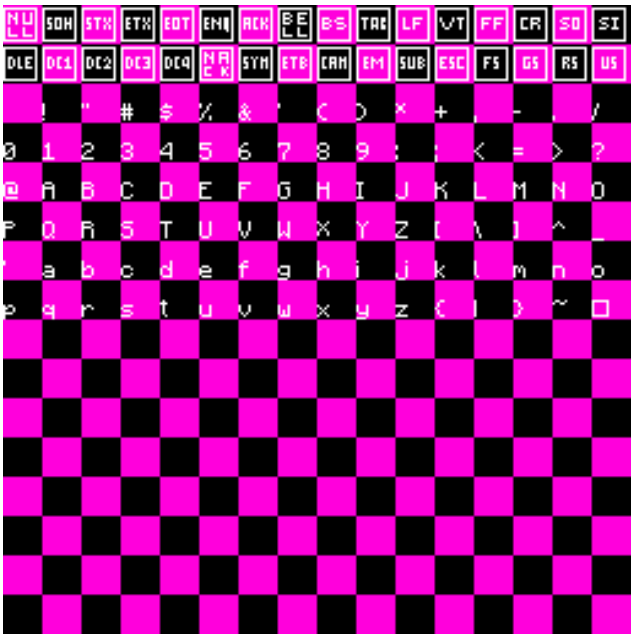
4.5. Grafičko korisničko sučelje

Za potrebe ovog projekta, grafičko korisničko sučelje (engl. Graphical User Interface) se sastoji od dvaju glavnih elemenata: teksta i gumbova. Prikazivanje teksta i gumbova na zaslon zahtjeva sličnu implementaciju – prikazivanje određene slike na zaslon na željenom položaju. Sve tehnike prikazivanja teksta na zaslon zahtijevaju prikazivanje svakog slova zasebno te se dijele u dvije kategorije: tehnike prikazivanja teksta vektorskog formata i tehnike prikazivanja teksta rasteriziranog formata .

Za prikazivanje teksta vektorskim formatom, potreban je set matematičkih krivulja koje definiraju oblik slova. Ovim načinom se može prikazati tekst bilo koje veličine bez gubitka oštrine teksture slova jer se prethodno ne definira gotova tekstura, već se tekstura generira po potrebi.

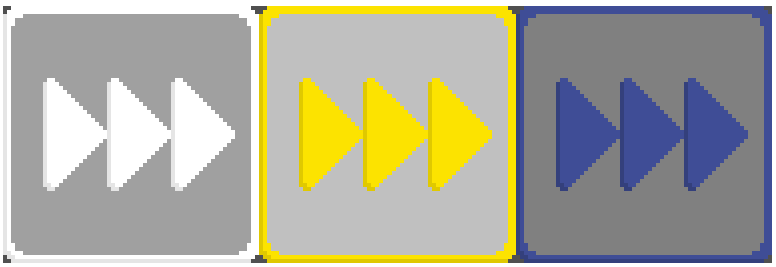
Prikazivanje teksta rasteriziranog formata zahtjeva generiranje tekstura pojedinačnog slova unaprijed čime se dobiva tekstura nepromjenjive rezolucije. Iako je ova tehnika lagana za implementirati te ne zahtjevu veću količinu resursa za prikazivanje teksta, ona zahtjeva generiranje i memoriranje teksture svakog seta slova koji će se koristiti što može zahtijevati velike količine radne memorije te zahtjeva generiranje više različitih veličina tekstura slova kako bi se tekst velikog fonta mogao prikazati dovoljno dobrom oštrinom. Također, pri korištenju ove tehnike implementira se atlas tekstura (engl. texture atlas) – velika tekstura koja sadrži teksture svih slova koje se mapiraju sa slovima teksta kojima odgovaraju što ubrzava postupak prikazivanja teksta na zaslon te zahtjeva manje memorije pri implementaciji. Tekst se u korisničkom sučelju koristi za

ispisivanje imena označenog asteroida ili planeta te za prikazivanje uputa o dodavanju nasumičnih planeta i asteroida.



Slika 4.11. *Atlas tekstura slova korišten za ovaj projekt. Ružičasto-crna pozadina je dodana radi lakše čitljivosti bijelog fonta slova.*

Prikaz gumbova na zaslonu je jednostavniji jer zahtijeva prikaz jedne teksture. Gumbi za ovaj projekt se koriste samo za podešavanje brzine simulacije te se svaki gumb nalazi u jednom od tri stanja: normalno stanje (engl. normal), stanje pri kojem se pokazivač miša nalazi iznad gumba (engl. hovered) i stanje pri kojem je gumb odabran (engl. selected). Ovisno o stanju u kojem se nalazi, gumb prikazuje različitu teksturu. Normalno stanje je prikazano teksturom s bijelim naglascima, stanje pri kojem pokazivač miša se nalazi iznad gumba je prikazano žutom bojom, a odabran gumb je prikazan plavom bojom. Pomoću ovih gumbova, simulaciju je moguće zaustaviti te postaviti normalnu ili veću brzinu simuliranja.

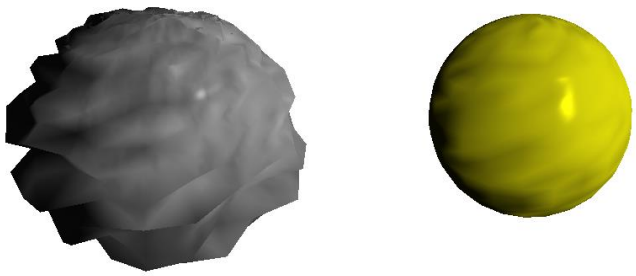


Slika 4.12. *Teksture svih stanja gumba za postavljanje brzine simulacije na trostruko veću vrijednost.*

4.6. Interakcija s objektima prikazanih na zaslonu

Jedna od glavnih interakcija korisnika sa simulacijom planetarnog sustava je odabir, dodavanje i brisanje planeta i asteroida. Kako se simulacija odvija u trodimenzionalnom okruženju, odabir planeta zahtjeva transformaciju koordinata pokazivača na zaslonu u koordinate virtualnog okruženja te broj mogućnosti pozicija novo dodanog objekta postaje prevelik.

Dodavanje planeta i asteroida u planetarni sustav je omogućeno pomoću tipkovnice. Pritiskom na tipku slova *I*, generira se planet čiji su ime, veličina, masa i boja nasumično generirani dok je model planeta model sfere čiji je parametar hrapavosti postavljen na 10. Udaljenost planeta od središnje zvijezde te njegov položaj na orbiti planeta su odabrani nasumično. Pritiskom na tipku slova *O*, generira se asteroid čiji su veličina i parametri modela sfera nasumično odabrani. Položaj asteroida se također nasumično odabire neovisno o položaju drugih planeta i asteroida. Generiranje modela asteroida i planeta uvijek generira nasumični model sfere čime niti jedan asteroid niti planet nemaju isti model, odnosno nemaju isti izgled.



Slika 4.13. *Izgled nasumično generiranih modela asteroida (sivi model) i planeta (žuti model)*

Odabir asteroida i planeta pokazivačem miša se sastoji od više koraka pri čemu svaki korak vrši neku vrstu transformacije koordinata pokazivača miša. Za transformaciju koordinata pokazivača u koordinate simulacije pri odabiru objekata, koriste se inverzi matrica modela, pogleda i projekcije. Prije transformacije, koordinate pokazivača na zaslonu je potrebno normalizirati, odnosno prebaciti iz koordinatnog sustava relativnog gornjem lijevom kutu zaslona u koordinatni sustav koji je relativan središtu zaslona pri čemu obje koordinate pokazivača miša pripadaju intervalu $[-1, 1]$. Množeći normalizirane koordinate pokazivača inverzom matrice projekcije se dobije pozicija pokazivača u koordinatnom sustavu relativnom kameri, a zatim množeći inverzom matrice pogleda kamere dobijemo stvarne koordinate u 3D okruženju odnosno

smjer gledišta virtualne kamere koji se dalje koristi za provjeravanje koji se asteroid ili planet nalazio iza pokazivača miša.

```
void MouseSelector::update()
{
    Vector clipCoords = Vector(Cursor::getNormalizedX(), Cursor::getNormalizedY(), -1.0f, 1.0f);
    Vector eyeCoords = toEyeCoords(clipCoords);
    Vector worldCoords = toWorldCoords(eyeCoords);

    currentRay = worldCoords;
}

Vector MouseSelector::toEyeCoords(Vector clipCoords)
{
    Matrix invertedProjection = Camera::getActiveCamera().getProjectionMatrix();
    invertedProjection.invert();
    Vector eyeCoords = invertedProjection * clipCoords;

    return Vector(eyeCoords.x, eyeCoords.y, -1.0f, 0.0f);
}

Vector MouseSelector::toWorldCoords(Vector eyeCoords)
{
    Matrix invertedView = Camera::getActiveCamera().getViewMatrix();
    invertedView.invert();
    Vector worldCoords = invertedView * eyeCoords;
    worldCoords.normalize();

    return worldCoords;
}
```

Slika 4.14. *Implementacija algoritma za dobivanje smjera gledišta virtualne kamere iz pozicije pokazivača miša na prozoru.*

Kako bi se provjerilo je li pokazivač miša zapravo bio iznad nekog objekta na zaslonu, potrebno je izvršiti provjeru nad svakim asteroidom i planetom. Provjera je ekvivalentna traženju sjecišta pravca, odnosno smjera gledišta virtualne kamere i sfere u trodimenzionalnom prostoru čiji je radijus jednak asteroidu ili planetu za kojega se vrši provjera.

Parametarska jednadžba pravca u trodimenzionalnom prostoru se pomoću vektora zapisuje na sljedeći način:

$$L(t) = P + tU \tag{4.6.}$$

gdje je:

- $L(t)$ – točka na pravcu
- P – proizvoljna točka (ishodište)
- U – jedinični vektor smjera pravca

Pri traženju odabranog objekta, proizvoljna točka pravca je položaj virtualne kamere, a smjer pravca odgovara smjeru gledišta virtualne kamere.

Vektorska jednadžba sfere u trodimenzionalnom prostoru se zapisuje na sljedeći način:

$$\|X - C\|^2 = r^2 \tag{4.7.}$$

gdje je:

- X – proizvoljna točka u prostoru
- C – središte sfere
- r – radijus sfere

Središte i radijus sfere pri određivanju odabranog planeta su položaj i radijus asteroida ili planeta. Za izvršavanje provjere sijeku li se pravac i sfera, odnosno jeli smjer gledišta virtualne kamere usmjeren prema asteroidu ili planetu, potrebno je uvrstiti jednadžbu 4.6. u jednadžbu 4.7. čime se dobije kvadratna jednadžba čija je nepoznanica označena slovom t.

$$(P + tU - C) \cdot (P + tU - C) = r^2$$

$$P \cdot P + P \cdot Ut - P \cdot C + P \cdot Ut + (Ut)^2 - C \cdot Ut - C \cdot P - C \cdot Ut + C^2 = r^2$$

$$(U \cdot U)t^2 + 2U \cdot (P - C)t + (P - C) \cdot (P - C) - r^2 = 0 \tag{4.8.}$$

Ako postoji realan broj t koji zadovoljava jednadžbu 4.8., odnosno, ako je determinanta kvadratne jednadžbe veća ili jednaka nuli, tada se pokazivač miša nalazi ispred promatranog asteroida ili planeta.

```
bool MouseSelector::isObjectSelected(const ISelectable* object) const
{
    if (object == nullptr) return false;

    Vector center = object->getPosition();
    float radius = object->getRadius();
    Vector point = Camera::getActiveCamera().getPosition();
    Vector unit = this->currentRay;

    Vector q = point - center;
    float a = Vector::DotProduct(unit, unit);
    float b = 2 * Vector::DotProduct(unit, q);
    float c = Vector::DotProduct(q, q) - radius * radius;
    float d = b * b - 4 * a * c;

    return d >= 0;
}
```

Slika 4.15. *Implementacija algoritma koji provjerava nalazi li se pokazivač miša iznad asteroida ili planeta*

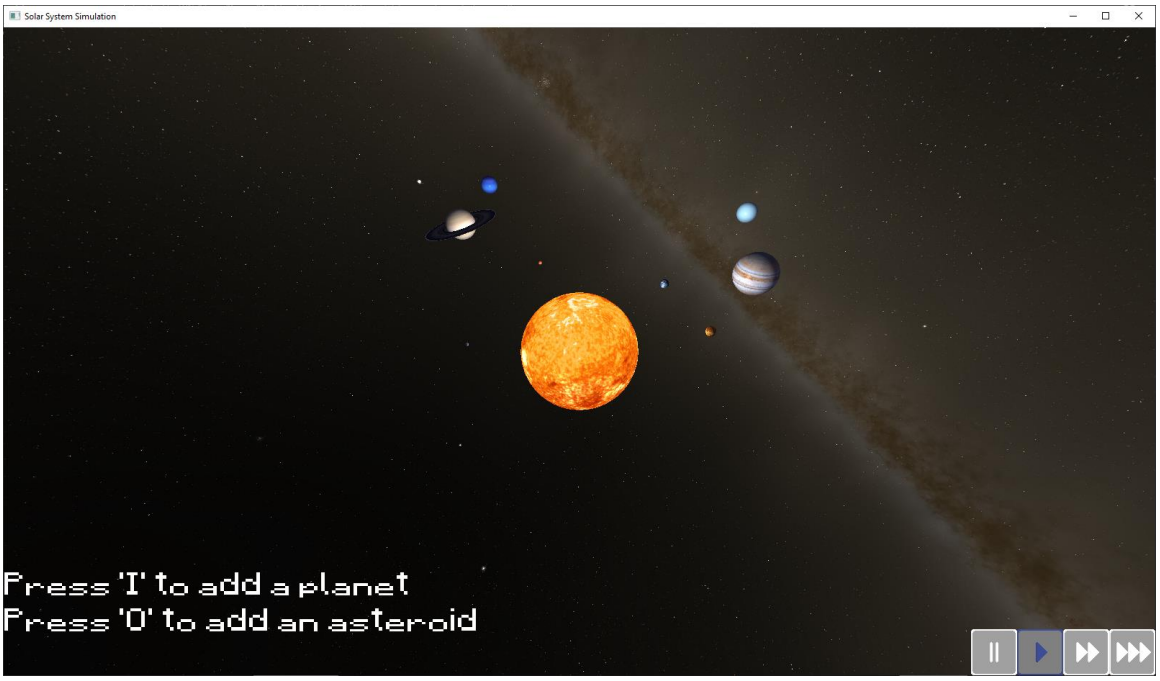
Brisanje asteroida, planeta i planetarnog sustava je implementirano na način da korisnik prvo pritiskom lijevim gumbom miša označi na objekt koji želi obrisati te na isti opet pritisne desnim gumbom miša. Time planetarni sustav briše asteroid ili planet iz popisa svih asteroida i planeta te se njegovo ime koje je prikazano na zaslonu također briše.

```
void MouseSelector::onLeftClick()
{
    if (selectedObject != nullptr)
    {
        selectedObject->deselect();
        selectedObject = nullptr;
    }

    for (const auto& planet : solarSystem.getPlanets())
    {
        if (isObjectSelected(planet))
        {
            planet->select();
            selectedObject = planet;
            return;
        }
    }

    for (const auto& asteroid : solarSystem.getAsteroids())
    {
        if (isObjectSelected(asteroid))
        {
            asteroid->select();
            selectedObject = asteroid;
            return;
        }
    }
}

void MouseSelector::onRightClick()
{
    if (isObjectSelected(selectedObject))
    {
        if (Planet* planet = dynamic_cast<Planet*>(selectedObject))
        {
            solarSystem.removePlanet(planet);
        }
        else if (Asteroid* asteroid = dynamic_cast<Asteroid*>(selectedObject))
        {
            solarSystem.removeAsteroid(asteroid);
        }

        selectedObject = nullptr;
    }
}
```

Slika 4.16. *Implementacija algoritma za brisanje asteroida i planeta iz planetarnog sustava.*

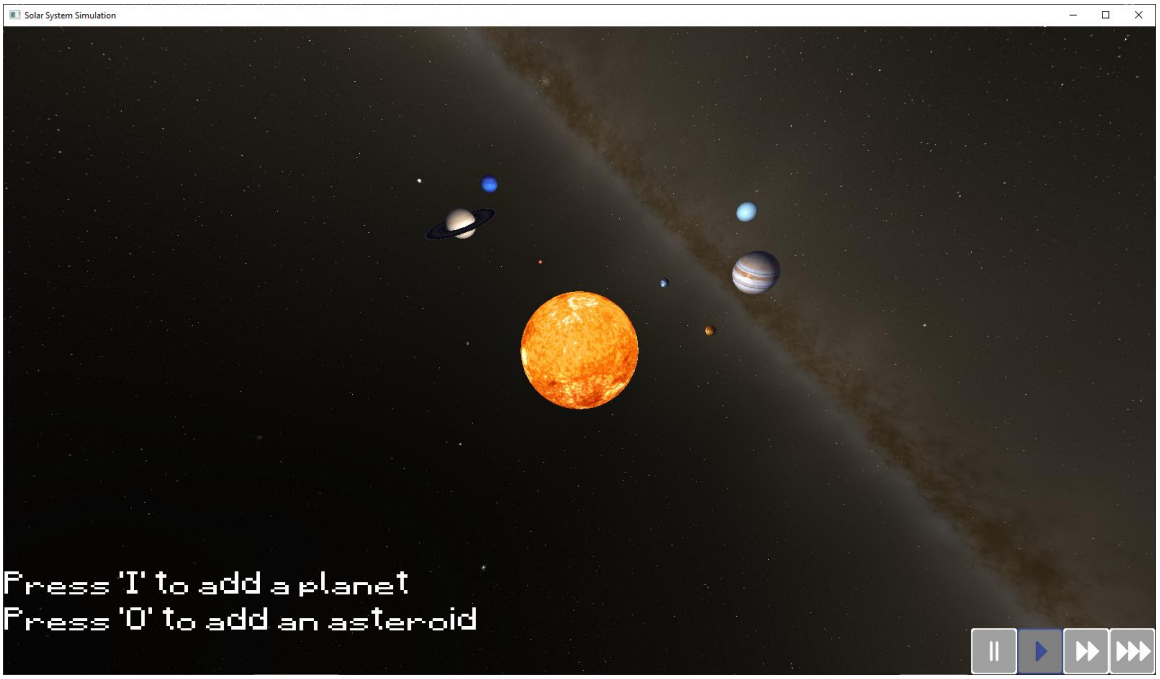
5. IZVEDBA PROGRAMA

Koristeći tehnike generiranja, translacije i projekcije modela te prikazivanje jednostavnog grafičkog korisničkog sučelja, moguće je napraviti simulaciju planetarnog sustava s varijabilnim brojem planeta i asteroida unutar sustava. Za bolji i realističniji prikaz planetarnog sustava u svemiru, korištena je pozadina sa slikom galaksija i zvijezda u svemiru metodom prikazivanja kocke (engl. cube mapping). Pozadina je kreiranja pomoću modela kocke čije stranice gledaju prema virtualnoj kameri. Virtualna kamera se uvijek nalazi unutar te kocke čime se dobiva iluzija realistične pozadine.



Slika 5.1. *Prikaz planetarnog sustava s galaksijama i zvijezdama u pozadini*

Korisniku je također omogućeno kretanje kroz planetarni sustav vrtnjom oko središnje zvijezde, ili u ovom slučaju, oko Sunca koristeći miš i tipke na mišu. Kretanjem kroz planetarni sustav, moguće je detaljno vidjeti površinu planeta i asteroida čiji su modeli sfere s neravnom površinom. Kretanje je također implementirano na način da se nikada ne zaustavi isti trenutak, već se kretanje polako usporava što daje realističnije kretanje kroz planetarni sustav. Prilikom označivanja planeta ili asteroida, na zaslonu se ispisuje ime označenog objekta kojeg se kasnije može ukloniti iz sustava.



Slika 5.2. *Primjer nasumično generiranog planetarnog sustava.*

Slike koje se koriste kao teksture planeta i središnje zvijezde su preuzete sa stranice Solar System Scope [3], a pozadinska slika s foruma Planetary Annihilation [13].

Slika 5.3. *Primjer modela Sunčevog sustava*

6. ZAKLJUČAK

U završnom radu izrađena je aplikacija za prikaz simulacije planetarnog sustava te je korisniku omogućeno kretanje kroz planetarni sustav, dodavanje, brisanje asteroida i planeta. Za kreiranje prozora i korištenje ulaza s vanjskih uređaja korišten je GLFW, a za učitavanje funkcija definiranih OpenGL API-jem za razvoj 3D okruženja je korišten GLEW. Kako je OpenGL API namijenjen nižim programskim jezicima, aplikacija je razvijena C++ programskim jezikom. Za strukturiranje i izradu projekta je korišten Microsoft Visual Studio 2019. Prikazivanje planetarnog sustava je zahtijevalo napredne vještine i dobro poznavanje linearne algebre, a za simulaciju poznavanje osnovnih zakona klasične mehanike iz fizike. Iako nije moguće koristiti prave odnose veličina masa i udaljenosti planeta, korištene veličine daju reprezentativan odnos veličina za vizualno dobar prikaz sustava.

Aplikaciju je moguće unaprijediti koristeći kompleksnije, ali i realističnije metode osvjetljivanja poput dodavanja sjena na planete i asteroide koji se nalaze iza drugih planeta i asteroida te dodavajući materijale, odnosno svojstava modela kojim je moguće za svaki model individualno postaviti svojstva komponenata Phongovog modela osvjetljavanja. Simuliranje putanja asteroida se dodatno može poboljšati koristeći realne vrijednosti za mase i udaljenosti planeta i asteroida te dodavanjem provjere za kolizije između asteroida koje bi utjecale na daljnju putanju asteroida.

LITERATURA

[1] Mass Effect 2 [online], Electronic Arts, dostupno na: https://www.ea.com/games/mass-effect/mass-effect-2 [09.09.2021.]

[2] Universe Sandbox [online], Giant Army, dostupno na: https://universesandbox.com [08.09.2021.]

[3] Solar System Scope [online], dostupno na: https://www.solarsystemscope.com/textures/ [04.04.2021.]

[4] Microsoft Visual Studio [online], Microsoft, dostupno na: https://visualstudio.microsoft.com/ [08.03.2021.]

[5] OpenGL [online], The Khronos Group, dostupno na: https://www.opengl.org/ [10.03.2021.]

[6] Learn OpenGL [online], Joey de Vries, dostupno na: https://learnopengl.com/ [08.08.2021.]

[7] Scratchpixel 2.0 [online], dostupno na: https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/rasterization-stage [30.08.2021.]

[8] The OpenGL Extension Wrangler Library [online], dostupno na: http://glew.sourceforge.net/ [11.03.2021.]

[9] GLFW [online], dostupno na: https://www.glfw.org/ [11.03.2021.]

[10] STB Image, GitHub [online], Sean Barret, dostupno na: https://github.com/nothings/stb [23.07.2021.]

[11] Painet.NET [online], Rick Brewster, dostupno na: https://www.getpaint.net/ [05.05.2021.]

[12] Researcg Gate [online], dostupno na: https://www.researchgate.net/figure/the-perspective-volume-in-OpenGL_fig8_283255927 [01.09.2021.]

[13] Planet Annihilation [online], dostupno na: https://forums.planetaryannihilation.com/threads/milky-way-skybox.72682/ [02.05.2021.]

SAŽETAK

Napretkom tehnologije u istraživanju svemira, sve je veća potreba za simulacijom planetarnih sustava, odnosno putanja planeta i asteroida. U završnom radu se proučava problem simulacija putanja svemirskih tijela te su navedeni postojeći programi i video igre koji implementiraju takve simulacije. Rad opisuje izradu aplikacije za simuliranje planetarnog sustava koristeći API za 3D grafiku OpenGL te knjižnice GLEW i GLFW za učitavanje funkcija OpenGL-a u memoriju i kreiranje prozora za aplikaciju. Koristeći ulaze s miša, korisniku je omogućena navigacija kroz simulacije i brisanje asteroida i planeta, dok je korištenjem tipkovnice omogućeno dodavanje novih asteroida i planeta. Primjenom osnova linearne algebre i klasične mehanike iz fizike, napravljena je aplikacija koja korisniku omogućava simuliranje i uređivanje planetarnog sustava te vizualizira kretanje planeta i asteroida.

Ključne riječi: 3D, asteroid, OpenGL, planet, simulacija

ABSTRACT

With the advancement of space exploration, there is a greater need for simulating planetary systems, that is simulating trajectories of planets and asteroids. The problem of simulating trajectories of space bodies were studied in this paper, while existing programs and video games were also shown and described. This paper describes the development of an application for simulating planetary systems using OpenGL for the 3D graphics API and using GLEW and GLFW for loading OpenGL functions into the memory and creating a window for that application. The user is able to navigate through the simulation and delete asteroids and planets using a mouse while using the keyboard, the user is able to add new planets and asteroids. Applying the basics of linear algebra and classical mechanics in physics, an application that allows the user to simulate and edit a planetary system and visualize the movement of planets and asteroids was created.

Keywords: 3D, asteroid, OpenGL, planet, simulation

PRILOZI (na DVD-u)

Prilog 1: Završni rad u docx i pdf formatu

Prilog 2: Microsoft Visual Studio projekt programa za simuliranje planetarnog sustava