VEGOVA | ELEKTROTEHNIŠKO-RAČUNALNIŠKA STROKOVNA ŠOLA IN GIMNAZIJA LJUBLJANA

VSSR

VESOLJSKI SPOPAD SUPER RAKET

Strokovno poročilo

Mentorica: Nataša Makarovič, prof.

Avtor: Jakob Jeraj

Logatec, april 2025

POVZETEK

Poročilo vsebuje opise in razlage principov in orodij, ki sem jih uporabil pri ustvarjanju računalniške igre VSSR. Računalniška igra je napisana v jeziku C++, za risanje pa uporablja tehnologijo OpenGL. Poleg igre je napisana še strežniška aplikacija, ki omogoči igranje več uporabnikom hkrati. Strežniška aplikacija deluje na podlagi vtičnikov.

KLJUČNE BESEDE

3D-igra, vesolje, C++, OpenGl, matrika, GJK,

ABSTRACT

The report contains descriptions and explanations of the principles and tools that I used in creating the computer game VSSR. The computer game is written in C++ and uses OpenGL technology for drawing. In addition to the game, a server application has been written that allows multiple users to play at the same time. The server application works based on sockets

KEY WORDS

3D-game, space, C++, OpenGL, matrix, GJK

KAZALO VSEBINE

KAZALO GRADIVA

1. UVOD

Računalniške igre so že od svojega začetka privabljale veliko pozornosti. Igre privabljajo igralce vseh generacij, še posebej mlajše. Mnogo posameznikov preživi ob igranju iger veliko prostega časa. Zelo malo pa je takih igralcev, ki se vprašajo, kaj se dejansko dogaja v računalniku, medem ko uživajo v igri.

Poročilo o izdelavi računalniške igre Vesoljski spopad super raket (VSSR) opisuje tehnologije, algoritme in matematične pristope, ki so potrebni za izdelavo preproste igre.

Računalniška igra Vesoljski spopad super raket je preprosta 3D strelska igra, ki podpira istočasno igranje več igralcev. Cilj igre je poraziti vesoljsko ladjo nasprotnega igralca in se izogniti vsem izstrelkom drugih igralcev. Ko je igralčeva vesoljska ladja poražena, mora počakati nekaj trenutkov, da se lahko zopet vrne v boj. Ob igri je tudi strežniški program, na katerega se lahko igralci povežejo, da igrajo skupaj.

2. METODOLOGIJA

2.1. UPORABLJENE TEHNOLOGIJE

Za izdelavo igre sem uporabil programski jezik C++, ki je trenutno eden izmed najbolj priljubljenih programskih jezikov. Priljubljen je zaradi svoje zmogljivosti, učinkovitosti in vsesplošne uporabnosti.

C++ program se ne more izvajati neposredno iz datoteke, v kateri je algoritem zapisan, temveč ga je treba prevesti v izvršilno datoteko. Ker projekt obsega več kot štirideset datotek in ker ima v skupnem seštevku okoli pet tisoč vrstic kode, sem za prevajanje projekta uporabil *CMake* tehnologijo. CMake je programska oprema, ki omogoča grajenje preprostih in zelo zapletenih programskih sistemov. CMake je podprt na večini platform, kar mi je omogočilo, da igra Vesoljski spopad super raket, deluje na operacijskem sistemu Windows in Linux.

Predpogoj za izdelavo igre je odpiranje okna v operacijskem sistemu in zajem podatkov iz tastature in računalniške miške. Za zgoraj navedene funkcionalnosti sem uporabil GLFW knjižnico. GLFW omogoča odpiranje oken, poskrbi za vhodne podatke iz tipkovnice, miške … in nudi podporo za grafične API-je kot so Vulkan, OpenGL itd.

OpenGL je eden izmed najbolj uporabljenih grafičnih API-jev in OpenGL je neodvisen od okenskih in operacijskih sistemov ter omogoča risanje objektov na zaslon in komuniciranje z grafično kartico v računalniku.

Igra Vesoljski spopad super raket omogoča tudi skupinsko igranje več igralcev. Da sem to lahko implementiral, sem uporabil sistem vtičnic. Vtičnice (*ang. socket*) omogočajo komunikacijo med omrežnimi napravami.

2.2. ALGORITMI IN MATEMATIČNI PRISTOPI

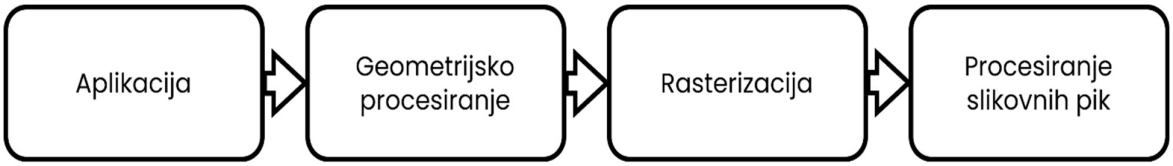
Računalniški zaslon lahko brez težav prikaže 2D grafiko. Težava nastopi, ko želimo na njem upodobiti 3D objekte. Ker računalniški zaslon nima globine, je potrebno 3D objekte prikazati s pomočjo iluzije. Če v resničnem svetu nekaj opazujemo, lahko ugotovimo, da stvari, ki so od nas bolj oddaljene, izgledajo manjše in da stvari, ki so nam bolj približane, izgledajo večje. Ta pojav

imenujemo perspektiva. Ta isti efekt skuša oponašati perspektivna projekcija. Ta način prikazovanja 3D objektov na 2D zaslonu sem uporabil v igri.

Med igranjem igre se lahko zgodi, da se igralčeva raketa kam zaleti. Za normalen in predviden potek igre je potrebno ta trk razrešiti. Za ugotavljanje morebitnih trkov med objekti sem uporabil algoritem GJK.

2.3. STANDARDNI GRAFIČNI CEVOVOD

Standardni grafični cevovod je orodje, katerega glavna naloga je upodabljanje 2D slik, 3D objektov in svetlobnih virov. Avtorji knjige *Real-time Rendering, Fourth Edition (Upodabljanje v realnem času)* navajajo: »Grafični cevovod tako predstavlja temeljno orodje, ki omogoča upodabljanje v realnem času« (Akenine-Möller et al., 2018, str. 11). Grafični cevovod je sestavljen iz več delov. Osnovna konstrukcija je sestavljena iz štirih faz. Vsaka izmed teh faz pa je lahko cevovod sama po sebi.



Slika 1: Sestavni deli standardnega grafičnega cevovoda (prirejeno po Akenine-Möller et al., 2018, str. 12).

2.3.1. Aplikacijska faza

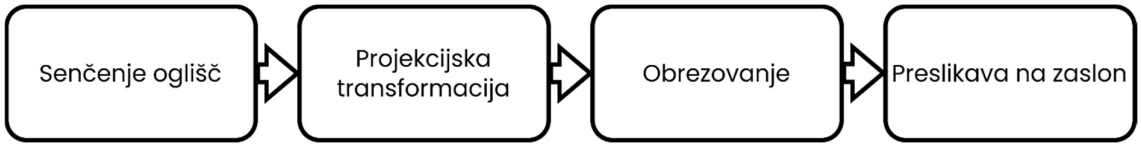
Aplikacijska faza se izvaja na CPE. V tej fazi ima razvijalec popoln nadzor nad potekom programa. Tukaj lahko tudi izvaja različne optimizacije. Ena izmed možnih optimizacij je zmanjševanje števila trikotnikov, ki jih je potrebno narisati.

Slika 2: Prikaz možne optimizacije števila trikotnikov (lasten vir).

V tej fazi sta pogosto implementirana zaznava in razreševanje trkov med objekti. V aplikacijski fazi se obdela tudi vhodne podatke. Ob koncu aplikacijske faze se podatke o objektih, ki jih je potrebno narisati, pošlje naslednji fazi grafičnega cevovoda.

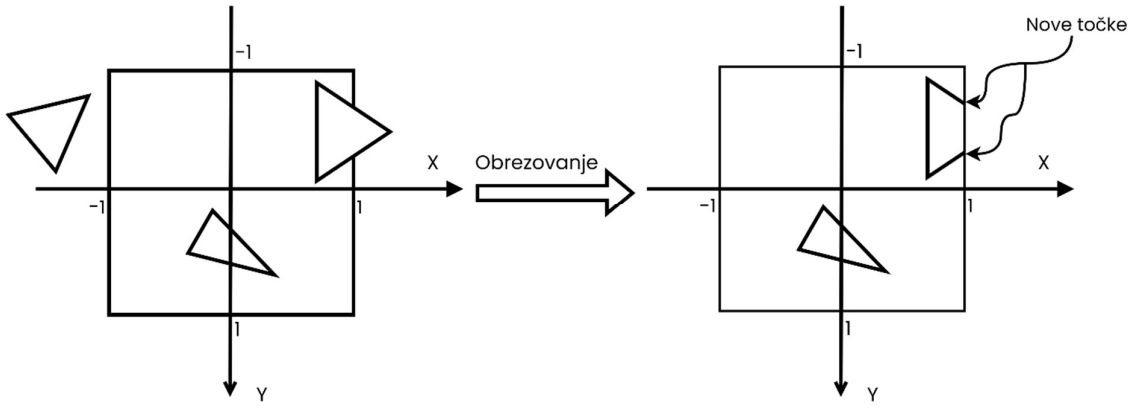
2.3.2. Geometrijsko procesiranje

Faza geometrijskega procesiranja je odgovorna za delo s trikotniki in oglišči. V okolju OpenGL se to imenuje *vertex shading*. Faza geometrijskega procesiranja je dodatno razdeljena na štiri podfaze.



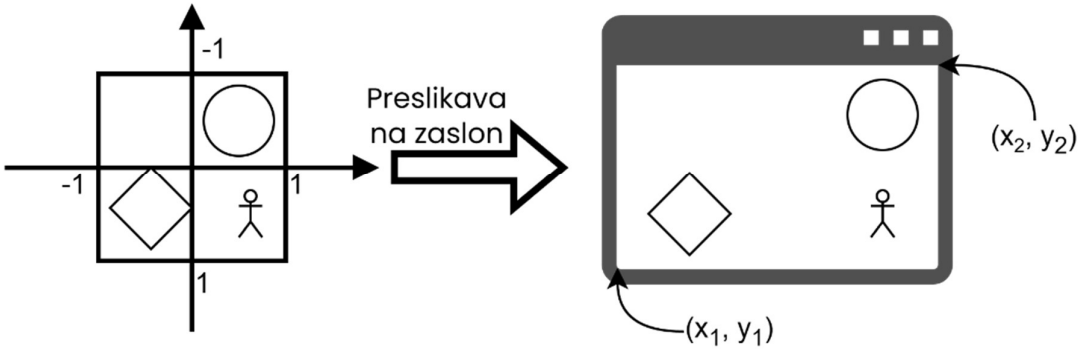
Slika 3: Prikaz podfaz faze geometrijskega procesiranja (prirejeno po Akenine-Möller et al., 2018, str. 14).

V prvi podfazi se nastavlja podatke, ki so povezani z oglišči. Ti podatki so barva, normale za vsako točko nekega objekta in podobno. V drugi pod fazi se a oglišča premika, s čimer dosežemo, da se objekt v 3D prostoru premakne. Nato sledi obrezovanje. Možno je, da objekti niso v celoti narisani, saj njihovi deli segajo izven navideznega prostora, katerega dimenzije segajo za 1 v vsako smer. Ta prostor se imenuje enotski prostor in objekte, ki segajo izven njega, je potrebno obrezati.



Slika 4: Prikaz obrezovanja in enotskega prostora (prirejeno po Akenine-Möller et al., 2018, str. 20).

Zadnji korak geometrijskega procesiranje je preslikava na zaslon. V tem koraku se koordinate preslikajo iz enotskega prostora v okno aplikacije, katerega koordinate se raztezajo od $x_1\ do\ x_2\ kjer\ je\ x_1 < x_2$ in podobno velja za y.



Slika 5 Preslikava na zaslon (prirejeno po Real-Time Rendering, Akenine-Möller et al., 2018, str. 20).

2.3.3. Rasterizacija

Rasterizacija je proces, kjer se ugotavlja katere slikovne pike na zaslonu so znotraj primitive, ki jo izrisuje program. Torej se v tem koraku ugotovi, katere slikovne pike pripadajo določenemu trikotniku. Proces je rasterizacije povezuje geometrijsko obdelavo s procesiranjem slikovnih pik.

2.3.4. Procesiranje slikovnih pik

V fazi procesiranja slikovnih pik je že znano, katere slikovne pike so znotraj trikotnika, ki se ga izrisuje. V tej fazi se slikovnim pikam nastavlja barvo ali teksture. Programer napiše program, v katerem določi pravila, na kakšen način se bodo slikovne pike obarvale. V okolju OpenGL se ta program imenuje *fragment shader*.

2.4. TRANSFORMACIJE

V prejšnjem poglavju sem opisoval, po katerih korakih računalnik na zaslon izriše nek objekt. Eden izmed korakov je geometrijsko procesiranje. V tem koraku se točke nekega objekta rotirajo, vrtijo in premikajo. S tem dosežemo, da se obdelovani objekt premakne in ali zavrti. Vsaka točka določenega objekta je v 3D igri opisana kot trirazsežni vektor. Ta se kasneje tekom procesiranja v OpenGL v *vertex shaderju* pretvori v štirirazsežni vektor, ki ga lahko s pomočjo matrik vrtimo in premikamo.

2.4.1. Vektor

Vektor je geometrijski objekt, ki ima smer in dolžino, ne vemo pa njegovega natančnega položaja v prostoru. V svoji igri sem vektor opisal s pomočjo treh koordinat. Implementacija izgleda tako:

```
class vec3
   {
   public:
       float x, y, z;
       vec3();
       vec3(float t_x);
       vec3(float t_x, float t_y, float t_z);
       vec3(const vec2 &t_x, float t_z);

       vec3 operator+(const vec3 &t) const;
       vec3 operator+(float t) const;
       vec3 operator-(const vec3 &t) const;
       vec3 operator-(float t) const;
       vec3 operator*(float t);
       vec3 operator*(const mat3 &t) const;
       vec3 operator*(const mat4 &t) const;
       bool operator==(const vec3 &t);
       bool operator!=(const vec3 &t);
       vec3 &operator+=(const vec3 &t);
       vec3 &operator+=(float t);
       vec3 &operator-=(const vec3 &t);
       vec3 &operator-=(float t);
       friend std::ostream &operator<<(std::ostream &os, const vec3 &t);

       float dolzina();
       vec3 normaliziraj();

   private:
   };
```

S pomočjo teh koordinat je mogoče izračunati ostale lastnosti vektorja. Dodal sem tudi metode za seštevanje, odštevanje, množenje, normalizacijo in dolžino.

Dolžino vektorja $\vec{a}$ se izračuna s Pitagorovim izrekom na način:

$$\|a\| = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

Normalizacija vektorja je spreminjaje vektorja tako, da je njegova dolžina enaka 1. Vektor katerega dolžina je 1, se imenuje enotski vektor in se uporablja za shranjevaje smeri. Izračuna se ga tako, da se vsako komponento vektorja deli z njegovo dolžino:

$$\hat{a} = \frac{a}{\|a\|} = (\frac{a_x}{\|a\|}, \frac{a_y}{\|a\|}, \frac{a_z}{\|a\|})$$

Pri vektorjih nastopata dva načina množenja, prvi je skalarni, ki vrne število, imenovano skalar. Skalarni produkt je definiran kot produkt dolžin dveh vektorjev in kota med njima ali pa kot seštevek zmnožkov posameznih komponent:

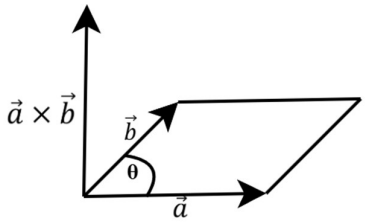
$$a \cdot b = \|a\|\|b\| \cos\theta = a_x b_x + a_y b_y + a_z b_z$$

S skalarjem, ki ga dobimo iz skalarnega produkta, se da izvedeti medsebojno lego dveh vektorjev. Če je skalarni produkt enak 0, pomeni, da sta vektorja pravokotna eden na drugega. V knjigi *Matematični priročnik z osnovani računalništva* avtor pojasnjuje: »Skalarni produkt dveh istosmernih vektorjev $\vec{a}$ in $\vec{b}$ je enak produktu dolžin obeh vektorjev ($\theta = 180^\circ, \cos 180^\circ = 1$), torej $\vec{a} \cdot \vec{b} = -\|a\|\|b\|$, … , Skalarni produkt nasproti usmerjenih vektorjev je enak negativnemu produktu dolžin vektorjev $\vec{a}$ in $\vec{b}$ ($\theta = 180^\circ, \cos 180^\circ = -1$), torej $\vec{a} \cdot \vec{b} = -\|a\|\|b\|$« (Stöcker, 2018, str. 297)

Druga vrsta množenja vektorjev je vektorski produkt. Vektorski produkt je definiran v trirazsežnem prostoru in rezultat te operacije je vektor. Definiran je na način:

$$\vec{c} = \vec{a} \times \vec{b} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix}$$

Vektor, ki nastane pri vektorskem množenju je pravokoten na ravnino, ki jo tvorita vektorja $\vec{a}$ in $\vec{b}$. Njegova velikost je enaka ploščini paralelograma, ki ga tvorita vektorja $\vec{a}$ in $\vec{b}$. »Vektorski produkt je največji, ko sta vektorja $\vec{a}$,$\vec{b}$ pravokotna« (Stöcker, 2018, str. 302).



Slika 6: Prikaz vektorskega produkta(prirejeno po Stöcker, 2018, str. 302).

Vektorje se da tudi transformirati (premikati), in sicer s pomočjo matrik.

2.4.2. Matrika

Matrika je pravokotna tabela realnih števil in je sestavljena in m-vrstic iz n-stolpcev, kjer sta m in n naravni števili. Matrika izgleda tako:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

V matematiki se reče, da je matrika A reda $m \times n$, kar pomeni, da ima matrika A m vrstic in n stolpcev. Posebna vrsta matrike je enotska matrika, to je matrika reda $m \times m$ in ima na glavni diagonali samo enice ostali elementi matrike so 0 in izgleda tako:

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

Prav tako kot enotska matrika je posebna tudi ničelna matrika. To je matrika, v kateri so vsi elementi enaki nič.

$$0 = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

Matrike je možno seštevati, odštevati, množiti s skalarji, množiti z drugimi matrikami in transponirati.

»Matriki A lahko prištejemo ali odštejemo matriko B, če imata matriki isti red« (Stöcker, 2018, str. 364).

$$A + B = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & \cdots & a_{1n}b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & \cdots & a_{mn}b_{mn} \end{bmatrix}$$

Vsota ali razlika matrik A in B, kjer sta matriki A in B reda $m \times n$, je matrika reda $m \times n$. Pri tej operaciji velja komutativnostni in asociativnostni zakon. Velja tudi:

- $A + 0 = A$
- $A + (-A) = 0$

Vsoto matrik sem implementiral tako, kot je prikazano.

```
// vsota
mat4 mat4::operator+(const mat4 &t) const
{
    mat4 tmp;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            tmp.mat[i][j] = mat[i][j] + t.mat[i][j];
    return tmp;
}
```

Naslednja operacija, ki jo lahko izvajamo z matrikami, je množenje matrike s skalarjem. Pri tej operaciji vsako komponento matrike pomnožimo s številom c (skalar).

$$cA = c\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} ca_{11} & \cdots & ca_{1n} \\ \vdots & \ddots & \vdots \\ ca_{m1} & \cdots & ca_{mn} \end{bmatrix}$$

Za množenje matrike s skalarjem velja komutativnostni, asociativnostni in distributivnostni zakon. Velja tudi, da če matriko pomnožimo z ena bo rezultat te operacije vhodna matrika:

$$1A = A$$

Matriko je možno pomnožiti z vektorjem, vendar le če velja, da je matrika reda $m \times n$ pomnožena z n-razsežnim vektorjem. Oziroma, kot navajajo avtorji knjige *Matematični priročnik z osnovami računalništva*: »Produkt matrike A je definiran le, če ima A natanko toliko stolpcev, kolikor ima vektor x komponent« (Stöcker, 2018, str. 367). Rezultat take operacije je m-razsežni vektor.

$$A\vec{b} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}\begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_{11}b_1 + a_{12}b_2 + \cdots + a_{1n}b_n \\ \vdots \\ a_{m1}b_1 + a_{m2}b_2 + \cdots + a_{mn}b_n \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_m \end{bmatrix} = \vec{c}$$

Če vektor pomnožimo z enotsko matriko I, je rezultat množenja isti vektor:

$$I\vec{b} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \cdot x + 0 \cdot y + 0 \cdot z \\ 0 \cdot x + 1 \cdot y + 0 \cdot z \\ 0 \cdot x + 0 \cdot y + 1 \cdot z \end{bmatrix} = \begin{bmatrix} x \cdot 1 \\ y \cdot 1 \\ z \cdot 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Matriko je poleg množenja s skalarjem in z vektorjem možno pomnožiti tudi z matriko. Za tako množenje obstajajo določena pravila: »Produkt AB pri množenju matrike A z matriko B je definiran le, če ima matrika A natanko toliko stolpcev kot ima matrika B vrstic« (Stöcker, 2018, str. 367).

Produkt množenja matrik A reda $m \times x$ in B reda $x \times n$ je matrika C reda $m \times n$.

$$C = AB = \begin{bmatrix} a_{11} & a_{11} & \cdots & a_{1x} \\ a_{21} & a_{21} & \cdots & a_{21} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & \cdots & a_{mx} \end{bmatrix}\begin{bmatrix} b_{11} & b_{11} & \cdots & b_{1n} \\ b_{21} & b_{21} & \cdots & b_{21} \\ \vdots & \vdots & \ddots & \vdots \\ b_{x1} & \cdots & \cdots & b_{xn} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + \cdots + a_{1x}b_{x1} & a_{11}b_{12} + a_{12}b_{22} + \cdots + a_{1x}b_{x2} & \cdots & a_{11}b_{1n} + \cdots + a_{1x}b_{xn} \\ a_{21}b_{11} + \cdots + a_{2x}b_{x1} & a_{21}b_{12} + a_{22}b_{22} + \cdots + a_{2x}b_{x2} & \cdots & a_{11}b_{12} + \cdots + a_{1x}b_{x2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{2x}b_{x1} & a_{m1}b_{12} + a_{m2}b_{22} + \cdots + a_{mx}b_{x1} & \cdots & a_{m1}b_{1n} + \cdots + a_{mx}b_{xn} \end{bmatrix}$$

Za posamezen element matrike C se da zapisati:

$$C_{ij} = a_i \cdot b_j = \sum_{l=1}^{x} a_{il}b_{lj}$$

V tej enačbi a_i nastopa kot vrstični vektor matrike A in b_j nastopa kot stolpčni vektor matrike B. Med a_i in b_j je skalarni produkt.

Matrično množenje ni komutativno, velja pa distributivnostni in asociativnostni zakon. Množenje matrike A z enotsko matriko I je A: $IA = A$ in množenje matrike A z ničelno matriko je ničelna matrika: $A0 = 0$.

Matrično množenje sem v svoji igri implementiral tako:

```
mat4 mat4::operator*(const mat4 &t) const
{
    mat4 tmp;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            for (int k = 0; k < 4; k++)
                tmp.mat[i][j] += mat[i][k] * t.mat[k][j];
    return tmp;
}
```

V računalništvu se za namene premikanje vektorja v prostoru uporablja matrike reda 4×4 in vektorje, ki imajo 4 dimenzije. Če vektor predstavlja točko, se njegovo zadnjo komponento w nastavi na 1, kar pomeni, da se vektor tretira kot točko. V primeru, da je w nastavljen na 0, pomeni, da je tam shranjena smer. Četrta komponenta vektorja igra ključno vlogo pri premikanju vektorja in pri premikih in projekcijah. Taki vektorji so homogeni. V svoji igri sem sicer uporabil trirazsežne vektorje, pred vsakim množenjem pa sem dodal četrto komponento.

2.4.3. Premik (Translacija)

Z matrikami je možno vektor premikati po prostoru. Prej sem omenil, da je nek 3D objekt, ki ga uporabljam v igri sestavljen iz množice točk. Te točke si lahko predstavljamo kot posamezne vektorje, ki imajo prijemališče v središču objekta. Če vsakega izmed vektorjev, ki tvorijo objekt, premaknemo za neko vrednost, dosežemo to, da se bo celoten objekt premaknil za neko vrednost. Za namene premikanja vektorjev, se uporablja translacijsko matriko. Translacijska matrika je definirana kot:

$$T = \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

V definiciji translacijske matrike P nastopa kot premik, ob njem pa je napisana os, po kateri naj se vektor premakne. Če s translacijsko matriko T pomnožimo vektor $\vec{a}$, bomo s tem dosegli, da se bo vektor $\vec{a}$ premaknil za P.

$$\vec{a}' = T\vec{a} = \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot x + 0 \cdot y + 0 \cdot z + P_x \cdot 1 \\ 0 \cdot x + 1 \cdot y + 0 \cdot z + P_y \cdot 1 \\ 0 \cdot x + 0 \cdot y + 1 \cdot z + P_z \cdot 1 \\ 0 \cdot x + 0 \cdot y + 0 \cdot z + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} x + P_x \\ y + P_y \\ z + P_z \\ 1 \end{bmatrix}$$

Četrta komponenta v vektorju poskrbi za to, da se vsak izmed premikov pomnoži z ena in se prišteje osnovnemu vektorju. Če le-te ne bi bilo premik ne bi bil mogoč. Prav tako, se vektor ne bi premaknil, če bi bila četrta komponenta nastavljena na nič, ker bi se vsak izmed premikov pred prištevanjem pomnožil z nič.

Matriko, ki služi za premikanje vektorja, sem ustvaril z metodo:

```
mat4 pozicijska(const mat4 &t, vec3 pozicija)
{
    mat4 tmp(1);
    tmp.mat[0][3] = pozicija.x;
    tmp.mat[1][3] = pozicija.y;
    tmp.mat[2][3] = pozicija.z;
    tmp.mat[3][3] = 1;
    return t * tmp;
}
```

2.4.4. Spreminjanje velikosti

Podobno, kot obstaja matrika za premikanje vektorjev, obstaja tudi matrika, ki vektor poveča ali zmanjša. Da bi vektor povečali ali zmanjšali, moramo vsako komponento vektorja pomnožiti z določeno številko. Če z matriko S, ki poveča vektor $\vec{a}$ za V, dobimo vektor $\vec{a}'$, ki je povečan ali pomanjšan za določeno vrednost.

$$S\vec{a} = \begin{bmatrix} V_x & 0 & 0 & 0 \\ 0 & V_y & 0 & 0 \\ 0 & 0 & V_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} V_x \cdot x + 0 \cdot y + 0 \cdot z + 0 \cdot 1 \\ 0 \cdot x + V_y \cdot y + 0 \cdot z + 0 \cdot 1 \\ 0 \cdot x + 0 \cdot y + V_z \cdot z + 0 \cdot 1 \\ 0 \cdot x + 0 \cdot y + 0 \cdot z + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} V_x \cdot x \\ V_y \cdot y \\ V_z \cdot z \\ 1 \cdot 1 \end{bmatrix}$$

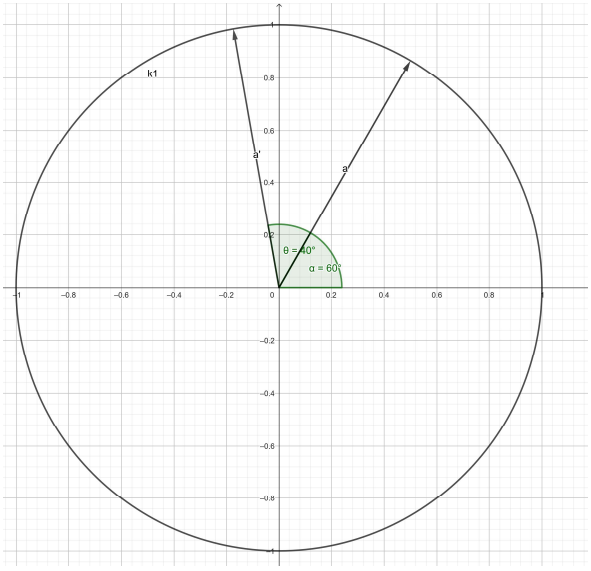
Koda, ki naredi matriko za spreminjanje velikosti, izgleda tako:

```
mat4 velikostna(const mat4 &t, vec3 velikost)
{
    mat4 tmp(1);
    tmp.mat[0][0] = velikost.x;
    tmp.mat[1][1] = velikost.y;
    tmp.mat[2][2] = velikost.z;
    tmp.mat[3][3] = 1;
    return t * tmp;
}
```

2.4.5. Rotacija

Tretja vrsta premikanja vektorja v trirazsežnem prostoru je rotacija. Rotacijska matrika je posebna vrsta matrike, ki se jo uporablja za vrtenje vektorjev v prostoru.

Rotacijo vektorja je najlažje razložiti v 2D prostoru, kjer vektor $\vec{a}$ zavrtimo za kot θ in dobimo vektor $\vec{a}'$. Dolžini obeh vektorjev sta enaki radiju krožnice.



Slika 7: Vrtenje vektorja v enotski krožnici(prirejeno po Sunshine2k, 2011).

Znano je:

$$a_x = r \cdot \cos a \text{ in}$$

$$a_y = r \cdot \sin a$$

x komponento novega vektorja se da izračunati po enačbi:

$$a'_x = \mathrm{r} \cdot \cos(\alpha + \theta) = r \cdot \cos\alpha \cos\theta - r \cdot \sin\alpha \sin\theta = a_x \cos\theta - a_y \sin\theta$$

Podobno se da izračunati y komponento novega vektorja:

$$a'_y = \mathrm{r} \cdot \sin(\alpha + \theta) = r \cdot \sin\alpha \cos\theta + \mathrm{r} \cdot \sin\theta \cos\alpha = a_y \cos\theta + a_x \sin\theta$$

Iz tega lahko izpeljemo matriko:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Če s to matriko pomnožimo vektor $\vec{a}$, bomo dobili vektor $\vec{a}'$, ki bo za kot θ zarotiran v nasprotni smeri urinega kazalca:

$$\vec{a}' = R\vec{a} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \cdot \cos\theta - y \cdot \sin\theta \\ x \cdot \sin\theta + y \cdot \cos\theta \end{bmatrix}$$

Matrika, ki zavrti vektor za kot θ okoli osi x, je definirana kot:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Če s to matriko pomnožimo vektor $\vec{a}$, dobimo vektor $\vec{a}'$, ki je za rotiran za kot θ.

$$R_x\vec{a} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 \cdot x + 0 \cdot y + 0 \cdot z + 0 \cdot 1 \\ 0 \cdot x + \cos\theta \cdot y - \sin\theta \cdot z + 0 \cdot 1 \\ 0 \cdot x + \sin\theta \cdot y + \cos\theta \cdot z + 0 \cdot 1 \\ 0 \cdot x + 0 \cdot y + 0 \cdot z + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} x \\ y\cos\theta - z\sin\theta \\ \mathrm{y}\sin\theta + \mathrm{z}\cos\theta \\ 1 \end{bmatrix}$$

Za rotacije po ostalih dveh oseh sta matriki zelo podobni. Rotacijske matrike sem v svoji igri naredil sspodnjo metodo:

```
mat4 rotacijska(const mat4 &t, vec3 rotacija)
{
    mat4 x(1), y(1), z(1);
    if (rotacija.z != 0)
    {
        z.mat[0][0] = cos(rotacija.z);
        z.mat[0][1] = -sin(rotacija.z);
        z.mat[1][1] = cos(rotacija.z);
        z.mat[1][0] = sin(rotacija.z);
    }
    if (rotacija.y != 0)
    {

        y.mat[0][0] = cos(rotacija.y);
        y.mat[0][2] = sin(rotacija.y);
        y.mat[2][2] = cos(rotacija.y);
        y.mat[2][0] = -sin(rotacija.y);
    }
    if (rotacija.x != 0)
    {

        x.mat[1][1] = cos(rotacija.x);
        x.mat[1][2] = -sin(rotacija.x);
        x.mat[2][1] = sin(rotacija.x);
        x.mat[2][2] = cos(rotacija.x);
    }

    mat4 tmp = y * x;
    tmp = z * tmp;
    return t * tmp;
}
```

2.4.6. Projekcija

V resničnem svetu se nam stvari, ki so od nas bolj oddaljene, zdijo manjše kot so v resnici, in stvari, ki so nam bližje, se zdijo večje. V 3D računalniških igrah se poskuša isti učinek posnemati s projekcijsko matriko.

Za ustvarjanje take matrike potrebujemo podatke o višini in širini okna (x, y), podatke o vidnem kotu (α) in o navidezni globini zaslona blizu in daleč.

Razmerje višine in širine okna a lahko izračunamo na način:

$$a = \frac{višina}{širina}$$

Projekcijska matrika je definirana na način:

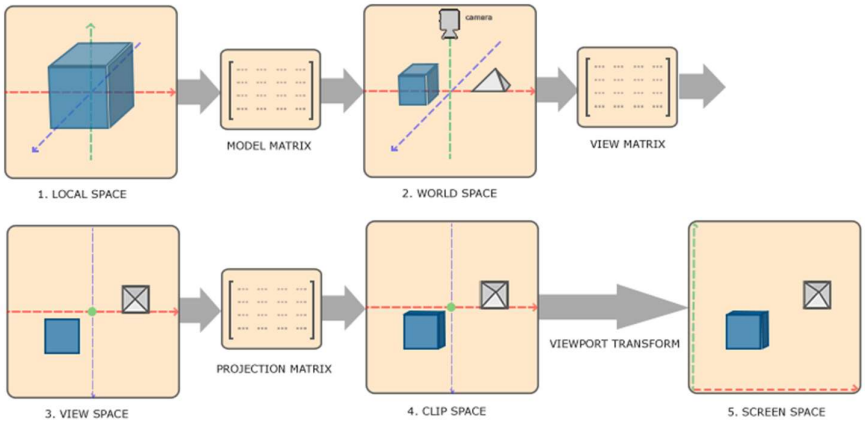
$$P = \begin{bmatrix} \frac{a}{\tan\frac{\alpha}{2}} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\frac{\alpha}{2}} & 0 & 0 \\ 0 & 0 & \frac{daleč}{daleč - blizu} & -\frac{daleč \cdot blizu}{daleč - blizu} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Če komponento x pomnožimo z a, dosežemo to, da se bodo vsi objekti prilagajali velikosti zaslona in da bodo ves čas v enakem razmerju. Nato je potrebno komponenti x in y deliti s faktorjem vidnega kota. Če bo vidni kot večji, bodo stvari manjše, in če bo manjši, bodo objekti večji. Tretji korak je prilagoditev z koordinat glede na »globino« zaslona. Projekcijsko matriko sem implementiral tako:

```
mat4 projekcija(const mat4 &t, float visina, float sirina, float vidni_kot, float z_blizu,
float z_dalec)
{
    mat4 tmp(0);
    float a = visina / sirina;
    vidni_kot = vidni_kot / 2;
    tmp.mat[0][0] = (1 / tan(vidni_kot)) * a;
    tmp.mat[1][1] = (1 / tan(vidni_kot));
    tmp.mat[2][2] = z_dalec / (z_dalec - z_blizu);
    tmp.mat[3][2] = 1;
    tmp.mat[2][3] = (-z_dalec * z_blizu) / (z_dalec - z_blizu);

    return t * tmp;
}
```

2.5. KOORDINATNI SISTEMI

Na začetku procesa risanja objekta so vse točke objekta postavljene relativno na središče objekta. Te točke so v objektnem prostoru (1. točka na spodnji sliki). Če vse točke premaknemo s pomočjo matrik, ki sem jih opisal v prejšnjem poglavju, objekt postavimo v svetovni prostor (2. točka na sliki), v katerem so še ostali objekti. Če proces nadaljujemo, objekt pomnožimo z matriko pogleda (kamero), ki je opisana v poglavju 2.6. Kamera, objekte premaknemo v nov prostor (3. točka na sliki). Da bi ustvarili iluzijo 3D sveta, jih na koncu pomnožimo še s projekcijsko matriko (4. točka na sliki). Koordinate točk, ki segajo izven enotskega prostora, obrežemo, tako kot je opisano v poglavju 2.3.2. Na koncu se objekti preslikajo v zaslonski prostor (5. točka na sliki).



Slika 8: Prikaz različnih prostorov v OpenGL (de Vries, 2018, str. 82).

2.6. KAMERA

V okolju OpenGL ni kamere ali njej podobne stvari. Ta sistem deluje na način, da vse ostale stvari premaknemo na tak način, da v igrici izgleda, kot da se je premaknila kamera. Ta premik se izvrši z množenjem z matriko kamere K.

$$K = \begin{bmatrix} D_x & D_y & D_z & 0 \\ G_x & G_y & G_z & 0 \\ S_x & S_y & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Kjer je:

- D vektor, ki kaže v desno stran od kamere,
- G vektor, ki kaže gor,
- S vektor, ki kaže v nasprotno smer pogleda kamere in
- P vektor, ki pove pozicijo kamere

2.7. SENČENJE

V koraku procesiranja slikovnih pik se implementira tudi senčenje. Sprva potrebujemo izvor luči. Najlažje je predpostaviti, da svetloba izvira iz točkastega objekta, ki ima podano pozicijo p_{luc} v 3D svetu. Nastavimo še barvo luči, označeno z b_l, in njeno intenzivnost, označeno z i. Za vsako

slikovno piko določenega objekta imamo podano tudi njeno normalo, imenovano n. Nato lahko izračunamo smer žarka s po enačbi:

$$\hat{s} = \frac{p_{luc} - p_{tocke}}{\|p_{luc} - p_{tocke}\|}$$

Pozicija točke je točka, ki je pomnožena le s translacijsko matriko. To je matrika, ki objekt premosti iz objektnega prostora v svetovni prostor. Iz teh podatkov je možno izračunati difuzivno osvetljenost, označeno z D_o. To je osvetljenost, ki pada z razdaljo in je odvisna od vpadnega kota.

$$D_0 = \begin{cases} n \cdot s + 0{,}1\ ;\ n \cdot s \geq 0 \\ 0{,}1\ ;\ n \cdot s < 0 \end{cases}$$

V primeru, da je D_o manjši od nič se nastavi vrednost 0,1. S tem poskrbimo, da bodo vse narisane ploskve, tudi tiste, ki so obrnjene proč od luči vsaj malo osvetljene.

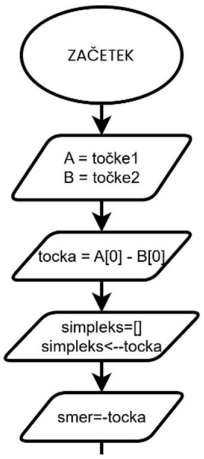
Kot zadnji korak senčenja se izračuna barva označena z b, ki jo pripišemo posamezni slikovni piki objekta:

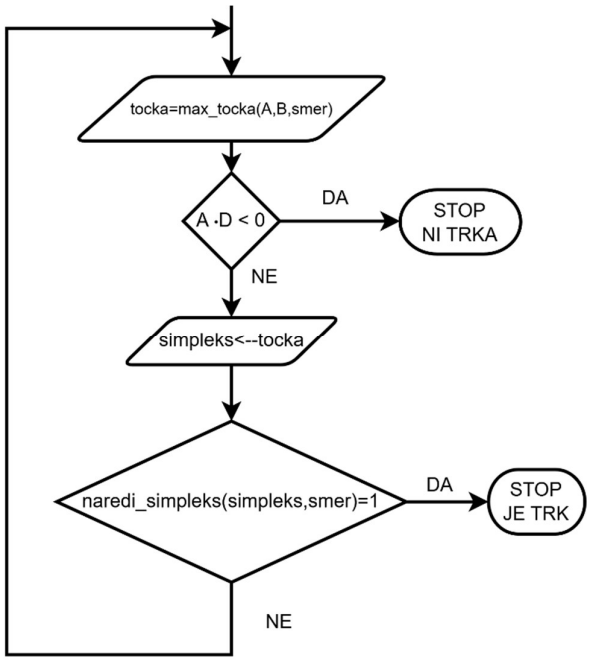
$$b = b_o \cdot b_l \cdot D_o \cdot i$$

2.8. GJK

GJK je računalniški algoritem, ki so ga zasnovali Elmer G. Gilbert, Daniel W. Johnson in S. Sathiya Keerthi leta 1988. Uporablja se za zaznavanje kolizije pri konveksnih oblikah. Za osnovo uporablja Minkowsko razliko. To je razlika dveh geometrijskih oblik, ki se jo izračuna na način, da vsaki točki prve geometrijske oblike odštejemo vsako točko druge geometrijske oblike. S tem nastane nova geometrijska oblika, za katero je značilno, da če je med začetnima geometrijskima oblikama kolizija, bo koordinatno središče znotraj na novo pridobljene geometrijske oblike. V nasprotnem primeru pa bo koordinatno izhodišče zunaj nove oblike. GJK algoritem je način, kako poiskati najpreprostejšo geometrijsko obliko (*simpleks*), ki lahko nastane iz točk, ki jih dobimo pri Minkowski razliki izhodiščnih oblik.

Algoritem deluje na način:





Slika 9: Diagram poteka GJK algoritma(prirejeno po Muratori, 2006)

Na začetku algoritma se določi točko iz minkowske razlike in se jo doda v tabelo točk, ki tvorijo simpleks. S pomočjo začetne točke je možno izračunati smer iskanja druge točke, ki je po navadi negativna vrednost začetne točke, torej v smeri koordinatnega izhodišča. V vsaki iteraciji se poišče naslednjo točko, ki naj bi segala čez koordinatno izhodišče, v nasprotnem primeru geometrijski obliki nista v koliziji. Če je točka uspešno poiskana, se jo doda v tabelo točk simpleks. Simpleks bo z vsako iteracijo imel več dimenzij. Nato se simpleks posreduje funkciji, ki glede na posredovan simpleks izračuna novo smer, v kateri se bo iskala točka, ki bo s prej poiskanimi točkami tvorila simpleks višje dimenzije. Ko bo tabela simpleks tvorila četverec in ko bo koordinatno izhodišče znotraj četverca, bo to pomenilo, da sta objekta v trku. Funkcija, ki glede na simpleks izračuna novo smer iskanja, je v moji igri taka:

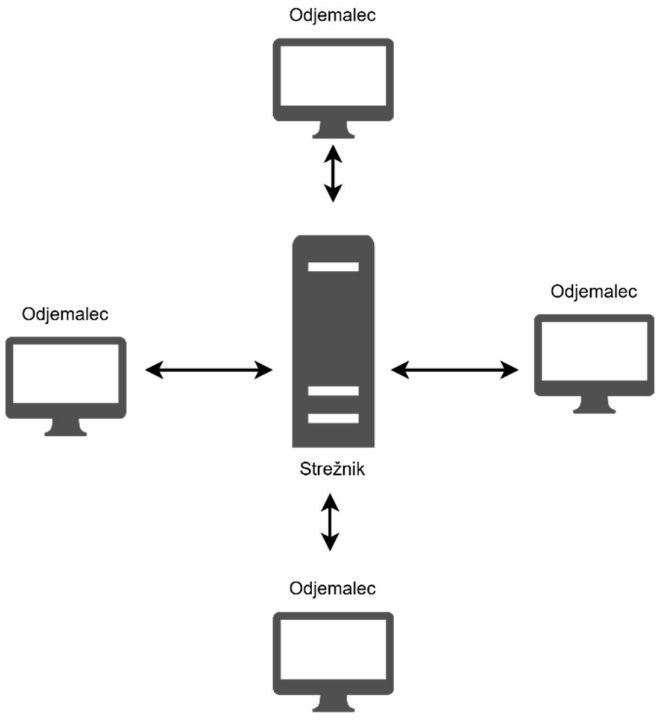
```
bool preveri_simpleks(std::deque<mat::vec3> &simpleks, mat::vec3 &smer)
{
    if (simpleks.size() == 2)
    {
        mat::vec3 A = simpleks[0], B = simpleks[1];
        mat::vec3 AB = B - A;            // vektor od A do B
        mat::vec3 A0 = mat::vec3(0) - A; // Smer od 0 proti A

        /*
        Če je koordinatno središče med A in B,
        se izračuna smer ki kaže od daljice do koordinatnega izhodišča
        */
        if (mat::skalarni_produkt(AB, A0) > 0)
        {
            smer = mat::vektorski_produkt(mat::vektorski_produkt(AB, A0), AB);
        }
        else //  v naspornem primeru se v simpleks da samo točko A in izračuna smer od A
proti koordinatnemu izhodišču
        {
            simpleks = {A};
            smer = A0;
```

```
        }
        return false;
    }
    if (simpleks.size() == 3) // V primeru, da imamo 2D simpleks (trikotnik)
    {
        mat::vec3 A = simpleks[0];
        mat::vec3 B = simpleks[1];
        mat::vec3 C = simpleks[2];

        /*
        Izračuna se daljice od B do A
        in od C do A
         */
        mat::vec3 AB = B - A;
        mat::vec3 AC = C - A;
        mat::vec3 A0 = mat::vec3(0) - A;

        /*
        Izračuna se normalo trikotnika
        */
        mat::vec3 normala_ABC = mat::vektorski_produkt(AB, AC); // pomaga določiti ali je
izhodišče nad ali pod trikotnikom

        /*
        Če je izhodišče na strani trikotnika, ki jo določa vektor AC, in je izven
trikotnika, izberemo novo iskalno smer glede na A in C.
        */
        if (mat::skalarni_produkt(mat::vektorski_produkt(normala_ABC, AC), A0) > 0)
        {
            simpleks = {A, C};
            smer = mat::vektorski_produkt(mat::vektorski_produkt(AC, A0), AC);
        }
        /*
        V primeru da je koordinatno izhodišče med a in b in je izven trikotnika
        se v simpleks da doljico A do b in smer se izračuna od daoljice proti koordinatnemu
izhodišču
        */
        else if (mat::skalarni_produkt(mat::vektorski_produkt(AB, normala_ABC), A0) > 0)
        {
            simpleks = {A, B};
            smer = mat::vektorski_produkt(mat::vektorski_produkt(AB, A0), AB);
        }
        /*
        če je koordinatno izhodišče znotraj trikotnika se izračuna novo smer glede na od
trikotnika proti koordinatenemu izhodišču
        */
        else
        {
            smer = normala_ABC * (mat::skalarni_produkt(normala_ABC, A0) > 0 ? 1 : -1);
        }
        return false;
    }
    if (simpleks.size() == 4)
    {
        mat::vec3 A = simpleks[0];
        mat::vec3 B = simpleks[1];
        mat::vec3 C = simpleks[2];
        mat::vec3 D = simpleks[3];

        mat::vec3 A0 = mat::vec3(0) - A;

        mat::vec3 normala_ABC = mat::vektorski_produkt(B - A, C - A);
        mat::vec3 normala_ACD = mat::vektorski_produkt(C - A, D - A);
        mat::vec3 normala_ADB = mat::vektorski_produkt(D - A, B - A);

        // Ti trije pogoji preverijo ali je koordinatno izhodišče zunaj tetraedra
        // V pogojih se preverja ali je koordinatno izhodišče v smeri normale, ki kaže izven
četverca
        if (mat::skalarni_produkt(normala_ABC, A0) > 0) // Če je koordinatno izhodišče zunaj
se je nova smer normala, ki kaže izven ploskve
        {
            simpleks = {A, B, C};
```

```
            smer = normala_ABC;
            return false;
        }
        if (mat::skalarni_produkt(normala_ACD, A0) > 0)
        {
            simpleks = {A, C, D};
            smer = normala_ACD;
            return false;
        }
        if (mat::skalarni_produkt(normala_ADB, A0) > 0)
        {
            simpleks = {A, D, B};
            smer = normala_ADB;
            return false;
        }
        return true; // če je znotraj pomeni, da je trk
    }

    return false;
}
```

2.9. KOMUNIKACIJA V OMREŽJU

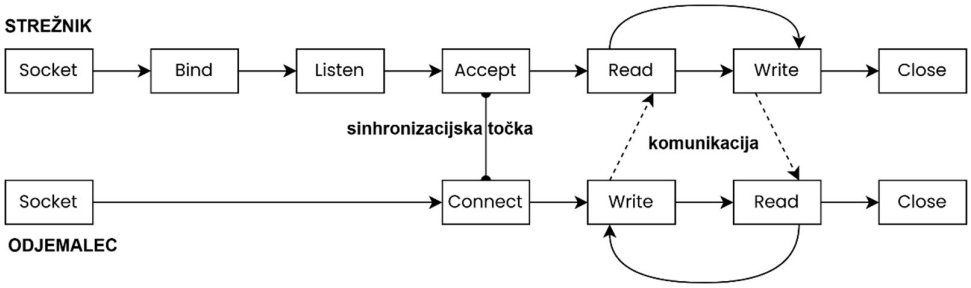
Igra VSSR je narejena tako, da lahko več igralcev igra istočasno. Za to funkcionalnost se morajo podatki o vseh igralcih prenašati po spletu. Za ta namen sem uporabil sporočilni sistem vtičnic. Večigralski del igre je zasnovan tako, da je nekje zagnana strežniška aplikacija, na katero se povežejo vsi igralci.



Slika 10: Prikaz modela s strežnikom in odjemalci (lasten vir).

Sistem vtičnic ima nekaj osnovnih funkcij oziroma oz. primitivnih ukazov, kot jih poimenuje knjiga *Računalniška omrežja z internetnimi storitvami*. Te funkcije so:

- »*Socket* kreriranje povezave
- *Bind* povezovanje lokalnega naslova z vtičnico
- *Listen* najava pripravljenosti
- *Accept* sinhronizacija pozivajočega za komunikacijo
- *Connect* aktivno vzpostavljanje povezave
- *Send* pošiljanje podatkov
- *Recive* sprejemanje podatkov
- *Close* sproščanje povezave« (Vidmar, 2013, str. 172).



Slika 11: Princip delovanja sistema vtičnic. (prirejeno po Vidmar, 2013, str. 172).

Na strežniški strani je potrebno odpreti vtičnico, jo povezati z lokalnim naslovom in začeti poslušati. Ko se na strežnik poskuša povezati odjemalec, ga je potrebno sprejeti. Takrat se začne komunikacija. Odjemalec in strežnik si v določenih časovnih intervalih izmenjujeta sporočila. Odjemalec posluša za morebitna sporočila na ločeni niti, ki se izvaja asinhrono in ločeno od glavne niti programa. Strežnik ima podobno kot odjemalec posebno nit za sprejemanje sporočil. Ima pa tudi posebno nit za pošiljanje podatkov.

Vsako sporočilo, ki se pošlje, ima vnaprej definirano strukturo. Za vsa sporočila je skupno, da se začnejo s bajtom v katerem je zapisan tip sporočila. Temu bajtu pa sledijo ostali podatki, ki so značilni za vrsto sporočila.

2.10. DELOVANJE IGRE

Igra ima glavno zanko v kateri se osvežuje zaslon. Na začetku zanke se zaslon pobriše, da se lahko nanj izrisujejo novi elementi. To naredi klic funkcije Risalnik::zacetek_okvir() Nato se, kliče glavno funkcijo trenutne scene, ki se izrisuje s klicem Risalnik::aktivna_scena_ptr->zanka() Na koncu se v funkciji Risalnik::konec_okvir() obdelajo vsi dogodki in se zamenja medpomnilnik, ki prepreči utripanje slike;.

Glavna zanka izgleda tako:

```
while (!Risalnik::ali_je_okno_za_zapreti())
{
    Risalnik::zacetek_okvir();
    if (Risalnik::aktivna_scena_ptr)
        Risalnik::aktivna_scena_ptr->zanka();
    Risalnik::konec_okvir();
}
```

V projektu je več scen, te so Igra_scena, v kateri je glavna logika igre, Zacetna_scena, v kateri je glavni meni, scena z nastavitvami in scena z navodili za igranje. Vsaka izmed scen ima funkcijo zanka() ta funkcija na zaslon izriše vse elemente scene. Razred risalnik ima shranjen kazalec, ki kaže na trenutno aktivno sceno, tako da lahko kliče njeno funkcijo.

Na začetku scene Igra_scena se odpre še dodatna nit, ki iz omrežja sprejema podatke. Glavna zanka se odvija v takem zaporedju dogodkov:

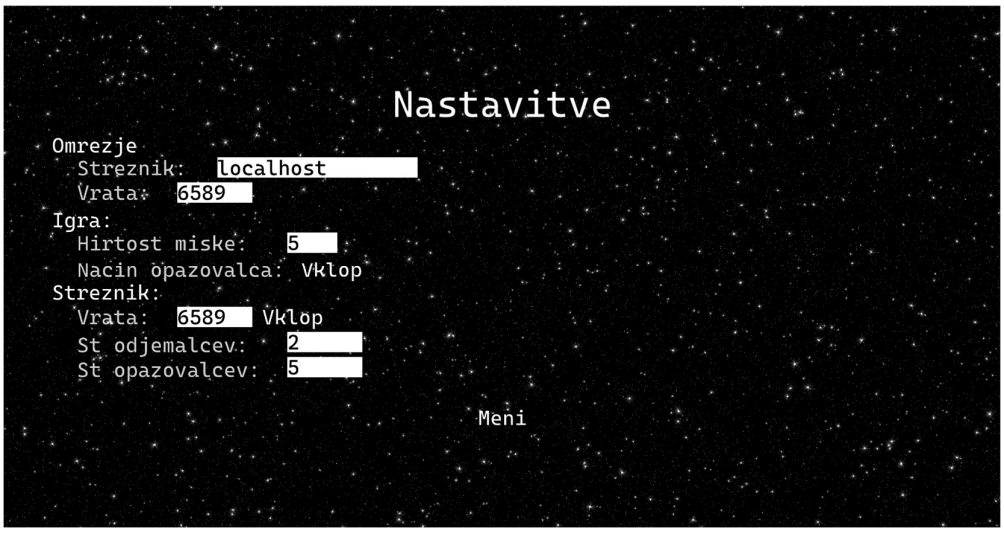
- Če je pavza ali kazen, ker je uporabnik umrl, se izriše poseben meni.
- Izriše se ozadje.
- Narišejo se vsi meteorji in se preverja kolizije med njimi.
- Izriše se merilec (kurzor na sredini zaslona).
- Izriše se število življenj.
- Če je uporabnik pritisnil levo tipko, se izstreli izstrelek.
- Izrišejo se vidni števci za teleportacijo in ponovno streljanje.
- Izpiše se število ladij, ki jih je igralec zadel.
- Izrišejo se izstrelki in se preverja ali je igralčeva ladja zadeta.
- Izrišejo se nasprotne ladje.
- Preverja se, ali je igralcu zmanjkalo življenj.
- Program preveri, ali je bila pritisnjena katera tipka.
- Na koncu zanke se glede na skrite števce strežniku pošilja podatke.

3. REZULTATI IN UGOTOVITVE

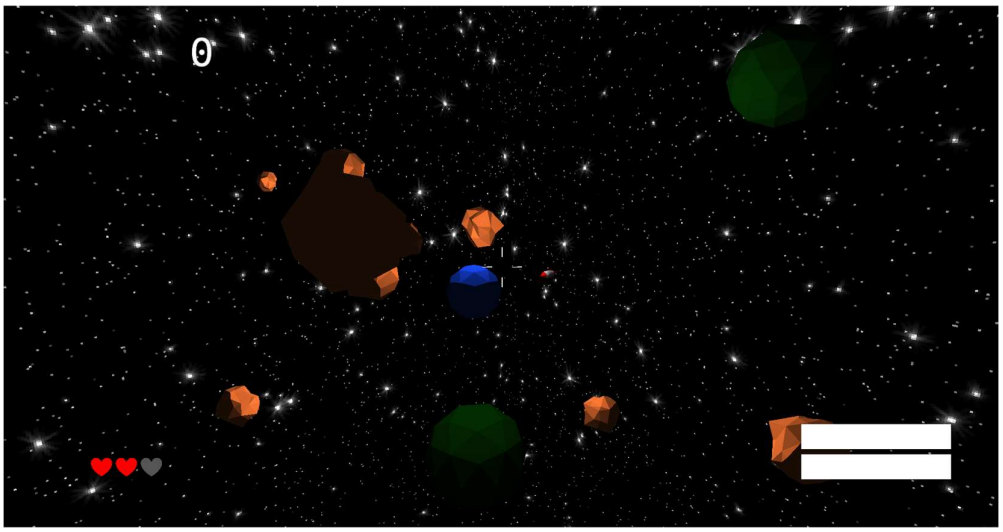
3.1. REZULTAT

Po 4622 vrsticah kode in mnogih urah truda je s pomočjo pristopov, omenjenih v prejšnjih poglavjih, nastala računalniška igra Vesoljski spopad super raket (VSSR). Spodaj je nekaj slik, ki prikazujejo, kako le-ta izgleda.

Slika 12: Glavni meni igre (lasten vir).



Slika 13: Meni z nastavitvami (lasten vir).



Slika 14: Prikaz igranja igre (lasten vir).

Cilj igre je uničiti nasprotnikove vesoljske ladje. Za premik v smeri gledanja je potrebno pritisniti tipko w, za nasprotno smer s, za v levo a in v desno d. Večjo natančnost streljanja omogoča desni klik. Ko namerite v nasprotnikovo ladjo pritisnite levi klik, da ga zadenete. Ob pritisku na tipko T si bo program zapomnil lokacijo na katero se lahko kasneje teleportirate s pritiskom na R. Če uporabnik prileti na zeleno gmoto in pritisne tipko H, si poveča število življenj. Za premor je potrebno pritisniti tipko z napisom ESC. Igra omogoča tudi način opazovalca; v tem načinu igralec ni vključen v igro, lahko jo samo spremlja.

3.2. MOŽNE IZBOLJŠAVE

Iz varnostnega vidika, bi moral igro zasnovati tako, da se vse odvija na strežniku. Trenutno pa se vse odvija pri odjemalcu, strežnik le pošilja podatke med odjemalci. V taki igri je zelo lahko goljufati, npr. ko odjemalec gotovi, da je zadel tujo ladjo, pošlje to sporočilo strežniku. Strežnik to pošlje zadetemu odjemalcu. Ko zadeti odjemalec dobi to sporočilo, si sam sebi odšteje eno življenje. Možno je, da odjemalec sam sebi ne bi odštel življenj in tako postal neuničljiv.

Ves prenos podatkov temelji na protokolu UDP. Ko se prenašajo pomembni podatki, kot je na primer vstop v igro, kjer se igralcu dodeli ID, se lahko zgodi, da paket ne prispe.

Za vsak izrisan objekt v igri pokličem funkcijo, da ga izriše. To igro upočasnjuje. Če bi več objektov zapakiral v en *glfwDraw* klic, bi se igra izvajala hitreje. Prav tako bi bilo bolje, da so točke objektov, ki se pogosto izrisujejo, shranjene v pomnilniku GPU, ne pa da se prenašajo za vsak klic.

4. ZAKLJUČEK

Napisati lastno igro v C++ z uporabo čim manj knjižnic je zahtevno, vendar je tudi zelo zanimivo in poučno. Nikoli si nisem mislil, da je v ozadju preproste igre toliko matematike. Spoznal sem tudi principe delovanja grafičnih knjižnic in sisteme za spremljanje verzij projekta (*git*). Čeprav igra na videz ni preveč obetavna, sem z njo zelo zadovoljen.

5. VIRI IN LITERATURA

- *3D projection*. (2025, March 21). Wikipedia, the free encyclopedia. Pridobljeno 6. marca 2025, Dostopno prek: https://en.wikipedia.org/wiki/3D_projection
- Benkovič, D., & Pagon, D. (2014). *Vektorji in matrike*.
- *Camera matrix*. (2023, June 28). Wikipedia, the free encyclopedia. Pridobljeno 6. marca 2025, Dostopno prek: https://en.wikipedia.org/wiki/Camera_matrix#
- *CMake: The standard build system*. (n.d.). CMake - Upgrade Your Software Build System. https://cmake.org/features/
- *Dot product*. (2023, August 4). Wikipedia, the free encyclopedia. Pridobljeno 6. marca 2025, Dostopno prek: https://en.wikipedia.org/wiki/Dot_product
- *Euclidean vector*. (2025, March 12). Wikipedia, the free encyclopedia. Pridobljeno 6. marca 2025, Dostopno prek: https://en.wikipedia.org/wiki/Euclidean_vector
- *Gilbert–Johnson–Keerthi distance algorithm*. (2024, June 18). Wikipedia, the free encyclopedia. Pridobljeno 6. marca 2025, Dostopno prek: https://en.wikipedia.org/wiki/Gilbert%E2%80%93Johnson%E2%80%93Keerthi_distance_algorithm
- (n.d.). GLFW. https://www.glfw.org/
- *How to multiply matrices*. (n.d.). Math is Fun. https://www.mathsisfun.com/algebra/matrix-multiplying.html
- *Implementing GJK - 2006* [Video]. (2016, May 12). YouTube. https://youtu.be/Qupqu1xe7Io
- *Implementing GJK (2006)*. (n.d.). Implementing GJK (2006). https://caseymuratori.com/blog_0003
- Khronos Group. (n.d.). OpenGL - The Industry Standard for High Performance Graphics. https://www.opengl.org/
- *Matrike*. (n.d.). Astra. https://astra.si/algebra/matrike/
- *Minkowski addition*. (2025, January 8). Wikipedia, the free encyclopedia. Pridobljeno 6. marca 2025, Dostopno prek: https://en.wikipedia.org/wiki/Minkowski_addition
- Möller, T., Haines, E., & Hoffman, N. (2018). *Real-time rendering*. A K PETERS.
- *Network socket*. (2025, February 22). Wikipedia, the free encyclopedia. Pridobljeno 6. marca 2025, Dostopno prek: https://en.wikipedia.org/wiki/Network_socket
- *Perspective projection matrix (Math for game developers)* [Video]. (2021, October 8). YouTube. https://youtu.be/EqNcqBdrNyI?list=PLYnrabpSIM-93QtJmGnQcJRdiqMBEwZ7_
- *Projection (linear algebra)*. (2025, February 17). Wikipedia, the free encyclopedia. Pridobljeno 6. marca 2025, Dostopno prek: https://en.wikipedia.org/wiki/Projection_(linear_algebra)#
- *Rotation matrix derivation (step-by-step prove)* [Video]. (2021, April 29). YouTube. https://youtu.be/EZufiIwwqFA
- *Rotation matrix*. (2023, October 26). Wikipedia, the free encyclopedia. Pridobljeno 6. marca 2025, Dostopno prek: https://en.wikipedia.org/wiki/Rotation_matrix
- Shreiner, D., Sellers, G., Kessenich, J. M., & Licea-Kane, B. (2013). *OpenGL programming guide: The official guide to learning OpenGL, version 4.3*. Addison-Wesley Professional.
- Stöcker, H. (2006). *Matematični priročnik*. Tehniška založba Slovenije.
- Sunshine2k. (n.d.). *Rotation derivation*. Sunshine's Homepage. https://www.sunshine2k.de/articles/RotationDerivation.pdf
- *Transformations*. (n.d.). Math is Fun. https://www.mathsisfun.com/geometry/transformations.html
- Vidmar, T., & Ciglarič, M. (2013). *Računalniška omrežja Z internetnimi storitvami*.
- Vries, J. D. (2020). *Learn OpenGL: Learn modern OpenGL graphics programming in a step-by-step fashion*.
- *Wavefront .obj file*. (2025, March 18). Wikipedia, the free encyclopedia. Pridobljeno 6. marca 2025, Dostopno prek: https://en.wikipedia.org/wiki/Wavefront_.obj_file

ZAHVALA MENTORICI

Rad bi se zahvalil svoji mentorici profesorici Nataši Makarovič, saj mi je s svojimi vselej dobrodošlimi konstruktivnimi kritikami in nasveti pomagala zasnovati igro in je nato s svojimi usmerjanji pripomogla k temu, da sem projekt uspešno zaključil.