Programsko ostvarenje algoritma roja čestica u jeziku C++

Perić, Matko

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:850492

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-18**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

MATKO PERIĆ

PROGRAMSKO OSTVARENJE ALGORITMA ROJA ČESTICA U C++

Završni rad

Pula, rujan 2020.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

MATKO PERIĆ

PROGRAMSKO OSTVARENJE ALGORITMA ROJA ČESTICA U C++

Završni rad

JMBAG: 0303034317, izvanredni student

Studijski smjer: Informatika

Predmet: Strukture podataka i algoritmi

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, rujan 2020.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Matko Perić, kandidat za prvostupnika Informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student
Matko Perić

U Puli, 01.09, 2020 godine

IZJAVA
o korištenju autorskog djela

Ja, Matko Perić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom Programsko ostvarenje algoritma roja čestica u c++ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 01.09.2020 (datum)

Potpis

_____

Sadržaj

UVOD……6
1. ALGORITAM ROJA ČESTICA……3
1.1 Opis algoritma……3
1.2 Primjena algoritma……4
2. RAZRADA TEME……4
2.1 Problematika završnog rada……4
2.2 Praktična primjena……5
2.3 Pregled postojećih programa……5
3. STRUKTURA PROGRAMA……5
3.1 Organizacija datoteka……5
3.2 Tijek rada programa……6
4. 3D PROSTOR……8
4.1 Model, Pogled, Projekcija……8
4.2 Jedinice za sjenčanje……9
4.3 Mapiranje kockom……10
5. OBJEKTI……11
5.1 Ptica……11
5.1.1 Varijable……13
5.1.2 Funkcije……14
5.2 Riba……21
5.2.1 Varijable……21
5.2.2 Funkcije……21
6. PROGRAMI JEDINICA ZA SJENČANJE……22
6.1 Varijable……22
6.2 Procesor vrhova……23
6.3 Procesor fragmenata……24
7. SPREMNICI VRHOVA……24
7.1 Varijable……24
7.2 VAO i VBO spremnici vrhova……25
7.3 IBO spremnik……26
8. ZAKLJUČAK……28
LITERATURA……30
Popis slika i kodova……31
Popis tablica……32
SAŽETAK / Abstract……33

UVOD

U današnjem modernom i tehnološkom svijetu ljudi sve više koriste znanost i modernu tehnologiju da bi im olakšala ostvariti svoje planove i ciljeve. Virtualni svijet, koji je ljudima sve bliži, otkriva svoje mogućnosti i prednosti. Svijet simulacija i prikaza realnog svijeta kroz virtualni postaje primamljiv, privlačan i zanimljiv sve većem broju ljudi. Gameing industrija ulazi u svijet simulacija i znanosti što zvuči odlično. Sama ideja ovog rada je pokušati spojiti područja igre i znanosti kroz simulaciju. Iz navedenoga je nastala i ideja za ovim radom kako bi se na neki način pokušalo shvatiti i približiti taj novi čarobni svijet.

Zamisao ovog završnog rada je da se koristi algoritam roja čestica i napravi grafička simulacija algoritma u 3D prostoru. Sam program simulira jato ptica kako traže ribu. Na neki način, plan je da se na samom primjeru iz kojeg je nastao algoritam napravi i program. Ptice pretražuju prostor tražeći ribu. Problem koji se rješava je pretraga prostora. Cilj je optimizirati tu pretragu i sve to grafički prikazati.

Program koji je priložen uz ovu projektnu dokumentaciju napravljen je u programskom jeziku C++, razvojnom okruženju Visual Studio i koristi grafičko sučelje OpenGL. U programu se koriste i dodatne knjižnice kao primjerice glew i glfw3 te matematička knjižnica glm.

U dokumentaciji su navedeni objekti koji se koriste u programu s opisima funkcija i podataka. Isto tako navedeni su i opisani programi za sjenčanje i spremnici koje se koriste. Ideja je da se prikaže i približi moderni OpenGL sa svojim osnovnim konceptom. Cilj nije bio da se napravi najmoderniji i najbolji program već osnovni tako da se može i vidjeti od kojih dijelova se sastoji i na koji način funkcionira. Današnji način korištenja modernog OpenGL-a je dosta složen i kompleksan. Korišteni programi za sjenčanje su prikazani u projektnoj dokumentaciji sa svojim varijablama i uz kratki opis. Pisani su u jeziku GLSL (OpenGL Shading Language).

U programu je stvoren 3D prostor u kojem se ptice kreću. Napravljen je isto tzv. „Skybox“ koji sa slikama okružuje 3D prostor. Težište nije samo na samom algoritmu već i na izgledu tj. prikazu same simulacije koja je podjednako zahtjevna. Iako nema puno objekata, ima jako puno funkcija koje optimiziraju pretragu i sam prikaz

simulacije. U programu se koristi osnovni algoritam roja čestica bez ikakvih izmjena, pošto kao takav zadovoljava i ostvaruje cilj. Osim samog algoritma koji je primijenjen u letu ptice postoje i razne funkcije koje optimiziraju kretanje same ptice. Način na koji lete, kut rotacije modela. Navedeno je bio zahtjevniji dio programa jer sam algoritam ne vodi računa o tome. Za potrebe programa izrađen je i 3D model galeba.

1. ALGORITAM ROJA ČESTICA

Algoritam roja čestica je prirodom inspiriran algoritam. Spada u heurističke algoritme jer se koristi iskustvenim spoznajama, nadalje spada u optimizacijske algoritme jer optimizira pretragu prostora i pritom koristi cijelu populaciju čestica pa ga to svrstava i u populacijske algoritme.

1.1 Opis algoritma

Algoritam roja čestica koristi roj čestica ili jedinki da bi optimizirao pretragu. Jedinka pretražuje prostor i na osnovu vlastitog iskustva i iskustva roja odlučuje u kojem pravcu će dalje pretraživati prostor. Prilikom izračuna smjera kretanja, svaka jedinka u određenoj mjeri uzima u obzir svoje do tada pronađeno najbolje rješenje koje možemo nazvati individualni faktor, te najbolje pronađeno rješenje svoje bliske okoline koje možemo nazvati socijalni faktor [4].

Susjedstvo ili socijalni faktor kojim se definira prijenos informacija možemo najčešće podijeliti na potpuno povezanu; gdje sve jedinke u svakom trenutku izmjenjuju informacije (globalno), prstenastu; gdje je svaka jedinka povezana sa dvije najbliže jedinke i zajedno čine zatvoreni krug i von Neumannovu; gdje je svaka jedinka povezana sa 4 susjedne jedinke koje čine matricu [3].

Pretraga se dodatno optimizira i s dodatnim parametrima koji ovise o samoj vrsti pretrage. Najčešće se koriste dodatni težinski parametri koji onda mogu utjecati na prioritet informacija koje jedinke dobivaju i time mogu usmjeravati samu pretragu.

1.2 Primjena algoritma

Algoritam se primjenjuje u optimizaciji i istraživanju raznih problema koji tradicionalnim putem nisu mogući u stvarnom vremenu. Široka je primjena u filmskoj industriji gdje se koriste razne inačice algoritma za kontroliranje velikih masa objekata, zatim u robotici, bioinformatici, biomedicini, isto tako se primjenjuje i u analizi slika i videa. Sve prisutniji su u primjeni dronova i bespilotnih letjelica koji uz pomoć algoritma pretražuju prostor u potrazi za požarima ,neovlaštenim ulazima na neko područje, čuvanju granica, vojnoj industriji itd. [7].

2. RAZRADA TEME

Programsko ostvarenje je napravljeno kao grafički prikaz samog algoritma. Jedinke pretražuju prostor u realnom vremenu. Model galeba predstavlja jedinku koja leti i traži ribu koja se nalazi na površini mora.

2.1 Problematika završnog rada

Problematika se može podijeliti na nekoliko cjelina, prva je izrada programa u OpenGl-u, druga je sam algoritam roja čestica tj. pretraga i treća je grafički prikaz samog algoritma u 3D prostoru.

Problematika vezana za izradu programa je složena i obuhvaća izradu samog 3D prostora, perspektive, način korištenja grafičkog procesora i grafičke memorije vezane za prikaz modela jedinki i samog prostora.

Algoritam je ukomponiran u sam prostor u kojem vrši pretragu. Na taj način je optimiziran i prilagođen. Cilj je napraviti simulaciju jata ptica koji pretražuju prostor koristeći algoritam roja čestica i koji predstavljaju prirodne procese koji se mogu promatrati. Izbor dodatnih parametara vezanih za optimizaciju pretrage kojima se balansira između brzine pretrage i vizualnog izgleda je potreban. Isto tako je i susjedstvo potrebno prilagoditi realnom svijetu.

Na kraju je tu more i ptice koje lete i traže ribu. Dolazi se do grafičkog prikaza. On nosi svoju problematiku koja je vezana za realan prikaz. Samo kretanje ptice i smjer određuje algoritam ali brzinu i okretanje modela ptice se optimiziralo i prilagodilo da bi prikaz pretrage izgledao realno.

2.2 Praktična primjena

Kako je problematika višeslojna tako i sama primjena može se kretati u više pravaca. Koristeći grafički prikaz može se optimizirati sam algoritam u određenom okruženju, tj. može se mijenjati i prilagođavati zadanim ciljevima i pritom vizualno promatrati. Vizualni prikaz može služiti za bolju optimizaciju algoritma roja čestica gdje se izmjenama na određenim parametrima mogu promatrati promjene.

S druge strane sam grafički program može se koristiti za simulacije i prikaze raznih algoritama gdje se može koristiti postojeći 3D prostor kao okruženje u kojem će se algoritmi primjenjivati i izvršavati.

2.3 Pregled postojećih programa

Postojeća rješenja mogu se podijeliti u nekoliko skupina. U prvu skupinu spadaju rješenja koja optimiziraju pretragu i nude samo rezultat bez vizualnog prikaza. U drugu skupinu mogu se staviti oni programi koji uz rješenja daju i prikaz rezultata ili same pretrage, najčešće u oblika grafa. Sljedeća skupina programa je prikaz samog rada algoritma u 2D prostoru gdje se može promatrati sama pretraga, najčešće su čestice prikazane točkom. I u posljednju skupinu mogu se staviti programi sa vizualnim prikazom rada u 3D prostoru, koji su vezani za simulaciju nekih postojećih objekata ili jedinki. Isto tako postoje programi koji izmjenjuju i optimiziraju sam algoritam stvarajući time njegove inačice koje su prilagođene rješavanju problema i mogu se nalaziti u svim prethodnim skupinama.

3. STRUKTURA PROGRAMA

Struktura programa je jednostavna i može se podijeliti na objekte koji se kreću, prostor u kojem se kreću i sam algoritam roja čestica koji govori na koji način se objekti kreću tj. pretražuju prostor.

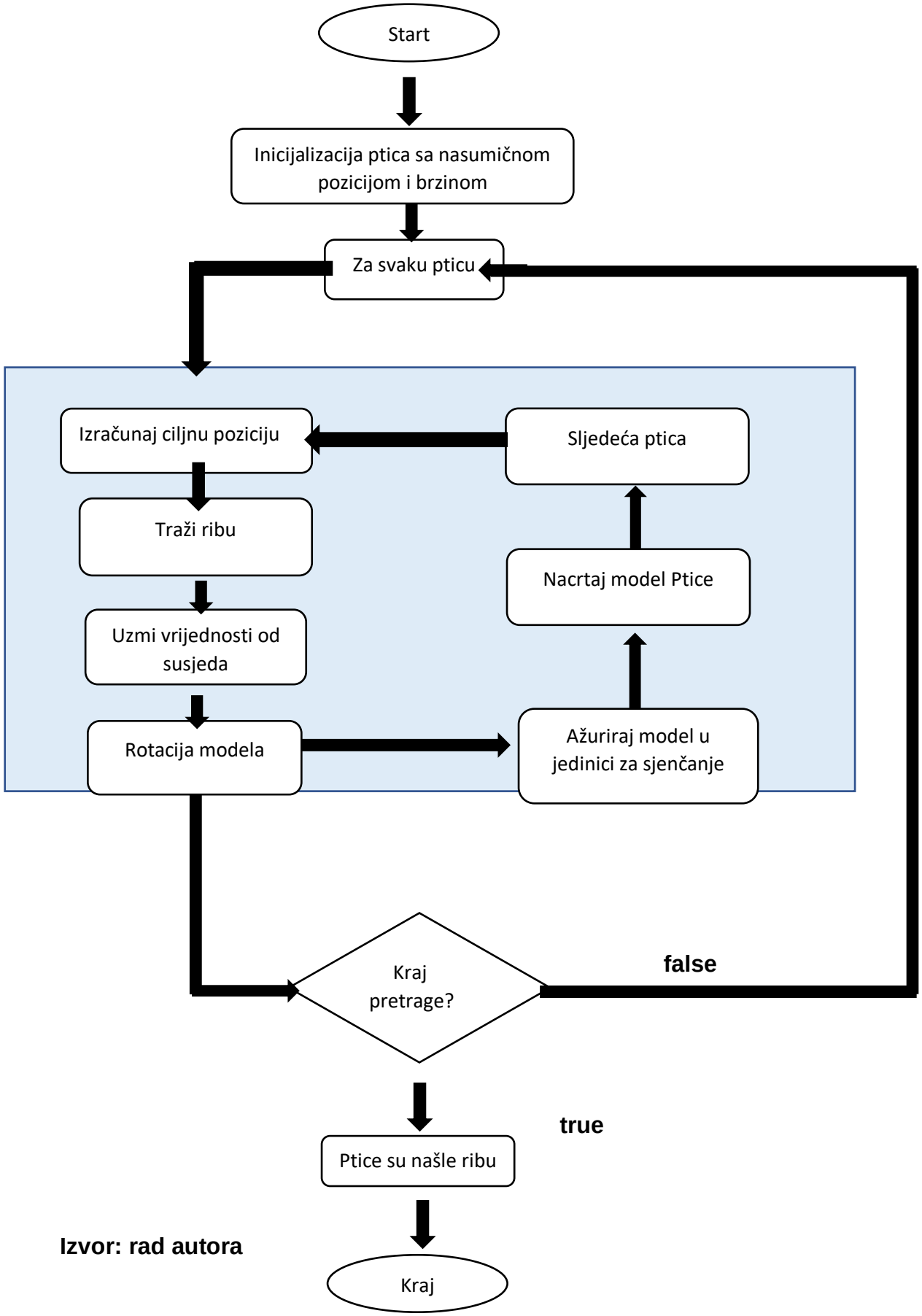
3.1 Organizacija datoteka

Sam program je napravljen u više datoteka. Sadrži dvije klase Ptica i Riba koje su podijeljene u datoteke zaglavlja Ptica.h i Riba.h u kojoj su definicije samih klasa i deklaracije njenih funkcija. Zatim datoteke Ptica.cpp i Riba.cpp u kojoj je implementacija klasa i definicije funkcija. Program ima glavnu datoteku main.cpp u kojoj se koriste objekti klase Riba i Ptica, isto tako nalaze se funkcije za stvaranje jedinica za sjenčanje i sam kod za procesore vrhova i fragmenata za objekte kao i za samu okolinu („skybox“). Program koristi stb_image.h datoteku za prikaz slika. U main.cpp datoteci se nalazi i glavna petlja u kojoj se izvršava sam prikaz algoritma sa potrebnim funkcijama.

3.2 Tijek rada programa

U priloženom dijagramu se može vidjeti detaljna shema na koji način algoritam u programu radi. Na početku se inicijaliziraju ptice s nasumično dodijeljenim vrijednostima kao što su pozicija ptice i dodatne varijable koje služe za određivanje početnog kretanja ili smjera same ptice. Zatim svaka ptica ima svoj niz funkcija koje optimiziraju pretragu i let ptice kao što su funkcije leti, traži ribu, vrijednost susjeda i rotacija modela. S desne strane dijagrama se nalaze funkcije i programi vezani za samo crtanje ptice. Funkcije se ponavljaju sve dok ptice ne pronađu ribu.

Slika 1.: Dijagram tijeka rada programa



Izvor: rad autora

4. 3D PROSTOR

Za potrebe programa napravljen je trodimenzionalni prostor u kojem se objekti kreću. U programu je koordinatni sustav gdje je X širina, Y visina, a Z dubina prostora. Centar ili ishodište samog prostora je 0,0,0 tj. 0 po svim dimenzijama prostora.

4.1 Model, pogled, projekcija

Da bi se uspješno napravila simulacija objekata koji se kreće u 3D prostoru mora se ispravno sagledati što čini taj prostor i koje su komponente potrebne da se prikaz samog programa ispravno vidi. U programu se koristi perspektivna projekcija, što znači da se u obzir uzima daljina i blizina objekata tj. objekti se smanjuju i povećavaju ovisno o udaljenosti od pogleda.

Za projekciju se koristi glm funkcija perspective koja uzima potrebne parametre.

Prvi parametar je kut gledanja, drugi je veličina početne plohe koja je određena omjerom širine i visine, treći i četvrti parametar govore o dubini samog prostora

Kod 1.: Prikaz koda perspektivne projekcije

```
glm::mat4 projekcija;
projekcija=glm::perspective(45.0f,(GLfloat)Sirina/(GLfloat)Visina, 0.1f, 100.0f);
```

Izvor: [1].

Model matrice predstavlja objekte koji imaju svoj lokalni prostor. Potreban je i pogled ili oko od kojeg se gleda model i sam prostor. Pogled predstavlja *pogled* matrica. Također, pogled se nije zaokrenuo već ga se samo malo približilo po z osi.

Kod 2.: Prikaz koda za *model* i *pogled*

```
glm::mat4 model;  // objekt(ptica)
glm::mat4 pogled;  // oko kamere

pogled = glm::translate(pogled, glm::vec3(0.0f, 0.0f, -30.0f));
```

Izvor: [1].

4.2 Jedinice za sjenčanje

Da bi sve navedeno funkcioniralo u modernom OpenGL-u koriste se programi za sjenčanje koji su detaljnije opisani u petom poglavlju. Ovdje se govori samo o onome što je potrebno za naš 3D svijet.

U programu za sjenčanje se koriste 3 uniformne varijable: *model*, *pogled* i *projekcija*. Njihova se lokacija dohvaća pomoću funkcije glGetUniformLocation koja ima dva parametra. Prvi je sam program u kojem se nalazi uniformna varijabla i drugi je naziv same varijable [6].

Kod 3.: Prikaz koda za uzimanje lokacije iz programa za sjenčanje

```
GLint modelLocation = glGetUniformLocation(shader, "model");
GLint pogledLocation = glGetUniformLocation(shader, "pogled");
GLint projLocation = glGetUniformLocation(shader, "projekcija");
```

Izvor: [1].

Nakon što se dohvate lokacije varijabli, moraju im se predati vrijednosti iz našeg programa koji se nalazi u CPU. Pomoću funkcije glUniformMatrix4fv predaju se vrijednosti na način da prvi parametar govori na kojoj lokaciji se nalazi dok zadnji predaje samu vrijednosti. *Pogled* i *projekciju* vrijednosti dovoljno je predati jednom budući da se one ne mijenjaju tijekom rada cijelog programa. Za vrijednost samog *modela* funkciju pozivamo prije crtanja jer sam model se mijenja tj. kreće se i okreće.

Kod 4.: Prikaz koda predavanje vrijednosti u program za sjenčanje

```
glUniformMatrix4fv(pogledLocation, 1, GL_FALSE, glm::value_ptr(pogled));
glUniformMatrix4fv(projLocation,1,GL_FALSE,glm::value_ptr(projekcija);
```

Izvor: [1].

4.3. Mapiranje kockom

U programu se koristi tehnika mapiranja kockom koja je najprikladnija za prikaz okruženja našeg svijeta u kojem se ptice kreću. Princip je jednostavan, potrebne su šest slika koje će se postaviti na svaku stranicu kocke koja će okruživati dotičan svijet. Budući da se sve zbiva unutar stranica kocke, dolazi se do predodžbe da se korisnik nalazi u svijetu koji se nalazi na samim slikama.

Za sve navedeno najprije se omogućilo korištenje teksture u programu i vezalo ju se za taj ID.

Kod 5.: Prikaz koda teksture

```
unsigned int textureID;
glGenTextures(1, &textureID);
glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);
```

Izvor: [1].

Cijeli kod za stvaranje i prikazivanje samog okruženja treba i svoje spremnike i svoje programe za sjenčanje i razne funkcije koje koriste slike te ih postavljaju na pravo mjesto. Cijeli kod može se vidjeti na stranici LearnOpenGL.com pod poglavljem „Cubemaps“ [1].

5. OBJEKTI

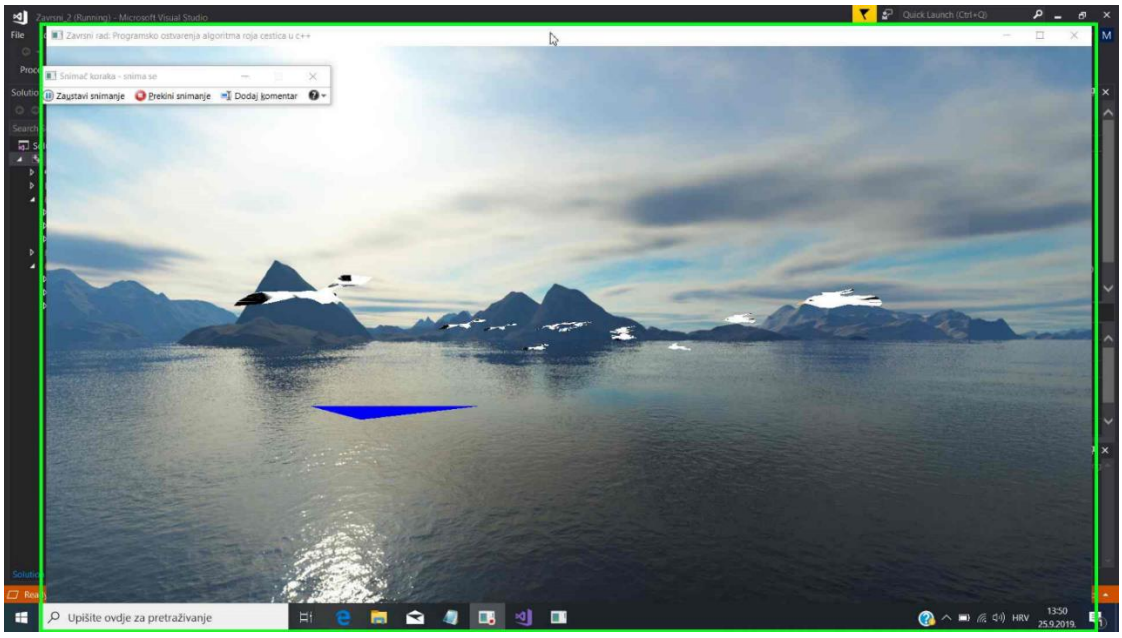
U programu su napravljene dvije klase Ptica i Riba. Ptica ima niz funkcija i varijabli koje služe za pretragu prostora dok Riba ima samo ulogu da se nalazi na određenom mjestu i „čeka“ da ju ptice pronađu.

5.1 Ptica

Ptica u programu pretražuje prostor i traži ribu. Kreće se algoritmom roja čestica. U konstruktoru objekta su dodijeljene nasumične varijable kao parametri kojima se služi da pretražuje prostor kao što su pozicija, susjedova pozicija ribe i globalna pozicija ribe. Od tih pozicija se računa planirana pozicija [3].

Objekt ptice koristi i dodatne varijable koje služe kao težinski faktori. Na početku svaka ptica ima težinski faktor postavljen na svoju najbolju poziciju, tj. time se daje prednost onome što sama ptica traži. Ako na tom putu od susjeda dobije bolju poziciju ribe onda se težinski faktor prebacuje na susjedovu najbolju poziciju. Ukoliko ptica dođe do svoje najbolje pozicije i ne pronađe ribu onda se težinski faktor prebacuje na globalnu poziciju ribe. Isto tako prema tim parametrima se određuje smjer rotacije samog modela ptice.

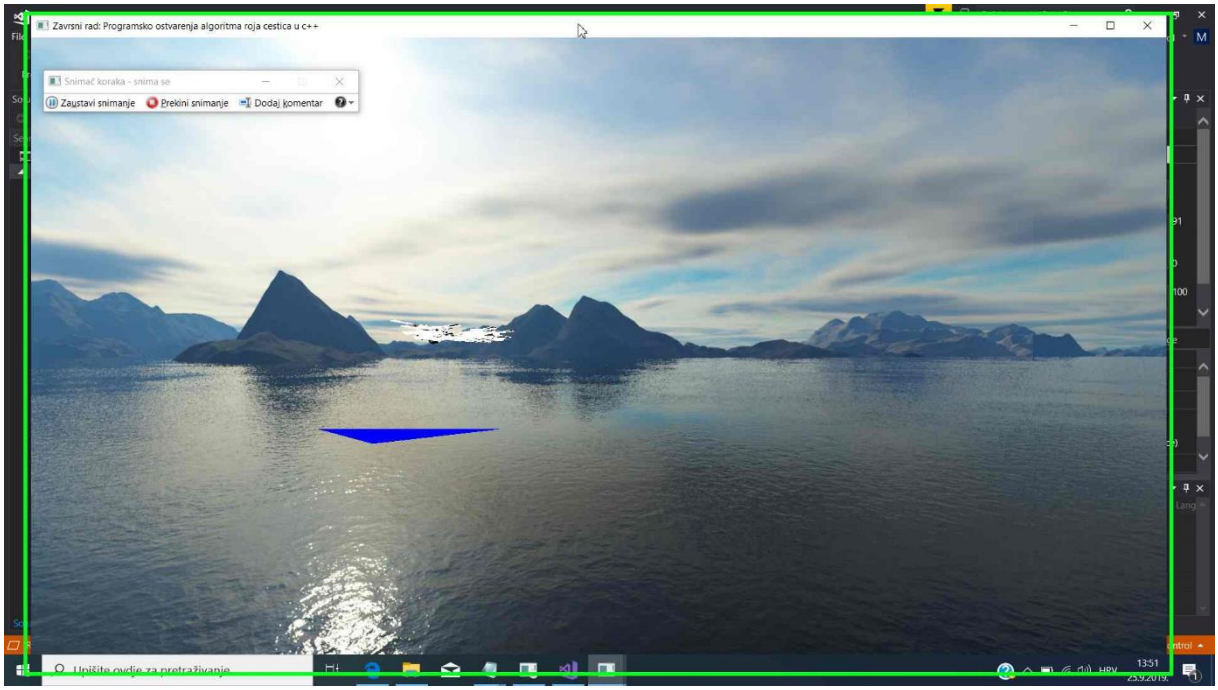
Slika 2.: Ptice su nasumično raspoređene i započinju pretragu



Izvor: snimak iz programa

Kad jedna ptica pronađe ribu onda se pozicija ribe stavlja u globalnu varijablu tako da onda sve ptice znaju gdje se nalazi riba. Ptica ima svoj smjer, kad se pronađe riba onda se mijenja i smjer (rotacija) samog modela. Na kraju ptice se okupe iznad ribe kao što se vidi na slici broj 3.

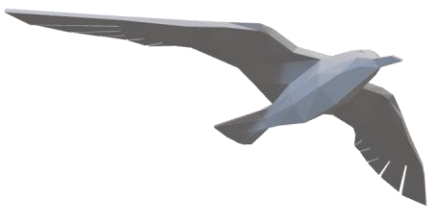
Slika 3.: Ptice su pronašle ribu



Izvor: snimak iz programa

U programu se koristi i 3D model galeba. Sve jedinke jata imaju isti model koji se onda kreće prema svojim parametrima pretrage.

Slika 4.: 3D model galeba koji se koristi u programu



Izvor: rad autora

5.1.1 Varijable

U klasi Ptica koriste se razne varijable od pozicije i boje modela do varijabli koje su potrebne za letenje i izračune potrebnih vrijednosti.

Tablica 1.: Varijable u klasi Ptica

NAZIV VARIJABLE	TIP	OPIS
Pozicija	vec3	Trenutna pozicija modela
ciljnaPozicija	vec3	Ciljna pozicija modela
najboljaPozicija	vec3	Najbolja nađena pozicija
najboljaPozicijaSusjeda	vec3	Najbolja nađena pozicija od susjeda
naj_glob_pozicija	static vec3	Najbolja nađena pozicija od svih modela
Susjedi	vector<Ptica>	Vektor modela ptica koji su trenutni susjedi tj. trenutno najbliži modelu
Vrijednost	GLfloat	Vrijednost nađene ribe
najboljaVrijednost	GLfloat	Najbolja vrijednost nađene ribe
najboljaVrijednostSusjeda	GLfloat	Najbolja vrijednost nađene ribe susjeda
naj_glob_vrijednost	static GLint	Najbolja nađena globalna vrijednost ribe
Kut	GLfloat	Stari kut pravca modela ptice
novi_kut	GLfloat	Novi kut pravca modela ptice
c1	GLfloat	Težinski faktor pravca kretanja modela prema svojim nađenim pozicijama

c2	GLfloat	Težinski faktor pravca kretanja prema susjedovoj najboljoj poziciji ribe
c3	GLfloat	Težinski faktor pravca kretanja prema globalnoj poziciji ribe
Vrhovi	Glfloat []	Pozicije vrhova modela
Boja	static Glfloat []	Boje modela

Izvor: rad autora

5.1.2. Funkcije

U klasi Ptica nalaze se sljedeće funkcije:

1) Funkcije koje uzimaju koordinate x, y, z modela ptice u prostoru

- float dohvati_X()
- float dohvati_Y()
- float dohvati_Z()

2) Funkcije koje postavljaju koordinate x, y, z u prostoru

- void postavi_X(float x)
- void postavi_Y(float y)
- void postavi_Z(float z)

- void traziRibu(const Riba polje[], int size);

Funkcija uzima argument *polje Riba* i veličinu polja *size*. Tipa je void i dio je klase Ptica. Sastoji se od dva dijela. U prvom se pretražuje prostor i traži riba uz pomoću for petlje koja prolazi kroz sve ribe i provjerava da li se u neposrednoj blizini određene ptice nalazi pozicija ribe, a to čini na način da provjeri da li od *pozicije.x* (riba) +-2 se nalazi *pozicija.x* ptice i tako za z. Y označava visinu pa smo ga izostavili.

Ukoliko je riba u blizini neke ptice, uspoređuje vrijednost ribe i ako je veća od dosad nađene postaje nova najbolja pozicija koju je ptica otkrila (*najboljaPozicija* i *najboljaVrijednost*) te se isto tako odredi nova globalna pozicija ribe. Poziva se funkcija *odrediKut* koja računa novu vrijednost kuta modela ptice.

Drugi dio funkcije služi ukoliko nije pronađena riba, a došlo se do svoje najbolje pozicije koja se na početku nasumično dodjeljuje svakoj jedinki, onda prebacujemo težište na globalnu poziciju i mijenjamo smjer i kut modela jedinke.

Kod 6.: Prikaz koda funkcije traziRibu kojom se jedinke koriste tražeći ribu

```
void Ptica::traziRibu(const Riba polje[], int size){

for (int i = 0; i < size; i++) {


if (pozicija.x >= polje[i].pozicija.x - 2.0f && pozicija.x <= polje[i].pozicija.x + 2.0f
&& pozicija.z >= polje[i].pozicija.z -2.0f && pozicija.z <= polje[i].pozicija.z + 2.0f) {

                        // provjeri vrijednost ribe
                        if (polje[i].vrijednost > najboljaVrijednost) {
                                najboljaPozicija = polje[i].pozicija;
                                najboljaVrijednost = polje[i].vrijednost;
                                c1 = 0.7f;  // ako sam ja našao ribu onda c1 je prioritet
                                odrediKut();  // odredi kut
                                Ptica::naj_glob_pozicija = polje[i].pozicija
                        }

                }
        }
        // ako si blizu najboljePozicije
if (pozicija.x >= najboljaPozicija.x - 2.0f && pozicija.x <= najboljaPozicija.x + 2.0f
   && pozicija.z >= najboljaPozicija.z - 2.0f && pozicija.z <= najboljaPozicija.z + 2.0f) {
                // ako nisi našao onda odredi novi kut prema globalnom
                if (najboljaVrijednost == 0 ) {
                        c3 = 0.7f;
                        c1 = 0.05f;

                        odrediKut();

                                                }
                    }

}
```

Izvor: rad autora

- void leti();

Funkcija *leti* računa i određuje let svake jedinke. U ovoj funkciji se primjenjuje algoritam roja čestica. Funkcija je tipa *void* i dio je klase Ptica.

Pravac kretanja ptice određuje se u dva koraka putem *ciljnePozicije* i *pozicije* koja je ciljna pozicija + trenutna pozicija.

Ciljna pozicija je u stvari planirana pozicija ili gdje bi trebala ptica doći. Ona se računa na način da se zbraja pozicija gdje je sama ptica vidjela ribu (*najboljaPozicija* - *pozicija*), gdje je susjedna ptica vidjela ribu (*najboljaPozicijaSusjeda* - *pozicija*) i gdje je najbolja pozicija ribe koju su vidjele sve ostale ptice (*naj_glob_pozicija* - *pozicija*) [4].

Sama funkcija ima i dodatne varijable *dg1*, *dg2*, *dg3* kojima je dodijeljena slučajna vrijednost između 0 i 1 koja služi kao slučajna vrijednost uniformne distribucije s kojom onda utječemo na pravac kretanja.

Isto tako u funkciji se koriste tri dodatne varijable *c1*, *c2*, *c3* koji su težinski koeficijenti kojima se određuje važnost određene pozicije ili joj se daje prednost. *c1* utječe na poziciju ribe koju je sama ptica našla ili traži, dok *c2* utječe na poziciju ribe koja se dodjeljuje preko susjednih ptica i *c3* utječe na poziciju ribe koja se nalazi u globalnoj poziciji. Svaka ptica na početku ima težište na *c1*, tj. na poziciju ribe koju je ona sama pronašla. Ako od susjeda dobije poziciju ribe onda se prebacuje težište na *c2* a ako na svom putu nije našla ribu onda se težište stavlja na *c3* tj. prema globalnoj poziciji ribe.

Varijable *kut* i *novi_kut* služe da se ptica ne okreće prebrzo već postepeno s time da se uzima u obzir i razlika između dva kuta pa se jedinka okreće prema manjoj razlici.

Kod 7.: Prikaz koda funkcije *leti*, kojom se jedinke koriste za let

```
void Ptica::leti() {

	GLdouble dg1, dg2, dg3;
	dg1 = distribution(generator);
	dg2 = distribution(generator);
	dg3 = distribution(generator);


ciljnaPozicija = (0.005f * ciljnaPozicija) +
	(c1 * (GLfloat)dg1 * (najboljaPozicija - pozicija)) +
	(c2 * (GLfloat)dg2 *   (najboljaPozicijaSusjeda - pozicija)) +
	(c3 * (GLfloat)dg3 * (Ptica::naj_glob_pozicija - pozicija));

// ovo je radi laganog skretanja od starog kuta,
// isto treba vidjeti da li je razlika veća od 180o, pa ako je onda neka se okreće
//obrnuto...
		if (kut < novi_kut) {
			// ako ide preko 180 onda ide sa druge strane -3
			if ((novi_kut - kut) < 180)
				kut += 3.0f;
			else
				kut -= 3.0f;
			// jer -1 je 359 stupnjeva
			if (kut < 0) kut = 360.0f;
		}

		if (kut > novi_kut) {

			if ((kut - novi_kut) <= 180)
				kut -= 3.0f;
			else
				kut += 3.0f;

			if (kut > 360) kut = 0.0f;
		}

		// dok se ne pronađe riba
		if (c3 < c1 && najboljaVrijednost==0)
			c1 += 0.01;

		pozicija = pozicija + ciljnaPozicija * 0.01f;  // puta 0.01f da se uspori

}
```

Izvor: rad autora

- void odrediSusjeda(Ptica polje[], int size);

Funkcija uzima polje ptica kao argument i veličinu polja *size*. Vrste je *void* i pripada klasi Ptica. Funkcija pretražuje koji su modeli u blizini ptice i stavlja ih u spremnik (*vector) susjedi*. Pomoću for petlje prolaze se sve ptice i provjeri se koje su ptice u našoj blizini 5+-, one koje se nalaze u našoj blizini postaju nam susjedi. Ova funkcija koristi na kraju funkciju *azurirajVrijednosti* tako da se odrede najbolje vrijednosti od svih susjeda. Ptice imaju dinamičke susjede tj. oni se mijenjaju i ovise o blizini, zbog toga je tu i funkcija koja onda odredi najbolje vrijednosti od trenutnih susjeda.

Kod 8.: Prikaz koda funkcije *odrediSusjeda*

```
void Ptica::odrediSusjeda(Ptica polje[], int size) {
susjedi.clear();  // obrišem na početku jer biram uvijek nove susjede

for (int i = 0; i < size; i++) {
                // +5 i -5 od mojih trenutnih koordinata
    if (polje[i].pozicija.x >= pozicija.x - 5.0f && polje[i].pozicija.x <= pozicija.x + 5.0f
      && polje[i].pozicija.y >= pozicija.y - 5.0f && polje[i].pozicija.y <= pozicija.y + 5.0f
      &&  polje[i].pozicija.z >= pozicija.z - 5.0f && polje[i].pozicija.z <= pozicija.z + 5.0f)
{

                        susjedi.push_back(polje[i]);
}
}
// nakon sto sam odredio sve susjede uzimam najbolju poziciju susjeda i njegovu
vrijednost ako su bolji od moje

        azurirajVrijednosti();

        susjedi.clear();
}
```

Izvor: rad autora

- void azurirajVrijednosti();

Funkcija pretražuje vrijednosti susjeda i ažurira ih na način da „vidi“ da li je „moja“ *najboljaVrijednost* manja od susjedove pa ako je onda uzima novu vrijednost i novu poziciju gdje se riba nalazi. Isto tako se težinski faktori mijenjaju tako da se poveća

c2 koji daje onda prednost susjedovoj poziciji ribe i mijenja se i kut modela prema novoj poziciji. Na kraju se svi susjedi obrišu. Funkcija pripada klasi Ptica i ne vraća nikakvu vrijednost.

Kod 9.: Prikaz koda funkcije *azurirajVrijednosti*

```
void Ptica::azurirajVrijednosti() {

for (int i = 0; i < susjedi.size(); i++) {
        if (najboljaVrijednost < susjedi[i].najboljaVrijednost ||
najboljaVrijednostSusjeda < susjedi[i].najboljaVrijednostSusjeda) {

                najboljaVrijednostSusjeda = susjedi[i].najboljaVrijednost;

                najboljaPozicijaSusjeda = susjedi[i].najboljaPozicija;

                // težinski faktori i novi kut modela ptice
                if (c3 < c1) {
                        c2 = 0.5f;
                        c1 = 0.05f;
                }
            }
          if (c2 > c1) odrediKut();
        }
        susjedi.clear();
}
```

Izvor: rad autora

- void odrediKut();

Funkcija koja određuje kut modela ptice. Kut je bitan da bi se realno prikazao let ptice. Ideja je sljedeća prvo se napravi jedna varijabla *cilj*, koja je ustvari novi vektor koji se dobije tako da se oduzme pozicija jedinke od ciljne pozicije. Pošto se u algoritmu roja čestica koriste tri razne pozicije za računanje leta ptice onda i imamo tri moguća cilja.

Zatim se odredi kut cilja na način da se dužina po *x*-u podjeli sa udaljenosti. Time se dobije kosinus kuta koji se stavlja u varijablu *trid_xz*. Zatim se pomoću funkcije acos() dobije kut u radijanima. Pomoću kosinusa se ne zna u kojem se kvadrantu nalazi jedinka tj. za dva različita kvadranta dobije se isti kosinus pa su se putem *if* uvjeta

odredili točni kvadranti. Isto tako u funkciji se radijani pretvaraju u stupnjeve. Na kraju koji će se cilj koristiti određuju težinski faktori *c1*,*c2*,i *c3*. Funkcija pripada klasi Ptica.

Kod 10.: Prikaz koda funkcije *odrediKut*

```
void Ptica::odrediKut() {

	glm::vec3 cilj;

	// ovisno o c postavljam cilj za izračun kuta
	 if(c3>c1 && c3>c2)
	 cilj = naj_glob_pozicija - pozicija;

	 if (c2>c1 && c2>c3)
		cilj = najboljaPozicijaSusjeda - pozicija;

	 if (c1>c2 && c1>c3)
		cilj = najboljaPozicija - pozicija;


	float trid_xz = cilj.x / (sqrt(cilj.x * cilj.x + cilj.z * cilj.z));

// dolnji lijevi kvadrant , prvo uvećam za 90 pa oduzmen od 180 i razliku od 360

	if (cilj.x < 0.0f && cilj.z > 0.0f) {
		novi_kut = (acos(trid_xz)) * 180.0f / 3.1415 + 90;
		novi_kut = 360 - (novi_kut - 180);
	}
	if (cilj.x < 0.0f && cilj.z < 0.0f) {

		novi_kut = acos(trid_xz) * 180.0f / 3.1415 + 90 ;

	}
	if (cilj.x > 0.0f && cilj.z > 0.0f) {

		novi_kut = acos(trid_xz) * 180.0f / 3.1415  ;

	}
	if (cilj.x > 0.0f && cilj.z < 0.0f) {

		novi_kut = acos(trid_xz) * 180.0f / 3.1415 + 90;
	}

}
```

Izvor: rad autora

- friend void ispis(Ptica polje_ptica[], int vel);

Funkcija uzima argument *polje ptica* u kojem se nalaze sve jedinke i njegovu veličinu argument *vel*. Ispisuje sve vrijednosti varijabli ptice. Prijateljska funkcija klase Ptica.

5.2. Riba

Objekt ribe služi samo kao pozicija u prostoru koju traže ptice te ima samo osnovne/minimalne varijable i funkcije.

5.2.1. Varijable

Koriste se varijable koje sadrže osnovne podatke o poziciji i boji.

Tablica 2.: Varijable u klasi Riba:

NAZIV VARIJABLE	TIP	OPIS
Pozicija	vec3	Trenutna pozicija modela ribe
Vrijednost	Glfloat	Vrijednost ribe, količina
Vrhovi	static Glfloat []	Pozicije vrhova modela
Boja	static Glfloat []	Boje vrhova modela

Izvor: rad autora

5.2.2. Funkcije

U programe se koriste funkcije za dohvaćanje podataka iz klase Riba.

U klasi Riba nalaze se sljedeće funkcije:

1) Funkcije koje uzimaju koordinate x, y, z modela ribe u prostoru
- GLfloat dohvati_X();
- GLfloat dohvati_Y();
- GLfloat dohvati_Z();

2) Funkcija koja uzima vrijednost ribe, tj. količinu

- GLfloat dohvatiVrijednost();

6. PROGRAMI JEDINICA ZA SJENČANJE

Program koristi jedinice za sjenčanje verzije 330. Sami programi jedinica za sjenčanje nisu izdvojeni u zasebnu klasu već su napisani u glavnoj (main.cpp) datoteci. Napravljene su dvije funkcije, prva koja služi za stvaranje samih jedinica za sjenčanje i druga funkcija koja ih stvara i povezuje u jedan program [2].

6.1. Varijable

U programu se koriste varijable za spremanje koordinata i boja, i varijable vezane za prikaz prostora. Slijedi detaljan opis svih varijabla koje se koriste u programu za sjenčanje jedinica

Tablica 3.: Varijable koje se koriste u programima jedinicama za sjenčanje:

NAZIV VARIJABLE	TIP	OPIS
ul_vrhovi	vec3	Sadrži koordinate, tj. pozicije vrhova modela
ul_boja	vec3	Sadrži boju pozicijskih vrhova modela
iz_boja	vec3	Sadrži boju koja se prenosi u procesor fragmenata
Model	mat4	Sadrži podatke o objektu
Pogled	mat4	Sadrži podatke o pogledu

		ili kameri
Projekcija	mat4	Sadrži podatke o perspektivi prostora
gl_Position	vec4	Sadrži konačne pozicije modela koji se prikazuje na ekranu
gl_FragColor	vec4	Sadrži boju modela koja se prikazuje na ekranu

Izvor: rad autora

6.2. Procesor vrhova

U procesor vrhova ulaze dvije varijable ul_vrhovi za pozicijske točke vrhova modela i ul_boja za boju. Tim podacima je dodijeljena i točna lokacija 0 i 1 koja odgovara poziciji podataka koji se nalaze u spremnicima. Isto tako imamo tri globalne varijable *model*, *pogled*, *projekcija* koje određuju završnu poziciju modela na ekranu. Iz procesora vrhova izlazi boja koja se prebacuje u procesor fragmenata [2].

Kod 11.: Prikaz koda procesor vrhova koji se koristi u programu

```
"#version 330\n"
        "layout (location = 0) in vec3 ul_vrhovi; \n"
        "layout (location = 1) in vec3 ul_boja; \n"


        " out vec3 iz_boja; \n"

        "uniform mat4 model;\n"
        "uniform mat4 pogled;\n"
        "uniform mat4 projekcija;\n"

        "void main(void) { \n"
        "iz_boja = ul_boja; \n"
        " gl_Position = projekcija * pogled * model * vec4(ul_vrhovi, 1.0f); \n"
        " } "
```

Izvor: [1].

6.3. Procesor fragmenata

U procesoru fragmenata ulazi varijabla iz_boja i izlazi gl_FragColor koja određuje boje modela koji se pojavljuje na ekranu [2].

Kod 12.: Prikaz koda procesor fragmenata koji se koristi u programu

```
"#version 330\n"
        " in vec3 iz_boja; \n"
        " out vec4 gl_FragColor; \n"
        "void main(void) { \n"
        " gl_FragColor = vec4 ( iz_boja, 1.0f ); \n"
        "} "
```

Izvor: [1].

7. SPREMNICI VRHOVA

Spremnici vrhova u OpenGL-u služe za spremanje podataka u grafičkoj memoriji (GPU). To se postiže stvaranjem memorije na GPU-u gdje se pohranjuju podaci, daju upute kako OpenGL treba interpretirati memoriju i određuje kako poslati podatke na grafičku karticu. Korištenje spremnika je dio modernog OpenGL-a koji se primjenjuje u programu [1].

Spremnici se koriste na način da je napravljeno jedno polje spremnika u koje se mogu ubaciti podaci pa onda pozivati i koristiti po potrebi. Isto tako se koriste indeksirani spremnici radi bolje izvedbe i izbjegavanja stvaranja nepotrebnih podataka koji se ponavljaju i opterećuju memoriju.

7.1. Varijable

U programu se koriste spremnici vrhova i indeksi spremnici vrhova. Spremnici vrhova su pohranjeni u polje *VAO*.

Tablica 4.: Varijable koje se koriste u spremnicima:

NAZIV VARIJABLE	TIP	OPIS
VAO	Gluint	Služi kao ID varijabla za polje spremnika
VBO	Gluint	Služi kao ID varijabla za spremnik vrhova
IBO	unsigned int	Služi kao ID varijabla za indeksirani spremnik vrhova

Izvor: rad autora

7.2. VAO i VBO spremnici vrhova

U programu se napravi polje spremnika *VAO* na koje se onda veže *VBO* spremnik vrhova.

VBO-u se dodjele podaci preko polja vrhova u kojima se nalazi pozicija i boja modela. Program koristi za svaki objekt svoj spremnik vrhova *VBO* i polje vrhova *VAO*. Preko *VAO* daju se upute na koji način će se koristiti podaci koji se nalaze u spremniku vrhova. U glavnom programu treba se samo pozvati *VAO* prije crtanja objekta [5].

Funkcijama glVertexAttribPointer se objašnjava na koji način će se čitati podaci iz polja. Za već spomenuti model ptice koristi se samo pozicije vrhova i boja. Prvi poziv funkcije određuje na koji način će se tumačiti pozicije vrhova. Prvi argument je ID koji se koristi u programu za sjenčanje. Drugi koliko podataka u polju čine jedan vrh, zatim kojeg su tipa. Predzadnji argument govori o veličini podatka vezanih za jedan vrh dok zadnji od koje pozicije u polju počinju pozicije vrhova. Isto tako drugi poziv funkcije govori o boji vrhova s time da ima svoj ID i zadnji argument pokazuje da počinje od 3 podatka u polju (nakon pozicije vrhova) [6].

Kod 13.: Prikaz koda koji se koristi za stvaranje spremnika vrhova modela ptica i na koji način se koriste u programu

```
        GLuint VBO, VAO;
        glGenVertexArrays(1, &VAO);
        glGenBuffers(1, &VBO);

        // na ovaj vao vežemo spremnik
        glBindVertexArray(VAO);
        glBindBuffer(GL_ARRAY_BUFFER, VBO);
        glBufferData(GL_ARRAY_BUFFER, sizeof(poljePtica[0].vrhovi),
poljePtica[0].vrhovi, GL_STATIC_DRAW);

        // kako čitamo podatke iz spremnika, pozicija i boja
        // Position attribute
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat),
(GLvoid *)0);
        glEnableVertexAttribArray(0);
        // color attribute
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat),
(GLvoid *)(3 * sizeof(GLfloat)));
        glEnableVertexAttribArray(1);
```

Izvor: [1].

7.3. IBO spremnik

Program se koristi i indeksiranim spremnikom vrhova, tako da ne bude puno podataka koji bi se ponavljali. Na spremnik *IBO* se veže polje indeksi u kojemu su podaci koji su jedinstveni. Na taj način se ubrzava izvođenje programa. Iz poziva funkcije se vidi da ima 1515 vrhova za prikaz modela galeba u programu.

Kod 14.: Prikaz koda za stvaranje indeksiranog spremnika vrhova koji se koristi u programu

```
unsigned int IBO;

glCreateBuffers(1, &IBO);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, 1515 * sizeof(unsigned int), indeksi,
GL_STATIC_DRAW);
```

Izvor: [1].

8. ZAKLJUČAK

Sam let modela ptica koristi algoritam roja čestica koji ima određena ograničenja i prednosti. Pretraga je, ako se pravilno primjeni algoritam, dosta jednostavna, no pretragu postaviti unutar neke cjeline sa prikazom je nešto drugo. Većina programa za prikaz rada algoritma koristi jednostavne modele čestice kao što su točka ili crtica. Na taj način se pojednostavljuje prikaz. S druge strane ako se odabere neki konkretni objekt, kao u ovom radu onda se moraju primijeniti pravila koja vrijede za taj objekt. Na taj način se prikaz komplicira i sama izvedba programa traži puno više prilagodbe i optimizacije da bi prikaz bio realniji. Ptice se kreću na određeni način. Tijelo ptice uvijek gleda u pravcu kretanja, nema oštrih zaokreta. Jato komunicira i djeluje kao cjelina. To su neka pravila koja su primijenjena u programu.

Potrebno je stvoriti prostor i okruženje u kojem će se jato ptica kretati. Koncept stvaranja grafičkog programa u modernom OpenGL-u je zahtjevan i složen. Potrebno je odrediti i stvoriti 3D prostor, odrediti na koji način koristiti spremnike, programe za sjenčanje i teksturu. OpenGL je izvrsna grafička knjižnica s kojom se mogu razumjeti upravo ovakvi koncepti jer se na neki način moraju stvoriti. Isto tako programski jezik C++ se pokazao kao izvrstan i pogodan jezik za ovaj zadatak. Budući da je zadatak bio dosta kompleksan, napravljen je na najjednostavniji način. Optimizacija u pogledu načina pisanja koda i samih performansi programa je potrebna, ali to je stvar budućih nadogradnji i poboljšanja.

Program se može proširiti u nekoliko pravaca. Ako se želi poboljšanje u pravcu pretrage onda se može dodati više lokacija ribe i ponavljajuću pretragu koja bi se vršila na raznim lokacijama i širila ovisno o rezultatima lokalne pretrage. Isto tako bi se mogle dodati opcije utjecaja na razne parametre i time poboljšavati ili usmjeravati samu pretragu. Dodavanjem evolucijskih algoritama na jedinke proširiti program u nekim smjerovima koji nisu dio ovog završnog rada ali su vrijedni spomena. Sam let ptica isto tako traži poboljšanja tako da se one mogu kretati kružno i različitim

brzinama. Problem kružnog kretanja u 3D prostoru pokazao se dosta kompleksan zadatak pa je ostavljen za daljnja poboljšanja.

Smisao cijelog projekta je bio shvatiti na koji način se može napraviti grafička simulacija nekog prirodnog procesa koristeći algoritam roja čestica. Tako da se razumiju glavne sastavnice modernog grafičkog prikaza ali i primjeni automatizirana pretraga prostora pomoću algoritma. Stoga, može se reći kako su glavni zadani ciljevi ostvareni.

LITERATURA

[1]. De Vries, Joey, Learn OpenGL, https://learnopengl.com (datum pristupa 01.09.2020)

[2]. Chernikov, Jan, The Cherno, https://www.youtube.com/user/TheChernoProject (datum pristupa 01.09.2020)

[3]. Ćutić, Denis. (2014) Programsko ostvarenje algoritama rojeva čestica, Zagreb 2014

[4]. Čupić, Marko. (2012) Prirodom inspirirani optimizacijski algoritmi. Metaheuristike., Zagreb 2012

[5] Sonar System, Modern OpenGL 3.0+ Tutorials, https://www.youtube.com/c/SonarSystemsCoUk/videos (datum pristupa: 01.09.2020)

[6]. Rodriguez, Jorhe and co., docs.GL, OpenGL API Documentation, http://docs.gl/ (datum pristupa 01.09.2020)

[7]. Poli, Riccardo. (2008) "Analysis of the publications on the applications of particle swarm optimisation." Journal of Artificial Evolution and Applications 2008 : 3.

Popis slika

Slika 1.: Dijagram tijeka rada programa.....7

Slika 2.: Ptice su nasumično raspoređene i započinju pretragu.....11

Slika 3: Ptice su pronašle ribu.....12

Slika 4.: 3D model galeba koji se koristi u programu.....12

Popis kodova

Kod 1.: Prikaz koda perspektivne projekcije.....8

Kod 2.: Prikaz koda za model i pogled.....9

Kod 3.: Prikaz koda za uzimanje lokacije iz programa za sjenčanje.....9

Kod 4.: Prikaz koda predavanje vrijednosti u program za sjenčanje.....10

Kod 5.: Prikaz koda teksture.....10

Kod 6.: Prikaz koda funkcije trazuRibu kojom se jedinke koriste tražeći ribu.....15

Kod 7.: Prikaz koda funkcije leti, kojom se jedinke koriste za let.....17

Kod 8.: Prikaz koda funkcije odrediSusjeda.....18

Kod 9.: Prikaz koda funkcije azurirajVrijednosti.....19

Kod 10.: Prikaz koda funkcije odrediKut.....20

Kod 11.: prikaz koda procesora vrhova koji se koristi u programu.....23

Kod 12.: prikaz koda procesora fragmenata koji se koristi u programu.....24

Kod 13.: Prikaz koda koji se koristi za stvaranje spremnika vrhova i na koji način se koriste u programu.....26

Kod 14.: Prikaz koda za stvaranje indeksiranog spremnika vrhova koji se koristi u programu.....27

Popis tablica

Tablica 1.: Varijable u klasi Ptica.....13

Tablica 2.: Varijable u klasi Riba.....21

Tablica 3.: Varijable koje se koriste u programima jedinicama za sjenčanje.....22

Tablica 4.: Varijable koje se koriste u spremnicima.....25

SAŽETAK

Glavna zamisao ovog rada je da se pokuša spojiti algoritam roja čestica i grafička simulacija rada samog algoritma u 3D prostoru na primjeru jata ptica kako pretražuje prostor u potrazi za ribom. Simulacija je napravljena u programskom jeziku C++, grafičkom sučelju OpenGL i razvojnom okruženju Visual studio 2017. Primijenjeni su osnovni dijelovi i koncepti koji su potrebni da se napravi 3D prostor, koristi moderni OpenGL, primjenjuje sam algoritam i grafički 3D model galeba.

KLJUČNE RIJEČI

algoritam roja čestica ARČ, C++, OpenGL, simulacija, grafički prikaz, optimizacija, 3D prostor

Abstract

The main idea of this work is to try to connect the algorithm of the particle swarm and the graphical simulation of the work of the algorithm itself in the 3D space on the example of birds searching the space in search of fish. The simulation was made in the C++ programming language, the graphical interface of OpenGL and the Visual Studio 2017 development environment. Basic parts and concepts that are needed to make 3D space are applied, using modern OpenGL, the algorithm itself and graphic 3D model of Seagull.

KEYWORDS

Particle swarm optimization PSO, C++, OpenGL, simulation, graphical representation, optimization, 3D space