Izrada preglednika za 3D modele uporabom OPENGL-a i C++ programskog jezika

Kakuk, Antun

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni preddiplomski studij računarstva

IZRADA PREGLEDNIKA ZA 3D MODELE UPORABOM OPENGL-A I C++ PROGRAMSKOG JEZIKA

Završni rad

Antun Kakuk

Osijek, 2018.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 26.09.2018.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Antun Kakuk
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3649, 21.09.2017.
OIB studenta:	18374562560
Mentor:	Doc.dr.sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Izrada preglednika za 3D modele uporabom OPENGL-a i C++ programskog jezika
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	26.09.2018.
Datum potvrde ocjene Odbora:	27.09.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.09.2018.

Ime i prezime studenta:	Antun Kakuk
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3649, 21.09.2017.
Ephorus podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Izrada preglednika za 3D modele uporabom OPENGL-a i C++ programskog jezika**

izrađen pod vodstvom mentora Doc.dr.sc. Časlav Livada

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Računalna grafika prešla je ogroman put od vlastitih začetaka i jednostavnih prikaza na katodnim monitorima pa do današnjih foto realističnih prikaza na suvremenim zaslonima visokih rezolucija. U današnje vrijeme računalna grafika primjenjuje se u mnogim multimilijunskim industrijama, kao na primjer u medicini, u filmskoj industriji, u vojnim sustavima, u industriji videoigara, u CAD sustavima i drugima. Kako bi se pobliže proučile moderne metode koje se koriste u računalnoj grafici kao tema ovog rada odabrana je izrada preglednika za trodimenzionalne modele. Za izradu preglednika koristi se *Open Graphics Library* odnosno OpenGL aplikacijsko programsko sučelje (engl. API, *Application programming interface*), programski jezik C++ te GLEW i GLFW programske biblioteke (engl. *library)*. *Microsoft Visual Studio 2017* poslužit će kao integrirano razvojno okruženje. Tijekom izrade rada potrebno je obratiti pozornost na tijek iscrtavanja (engl. *rendering pipeline*), budući da se slika iscrtava (engl. *render*) po koracima u točno određenom slijedu. U drugom poglavlju nalazi se opis alata i tehnologija korištenih u izradi završnog rada, dok je u trećem poglavlju dan opis izrade završnog rada.

1.1. Zadatak završnog rada

Završni rad obuhvaća uspostavljanje i konfiguraciju integriranog razvojnog okruženja nakon kojeg slijedi izrada preglednika trodimenzionalnih modela. Za izradu samog preglednika potrebno je poznavati tijek iscrtavanja i sva korištena aplikacijska programska sučelja te se također koriste znanja iz različitih matematičkih disciplina.

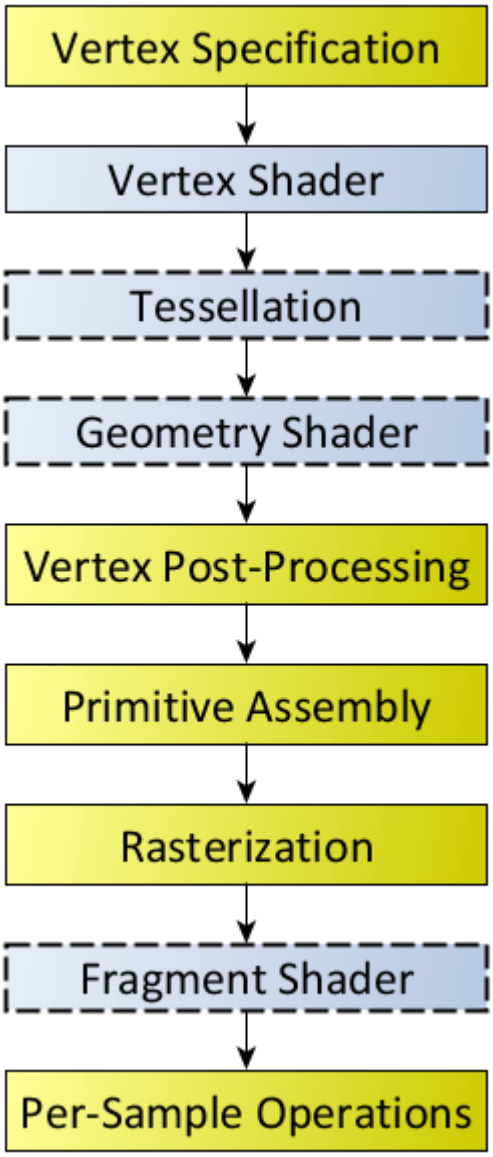
2. ALATI I TEHNOLOGIJE

Pri izradi ovog rada koristi se nekolicina aplikacijskih programskih sučelja, programskih knjižnica i integriranih razvojnih okruženja. U narednim potpoglavljima opisat će se ti alati i tehnologije.

2.1. OpenGL

Open Graphics Library (OpenGL) jest sučelje koje omogućuje programskim jezicima komunikaciju s grafičkim očvrsjem (engl. *hardware*) da bi se pomoću očvrsja ubrzalo iscrtavanje (engl. *hardware-accelerated rendering*). Iako je primarna svrha OpenGl-a implementacija na razini očvrsja, postoji mogućnost implementacije na razini programske podrške (engl. *software*). Vrlo je fleksibilan te se može implementirati na više različitih računalnih platformi. Nije ovisan o programskom jeziku u kojem se vrši implementacija niti o mrežnim protokolima . Koristi se za iscrtavanje (engl. *rendering*) dvodimenzionalne i trodimenzionalne računalne vektorske grafike. Razvijen je 1992. godine od strane *Silicon Graphics Inc.* kao otvorena alternativa IrisGL-u.

Za uspješno savladavanje koncepata OpenGL-a potrebno je vrlo dobro razumjeti tijek iscrtavanja. Tijek iscrtavanja (Sl. 2.1.) točno je određeni niz koraka kroz koji svaki objekt iscrtavanja prolazi kako bi se prikazao na zaslonu. Dio očvrsja koji je zadužen za tijek iscrtavanja grafička je kartica računala. Tijek iscrtavanja izvodi se u sljedećim koracima:



Slika 2.1. Tok iscrtavanja [1]

1. Određivanje vrhova (engl. *Vertex Specification*)

   Vrh (engl. *vertex*) je točka u prostoru, najčešće određena koordinatama x, y i z ako se radi o trodimenzionalnom prostoru, odnosno x i y ako se radi o dvodimenzionalnom prostoru. Jedan ili više vrhova zajedno čine jednostavne oblike (engl. *primitives*) koji čine kompleksnije modele. Kao jednostavni oblici u pravilu se koriste trokuti, no moguće je koristiti točke, linije ili pravokutnike. U ovom koraku se unosi slijed podataka koji određuju vrhove koje ćemo koristiti u iscrtavanju jednostavnih oblika. U tu svrhu koriste se objekt niza vrhova (engl.

VAO, *Vertex Array Object*) i objekt međuspremnika vrhova (engl. VBO, *Vertex Buffer Object*) te će se u daljnjem navođenju koristiti engleske kratice. Pomoću VAO definiramo koju strukturu podataka vrh sadrži (pozicija, boja, normala, itd.) dok pomoću VBO definiramo same podatke. Za jedan VAO moguće je vezati više VBO. Svi ovi podatci spremaju se na radnu memoriju grafičke kartice kako bi im grafička kartica što brže pristupila. Također, dio određivanja vrhova su i pokazivači na atribute (engl. *Attribute Pointers*) koji određuju gdje i kako shaderi (engl. *shaders*) mogu pristupiti podatcima o vrhovima. Naposljetku, nakon što definiramo sve podatke potrebno je odrediti koje shadere koristiti, vezati VAO koji želimo iscrtati te pozvati *glDrawArrays* metodu za iscrtavanje koja započinje ostatak tijeka iscrtavanja.

2. Obrada vrhova (engl. *Vertex Processing*)

   Nakon što se odrede vrhovi i inicijalizira iscrtavanje započinje korak obrade vrhova. U koraku obrade vrhova vrhovi prolaze kroz shader vrhova (engl. *Vertex Shader*) koji je obavezan potkorak te popločavanje (engl. *Tessellation*) i shader geometrije (engl. *Geometry Shader*) koji su neobavezni. Shader vrhova rukuje zasebnim vrhovima, a kao ulaz prima podatke vrha. Možda najbitnija stavka jest što unutar shadera vrhova varijabla *gl_Position* mora biti definirana jer se prosljeđuje kao izlaz i koristi u daljnjim koracima. Ovaj shader se najčešće koristi ako želimo primijeniti matrice pogleda (engl. *View Matrix*) i matrice projekcije (engl. *Projection Matrix*).

   Popločavanje je relativno novi shader i u ovom radu se neće koristiti. Služi za dijeljenje jednostavnih oblika na manje dijelove, na primjer pravokutni jednostavni oblik na dva trokuta. Može služiti za dinamično povećanje detalja i stvaranje realističnijih prikaza površina materijala.

   Shader geometrije kao ulaz uzima jednostavan oblik, a kao izlaz može izbaciti taj isti jednostavni oblik ili na njemu izvršiti razne izmjene poput mijenjanja podataka, stvaranja novih jednostavnih oblika ili izmjene vrste jednostavnih oblika.

3. Naknadna obrada vrhova (engl. *Vertex Post-Processing)*

   Prvi od dva podkoraka je transformacija povratne informacije (engl. *Transform Feedback*). Ako je poznato da se određeni vrhovi i jednostavni oblici neće mijenjati u ovom

podkoraku moguće ih je pohraniti u međuspremnik. Pohranom u međuspremnik podatci vrhova i jednostavnih oblika ne moraju ponovno prolaziti ranije korake i time se štedi na resursima. Ovaj podkorak nije obavezan i može se koristiti po želji.

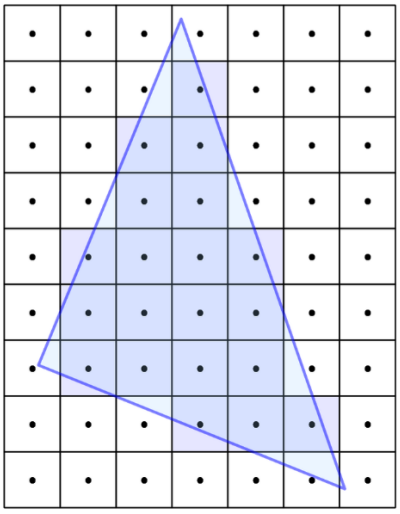
Idući podkorak jedan je od važnijih i naziva se isijecanje (engl. *Clipping*). U ovom podkoraku izbacuju se jednostavni oblici koji nisu vidljivi, odnosno ne nalaze se unutar volumena pogleda (engl. *view volume*), budući da se ne želi trošiti resurse na iscrtavanje onoga što se ne može vidjeti. Nadalje, u ovom koraku također se provodi pretvorba položaja iz prostora isijecanja (engl. *clip-space*) u prostor prozora(engl. *window space*).

4. Sabiranje jednostavnih oblika (engl. *Primitive Assembly*)

   U koraku sabiranja jednostavnih oblika događaju se dvije važne stvari. Prva je da se sve matematički redefinira kao jednostavni oblik. Ulazni niz vrhova se po određenim pravilima pretvara u izlazni slijed jednostavnih oblika. Dakle, ako je kao oblik iscrtavanja uzet trokut, niz od šest vrhova postat će niz od dva trokuta. Druga je odstrjel lica (engl. *Face culling*). Nakon što su svi vrhovi grupirani u jednostavne oblike potrebno je odrediti koja je površina geometrijskog lika vidljiva te označiti one koji nisu kako se ne bi iscrtavali. Time se štedi na računalnim resursima.

5. Rasterizacija (engl. *Rasterization*)

   Prikaz slike na zaslonima pikseliziran je, a pojedini piksel nedjeljiva je komponenta koja trenutno može poprimiti isključivo jednu vrijednost, što znači da se prostor piksela nalazi u diskretnoj domeni. S druge strane, prostor vektora kontinuiran je, stoga vektori mogu poprimiti bilo koju vrijednost te se tako može dogoditi da pojedini vektor presijeca pojedini piksel (Sl. 2.2.).



Slika 2.2. Prikaz dobivanja fragmenta iz jednostavnog oblika [4]

Time se stvara potreba za određivanjem koji piksel pripada kojem jednostavnom obliku. Proces kojim se pomoću interpolacije određuje kojem se pikselu pridružuju koje vrijednosti naziva se rasterizacija. Korak rasterizacije kao ulazni parametar prima jednostavan oblik proslijeđen iz koraka sabiranja jednostavnih oblika te ih pretvara u skupine piksela koji se dodjeljuju određenom jednostavnom obliku, odnosno pretvara ih u fragmente ili dijelove. Fragmenti su izlazni parametar ovog koraka. Za svaki jednostavni oblik stvara se barem jedan fragment, no moguće ih je stvoriti i više.

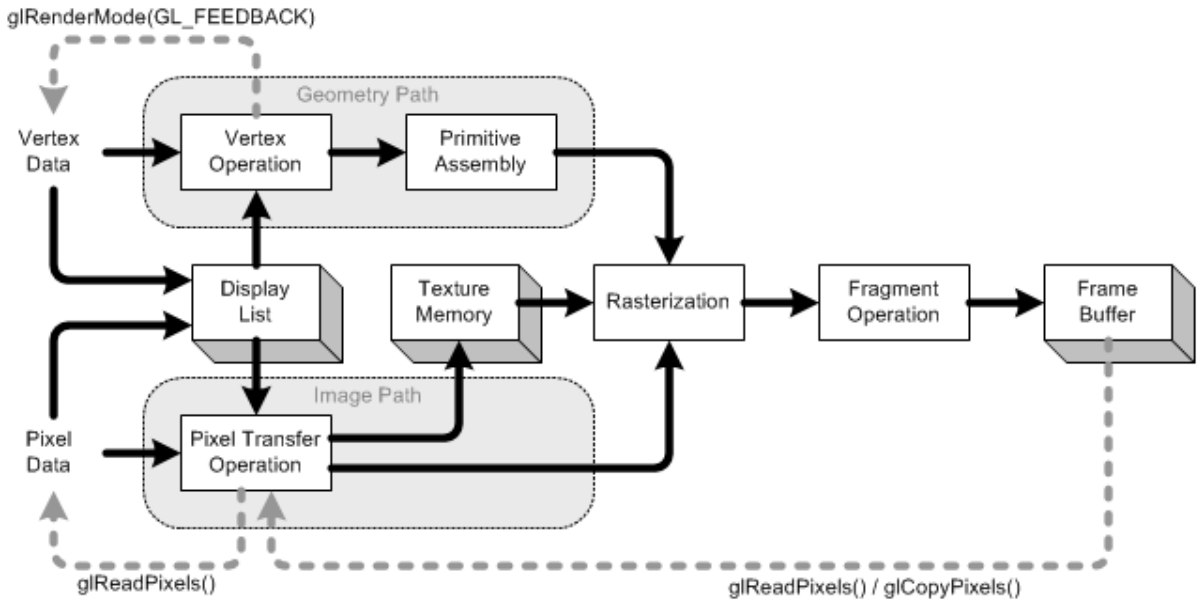
6. Shader fragmenata (engl. *Fragment Shader*)

Nakon rasteriziranja jednostavnih oblika, svaki dobiveni fragment moguće je provući kroz shader fragmenata. Shader fragmenata kao ulazni parametar uzima izlazni fragment iz koraka rasterizacije, nad njim vrši korisnički definirane operacije i prosljeđuje ga posljednjem koraku. Iako nije obavezan korak, shader fragmenata se u pravilu gotovo uvijek koristi, budući da je neophodan za osvjetljenje, rukovanje teksturama i sjene.

7. Obrada po uzorku (engl. *Per-Sample Processing*)

Obrada po uzorku zapravo je niz testova nad fragmentima kojima se određuje hoće li se uopće neki fragment iscrtati na zaslonu ili ne. Također, iscrtava se fragmente u trenutno vezani

međuspremnik okvira (engl. *Framebuffer*). U testu vlasništva piksela (engl. *pixel ownership test*) odbacuju se svi fragmenti pokriveni nekim drugim prozorom. Kod testa škara (engl. *scissor test*) odbacuju se svi fragmenti koji su izvan trenutno vezanog međuspremnika okvira. Operacije s više od jednog uzorka (engl. *multisampling*) nam omogućuju zaglađivanje rubova mnogokuta (engl. *polygon*). Test dubine (engl. *depth test*) odbacuje sve fragmente koji su iza drugih fragmenata pa ih nije potrebno iscrtavati. Stapljanje (engl. *blending*) određuje boju piksela kod fragmenata s određenim stupnjem prozirnosti. Završetkom ovog koraka tijek iscrtavanja kreće iznova. Alternativni prikaz toka iscrtavanja dan je na (Sl. 2.3.).



Slika 2.3. Tok iscrtavanja (alternativni prikaz) [5]

2.2. Programski jezik C++

C++ je programski jezik srednje razine za općenitu namjenu te pronalazi primjenu u širokom spektru područja. Definira se kao jezik srednje razine jer obuhvaća mogućnosti jezika viših i nižih razina. Istovremeno je proceduralan, funkcijski i objektno orijentiran jezik. Preteča C++ je C s klasama, koji je kao i C++ razvio Bjarne Stroustrup u *AT&T Bell Labs* kao unaprijeđenu verziju C programskog jezika. U 1983. godini C s klasama preimenovan je u C++, a dvije godine kasnije izlazi i prva inačica programa. Međunarodna organizacija za standardizaciju normirala je C++ 1998. godine. Najvažnija razlika između C++ u odnosu na C programski jezik je dodana mogućnost korištenja klasa.

Dodatkom klasa C++ postaje objektno orijentirani jezik i dobiva četiri nove značajke koje su isključive objektno orijentiranim jezicima: uopćavanje (engl. *abstraction*), učahurivanje (engl. *encapsulation*), nasljeđivanje i višeobličje (engl. *polymorphism*). Pomoću uopćavanja mogu se stvoriti jednostavni modeli iz složenih objekata kako bi se proučavale ili kako bi se koristile one komponente složenih objekata koje su nam važne, a izbacile one koje nisu.

2.3. *Microsoft Visual Studio 2017*

Microsoft Visual Studio 2017 integrirano je razvojno okruženje koje omogućuje razvoj računalnih programa, mrežnih stranica, mrežnih aplikacija, mobilnih aplikacija i mrežnih usluga. Podržava preko trideset programskih jezika među kojima su i neki od najpopularnijih poput C, C++, C++/CLI, C#, JavaScript, Visual Basic .NET, HTML, CSS te mnogi drugi. Značajke *Microsoft Visual Studija* koje se koriste u ovom radu su uređivač koda (engl. *code editor*) i program za ispravljanje grešaka (engl. *debbuger*). Neke su od ostalih značajki: dizajner za programe s grafičkim sučeljem (engl. *forms designer*), dizajner za klase, dizajner za baze podataka i mrežni dizajner.

2.4. GLEW

OpenGL Extension Wrangler Library (GLEW) višeplatformska je biblioteka otvorenog koda za C/C++ programske jezike [2]. Budući da se u modernim verzijama OpenGL-a (verzija 3.2+) API funkcije određuju prilikom pokretanja programa, a ne tijekom prevođenja programa (engl. *compiling*) GLEW umjesto korisnika upravlja pokazivačima na te funkcije i inicijalizira ih te time skraćuje dugotrajan i zamoran proces koji je podložan ljudskim greškama. Nakon što inicijalizira dostupne funkcije sprema ih u datoteku zaglavlja za daljnje korištenje. Korištenje GLEW biblioteke omogućuje vrlo jednostavno određivanje podržanih proširenja (engl. *extensions*) od strane upravljačkog programa grafičke kartice. Trenutno postoji više od tristo različitih proširenja, dok pojedine grafičke kartice podržavaju više od sto proširenja.

2.5. GLFW

Prema [3] *Graphics Library Framework (*GLFW) višeplatformska je biblioteka za OpenGL, OpenGL ES i Vulkan na desktop računalima. Pruža jednostavno sučelje za stvaranje i upravljanje prozorima, kontekstima i površinama te se uz to koristi za primanje i rukovanje unosima (engl. *inputs*) i događajima (engl. *events*). Iako postoje mnoge biblioteke koje objedinjuju funkcije ove biblioteke, kao i neke dodatne funkcije, ova biblioteka odabrana je jer je najlakša po računalne resurse (engl. *lightweight*).

2.6. GLM

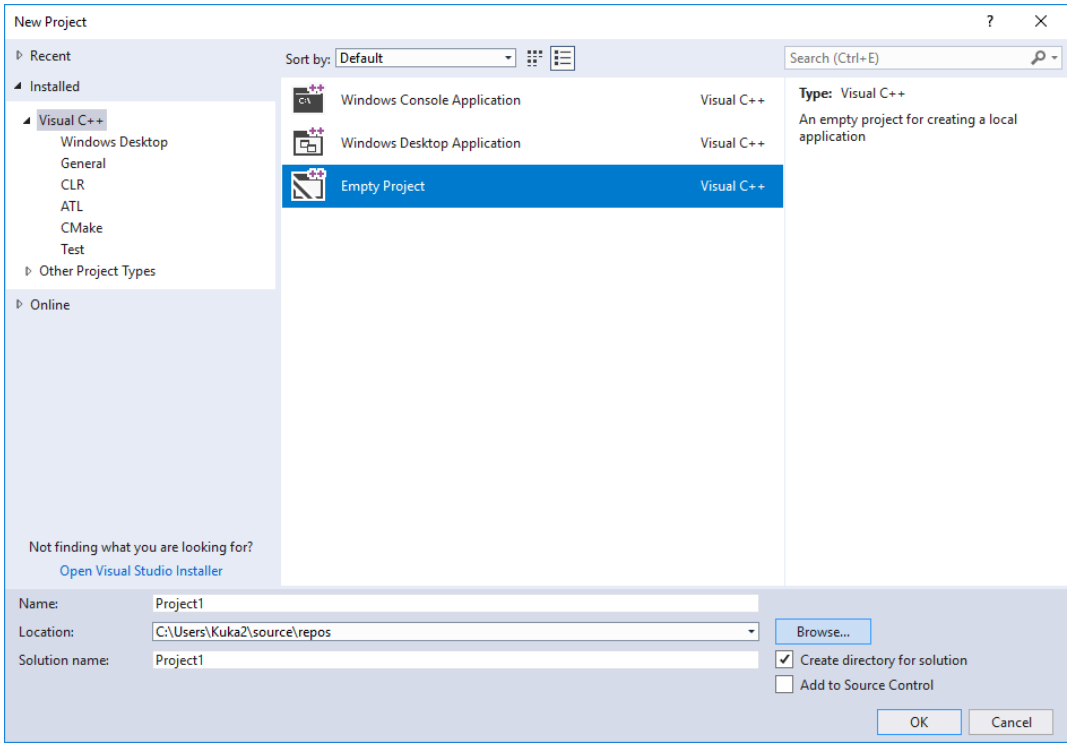
GLM (*OpenGL Mathematics*) je matematička knjižnica za grafičku programsku podršku temeljena na OpenGL jeziku za sjenčanje (engl. *OpenGL Shading language*), nadalje GLSL [7]. Ova knjižnica omogućuje korištenje složenih matematičkih funkcija unutar c++ programskog jezika. Imena svih matematičkih funkcija su temeljena na GLSL-u. U završnom radu koristiti će se za matrične transformacije.

2.7. *Open Asset Import Library*

Open Asset Import Library je knjižnica otvorenog koda koja služi uvozu raznih dobro poznatih formata 3D modela [9]. Napisana je u C++ programskom jeziku, no moguće ju je koristiti i u raznim drugim programskim jezicima. *Open Asset Import Library* nastoji pružiti mogućnost potpune konverzije raznih vrsta 3D modela, što znači da je sposobna omogućiti učitavanje modela baš kao i njihov izvoz. Namijenjena je korištenju u širokom spektru aplikacija.

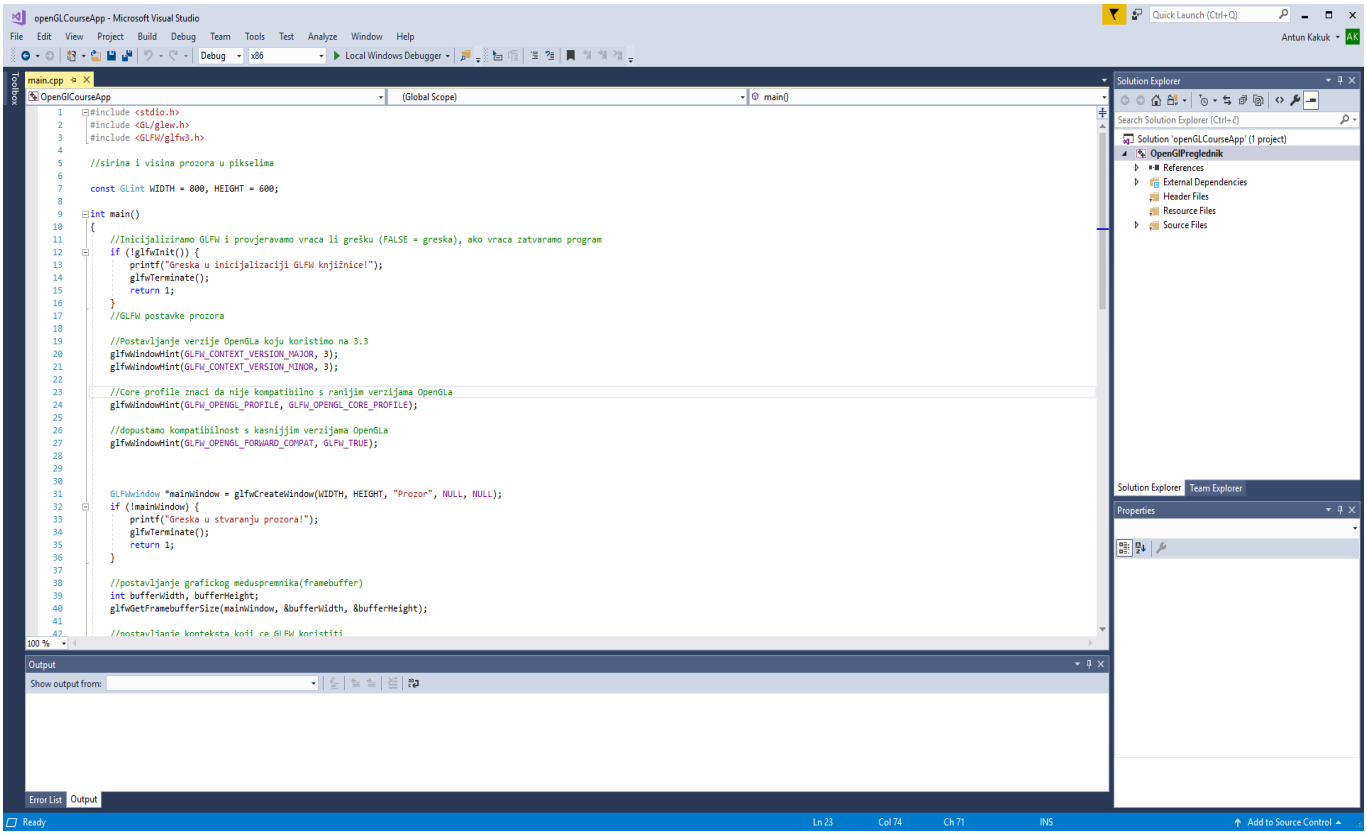
3. IZRADA PREGLEDNIKA ZA 3D MODELE

Kako bi se uopće pristupilo implementaciji samog OpenGL-a potrebno je prvo kreirati novi projekt unutar odabranog razvojnog okruženja. Novi projekt unutar *Microsoft Visual Studio 2017* razvojnog okruženja kreira se klikom na File >> New >> Project… ili kombinacijom tipki Ctrl + Shift + N čime se otvara novi prozor (Sl.3.1.) u kojemu možemo konfigurirati parametre novog projekta.



Slika 3.1. Prozor za kreiranje novog projekta.

Neki od parametara koje je moguće konfigurirati vrsta su projekta, ime projekta, lokacija spremanja projekta i ime rješenja projekta. Kao vrstu projekta odabire se prazan projekt, dok su ostali parametri proizvoljni. Nakon kreacije projekta prikazuje se korisničko sučelje (Sl.3.2.). Kako bi bilo moguće koristiti odabrane knjižnice potrebno ih je prvo povezati s projektom u kojemu se koriste. Nakon što su knjižnice povezane sve je spremno za rad.



Slika 3.2. Korisničko sučelje Microsoft Visual Studia 2017

3.1. Stvaranje prozora i iscrtavanje jednostavnog oblika

Na samom početku izrade potrebno je inicijalizirati GLFW pomoću naredbe *glfwInit()* kojom se omogućuje korištenje svih GLFW funkcionalnosti. Nakon inicijalizacije potrebno je postaviti verziju openGL-a na kojoj radimo pomoću funkcije *glfwWindowHint()*. Prozor kreiramo naredbom *glfwCreateWindow()* koji prima argumente visina i širina prozora u pikselima i naziv prozora. Nakon toga je potrebno odrediti veličinu međuspremnika naredbom *glfwGetBufferSize()*, postaviti glavni prozor u trenutni kontekst naredbom *glfwMakeContextCurrent()*. Kontekst u sebi sadrži sva stanja trenutne instance OpenGL-a.

Kako bi se iscrtao jednostavan oblik potrebno je koristiti već spomenute VAO i VBO objekte te uz njih vezati podatke o vrhovima. Naredbe kojima stvaramo i manipuliramo VAO i VBO objektima su *glGenVertexArrays()*, *glBindVertexArray()*, *glGenBuffers()* i *glBindBuffer()*.

Također je potrebno napisati jednostavne verzije shadera vrhova i shadera fragmenata. Shaderi su svojevrsni kod u kodu jer su definirani kao niz podataka tipa *char* koje se zatim u OpenGL-u prevodi i veže uz objekt tipa program koji ih povezuje s grafičkom karticom. Budući da su shaderi prije prevođenja samo niz podataka tipa *char* moguće ih je učitati iz obične tekstualne datoteke, gdje se često i spremaju radi olakšane izmjene. Uniformne varijable su vrsta varijabli u shaderu, općenite su za shader te nisu vezane za neki određeni vrh. Svaka uniformna varijabla ima lokacijski identitet (engl. *location* ID) u shaderu. Potrebno je pronaći lokaciju kako bi se s njom povezala vrijednost. Uniformne varijable služe prosljeđivanju globalnih podataka shaderu.

3.2. Kamera, prikaz i transformacije

3D modeli se unutar OpenGL-a spremaju kao niz vrhova koji se prikazuju kao vektori unutar određenog koordinatnog sustava. Različiti koordinatni sustavi koji se koriste u OpenGLu nazivaju se prostori. Kako bi se vektorima moglo manipulirati u koordinatnom sustavu te kako bi bilo moguće prelaziti iz jednog koordinatnog prostora u drugi potrebno je uvesti matrične transformacije. Tako razlikujemo matrične transformacije translacije, skaliranja i rotacije. Za razumijevanje matričnih transformacija potrebno je osnovno poznavanje linearne algebre.

Translacija podrazumijeva pomicanje vektora, služi mijenjanju položaja objekta. Ako se želi promijeniti položaj objekta s određene točke u prostoru to se čini na način da se matrica položaja množi s translacijskom matricom, a dobiveni rezultat, tj. vektor označava novu poziciju objekta.

Skaliranjem se mijenja veličina vektora, koristi se kako bi se objekt povećao ili smanjio. Skaliranje se ne upotrebljava često jer, na primjer, povećavanjem nekog objekta dolazi do grafičkih manjkavosti.

Rotacija je pomicanje objekta oko zamišljene osi koja najčešće prolazi kroz ishodišnu točku objekta. Zamišljenu os u OpenGL-u definiramo kao vektor. Kod kombiniranja matričnih transformacija važan je redoslijed množenja matrica.

Svaki prostor posjeduje vlastite koordinate s vlastitim ishodištem (Sl. 3.3.). Prostori unutar OpenGL-a su kako slijedi:

1) Prostor modela ili lokalni prostor (engl. *Model space, Local space*)

Svaki model ima svoje vlastito ishodište, najčešće u središtu modela. Svi vrhovi modela se gledaju u odnosu na to ishodište. Svaki model započinje u lokalnom prostoru. Kako bi koordinate modela transformirale u koordinate prostora svijeta primjenjuje se matrica modela.

2) Prostor svijeta (engl. *World-space*)

Unutar prostora svijeta su globalne koordinate. Više modela može se nalaziti u jednom prostoru svijeta. Udaljenost modela od ishodišta je zapravo udaljenost ishodišta modela od ishodišta prostora svijeta. Koristeći matricu pogleda, koordinate prostora svijeta se transformiraju u koordinate prostora pogleda.

3) Prostor pogleda ili kamere (engl. *View space*)

Kamera procesuira scenu, ono što se gleda, odnosno ono što se vidi u prostoru pogleda. Prostor pogleda je koordinatni sustav sa svakim vrhom vidljivim iz kamere i sve se promatra u odnosu na kameru. Da bi se prostor svijeta pretvorio u prostor pogleda koristi se matrica pogleda. Kako bi izračunali matricu pogleda potrebno je poznavati četiri vrijednosti: poziciju kamere, smjer kamere (ono gdje je usmjerena, vektor smjera usmjeren je u obrnutom smjeru od pretpostavljenog smjera), vektor usmjeren desno od kamere, to jest x-os. Umnoškom matrica vektora smjera i vektora usmjerenog desno dobije se vektor usmjeren gore. Kamera smješta vrijednosti u matrice i računa matricu pogleda. Matricu pogleda koristimo kako bi vrhove iz prostora svijeta prebacili u prostor pogleda. Matricu pogleda moguće je izračunati pomoću naredbe iz glm knjižnice na sljedeći način:

glm::mat4 viewMatrix = glm::lookAt(position, target, up);

pri čemu *position* označava koordinate kamere, *target* koordinate točke u koju se gleda, a *up* usmjerenje prostora svijeta prema gore. Kako bi se iskoristila matrica pogleda potrebno ju je vezati uz uniformnu varijablu shadera te ju primijeniti između matrice projekcije i matrice modela kako slijedi:

*gl_Position = projection * view * model * vec4(pos, 1.0).*

Kako bi se kamera pomicala koristi se *glfwGetKey()* naredba kojom se hvata događaj pritiska određene tipke na tipkovnici te se u skladu s pritisnutom tipkom kamera pomiče u odabranom smjeru (iako se u stvarnosti unutar OpenGl-a svijet pomiče u odnosu na kameru).

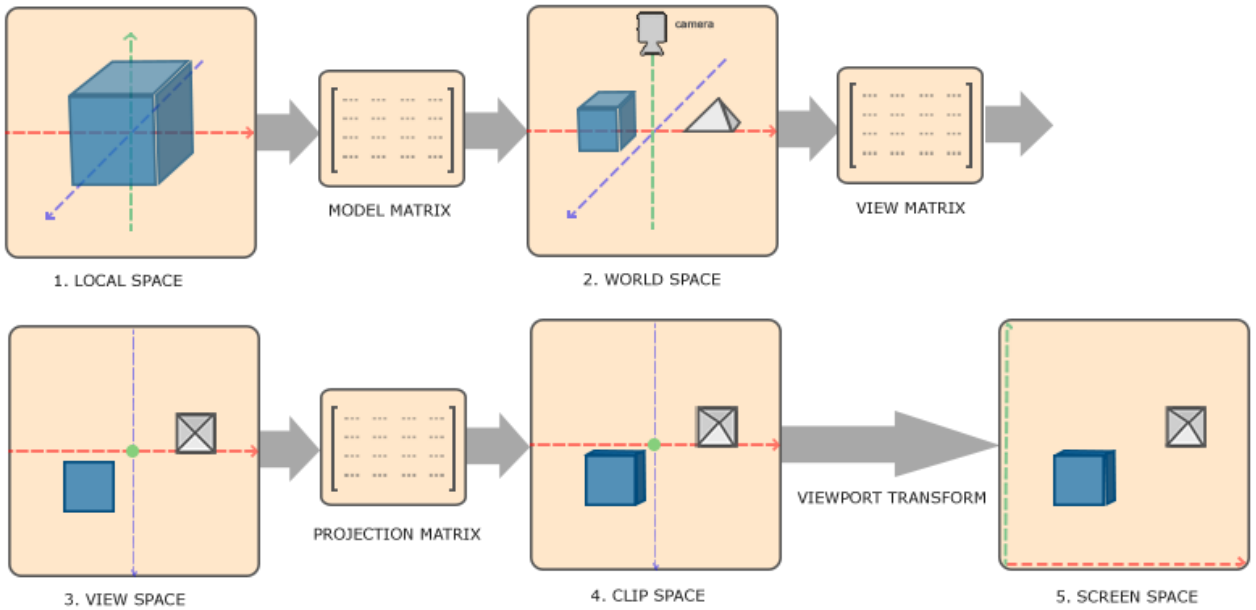
4) Prostor projekcije (engl. *Projection space*)

Prostor projekcije je definiran kao volumen. Sve izvan tog volumena se isijeca. Dvije najčešće korištene projekcije su ortografska i perspektivna. Ortografska se koristi kod dvodimenzionalnih prikaza dok se projekcijska koristi kod trodimenzionalnih prikaza zbog svoje sposobnosti prikaza dubine. Transformacijom okvira pogleda dobije se prostor zaslona. Za izračun projekcijske matrice u završnom radu se koristi naredba:

glm::mat4 = glm::perspective(fov, aspect, near, far).

5) Prostor zaslona

Koordinate iz ovog prostora šalju se na rasterizaciju.



Slika 3.3. Prikaz prostora u OpenGlu[6]

3.3. Teksture

Teksture su slike koje se koriste kako bi se dodali dodatni detalji nekom objektu. Ipak, teksture ne moraju uvijek biti slike, mogu sadržavati i razne druge vrste podataka, no to pripada mnogo složenijoj razini. Najčešće su teksture u 2D obliku, ali također mogu biti i 1D i 3D teksture. Točke na teksturama nazivaju se tekseli, ne pikseli, a određeni su između 0 i 1. Stvaranje tekstura otprilike odgovara stvaranju VBO-a/VAO-a (*glGenTextures(), glBindTexture()*). Teksture se vežu na objekte tekstura. Kako bi se uštedjelo na resursima koristi se mipmap, odnosno dalje teksture se iscrtavaju s manjom rezolucijom od bližih.

3.4. Osvjetljenje

Phongov model osvjetljenja sastoji se od tri dijela: ambijentalnog osvjetljenja, difuznog osvjetljenja i reflektirajućeg osvjetljenja. Ambijentalno osvjetljenje sastoji se od svjetla koje je uvijek prisutno, čak i ako je izravan put od izvora svjetla na bilo koji način blokiran. Ambijentalno osvjetljenje, u odnosu na ostale, najjednostavniji je koncept svjetlosti. Ono simulira odbijanje svjetlosti od druge objekte. Stvaranje faktora ambijentalnog osvjetljenja:

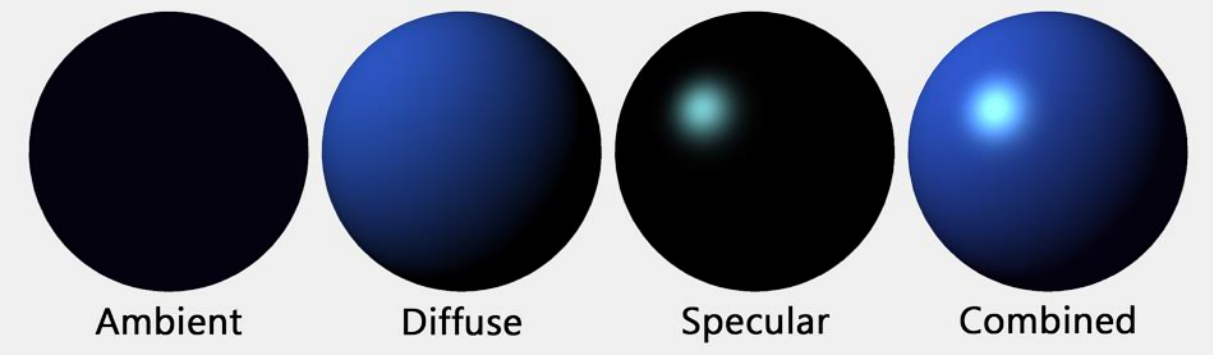
$$ambient = lightColour * ambientStrength.$$

Taj faktor pokazuje koliko boje fragmenta ambijent ovog svjetla prikazuje *fragColour * ambient*. Ako je ambijent 1 onda je fragment uvijek potpuno osvijetljen. Ukoliko je ambijent 0 onda je fragment uvijek crn. Ako je ambijent 0.5 onda je fragment u pola intenziteta svojeg uobičajenog intenziteta boje.

Difuzno osvjetljenje određeno je smjerom izvora svjetlosti, ono stvara efekt nestajanja što se više udaljava od izvora svjetlosti. Složenije je od ambijentalnog osvjetljenja. Ono simulira smanjenje svjetla kako kut svjetla postaje manji. Strana koja je okrenuta direktno izvoru svjetla snažnije je osvijetljena, ako se ostvaruje pod određenim kutom bit će slabije osvjetljena. Može se koristiti kut između vektora koji spaja izvor svjetlosti s fragmentom te vektora koji je pod pravim

kutom u odnosu na lice odnosno normalom.

Reflektirajuće osvjetljenje je svjetlo koje se neometano reflektira od izvora svjetlosti do promatračeva oka, načelno gledano može se reći da se oslanja na položaj promatrača. To je direktan odraz izvora svjetla usmjerenog prema oku promatrača. Pomicanje će utjecati na poziciju refleksije površine. Ovisi o koeficijentu sjajnosti, što je materijal sjajniji refleksija je jača. Razliku između različitih vrsta osvjetljenja možemo vidjeti na (Sl. 3.4.)



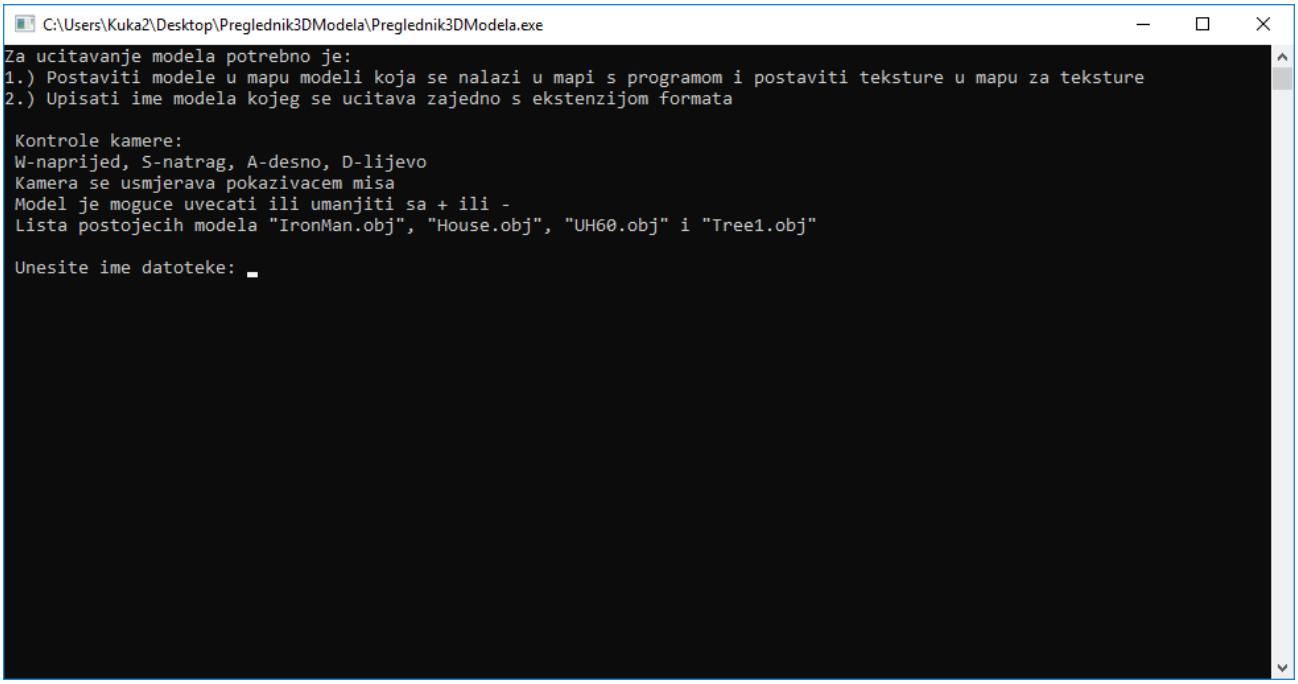
Slika 3.4. Razlika između različitih tipova osvijetljenja [8]

Vrste svjetlosti

a) Direkcijsko svjetlo je svjetlo koje nema svoju poziciju ili izvor. Svo svjetlo dolazi iz paralelnih zraka iz naočigled beskrajne udaljenosti. Najbolji primjer je Sunce.
b) Točkasti izvor svjetlosti svjetlo je s položajem koje svijetli u svim smjerovima, a najbolji primjer toga je žarulja.
c) Usmjereno svjetlo slično je točkastom svjetlu, ali ono je "odsječeno" kako bi emitiralo u određenom rasponu i pod određenim kutom, kao na primjer ručna svjetiljka.
d) Prostorno svjetlo napredniji je oblik svjetla koje emitira svjetlost iz nekakvog područja, recimo veliki osvjetljeni panel na stropu ili zidu.

3.5. Izgled korisničkog sučelja

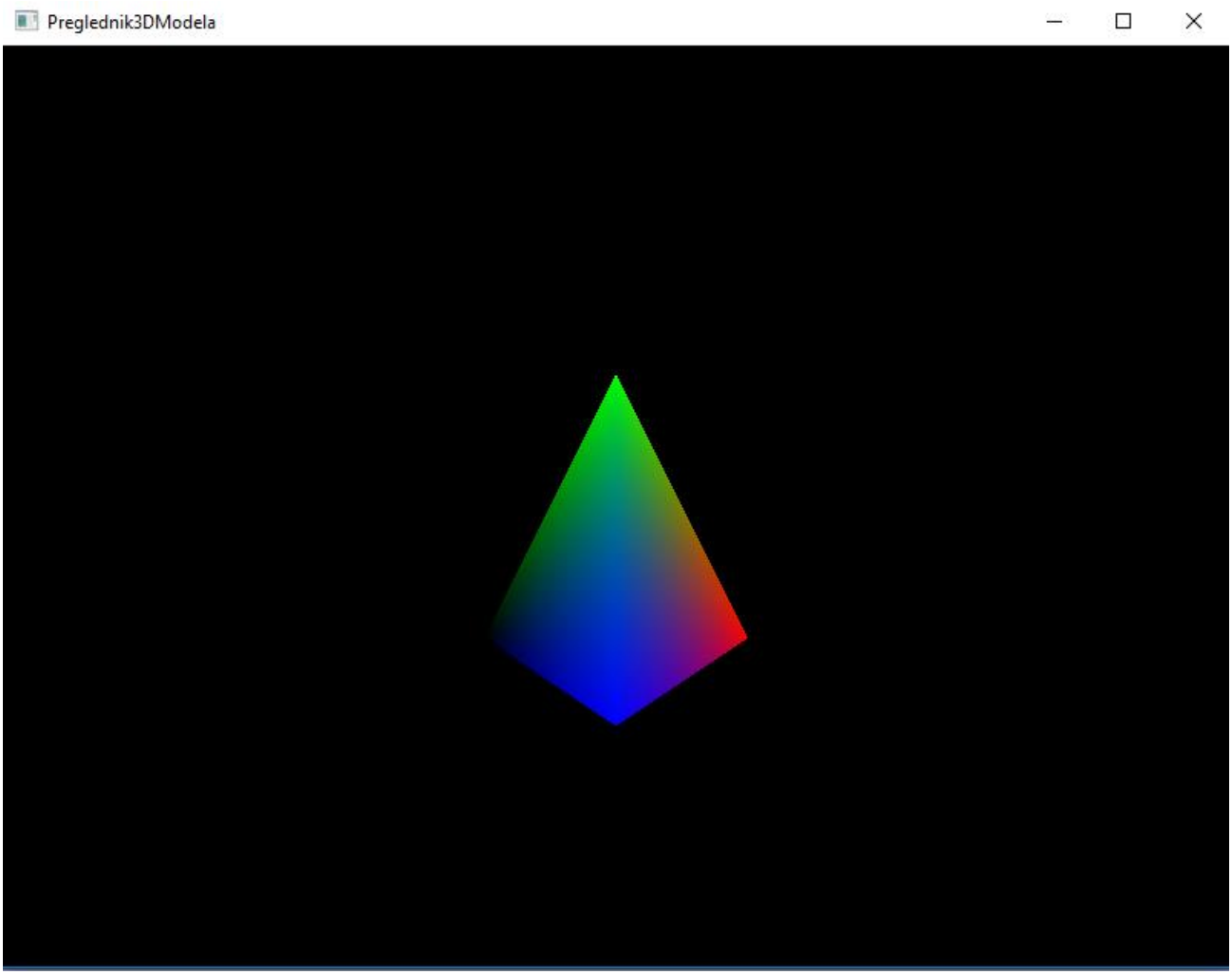
Korisničko sučelje izrađenog preglednika za 3D modele vrlo je jednostavno. Prilikom pokretanja programa pojavljuje se takozvana konzola (Sl. 3.5.), odnosno prozor zasnovan na tekstualnom unosu i prikazu. U prozoru je prikazana kratka poruka koja pojašnjava kako učitati 3D model u program i kako kontrolirati kameru jednom kada je model učitan. Model je moguće učitati tako da ga se prvo postavi u mapu namijenjenu za modele. U mapi je moguće držati više modela, a odabir modela vrši se unosom imena modela koji želimo učitati te ekstenzije formata u kojem je model pohranjen. Na primjer model *cajnik* u formatu *.obj* moguće je učitati upisivanjem *cajnik.obj* kako je prikazano na (Sl. 3.5.). Kako bi se model učitao također je potrebno postaviti teksture modela u mapu *Textures*.



Slika 3.5. Prikaz zaslona za unos 3D modela

Za učitavanje modela koristi se *Open Asset Import Library* koja podržava mnogo različitih formata zapisa 3D modela, no za ovaj završni rad odabran je *.obj* format. Nakon unosa imena modela model se učitava i prikazuje u novom prozoru kojim upravlja OpenGL (Sl.3.6.). Model je

moguće promatrati kroz kameru koju kontrolira korisnik pomoću tipki W, S, A, D i pokazivača miša. Kako bi se umanjili modeli koji su čija je veličina u prostoru svijeta prevelika ili umanjili onaj čija je veličina premala koriste se tipke + i – na numeričkom dijelu tipkovnice čime se skalira model.



Slika 3.6. Prikaz preglednika učitanog 3D modela

4. ZAKLJUČAK

U završnom radu izrađena je računalna aplikacija pomoću koje je moguće učitati datoteku trodimenzionalnog modela u samu aplikaciju te ju potom prikazati na zaslonu. U prvom dijelu ovog rada navedeni su svi alati korišteni pri izradi te je za svaki alat ponuđen njegov kratak opis. Za samu izradu računalnog programa korišten je OpenGL kao osnova za iscrtavanje računalne grafike. Zatim, korišten je C++ programski jezik pomoću kojega su implementirane OpenGL funkcionalnosti. GLEW i GLFW biblioteka olakšale su i skratile sam proces implementacije. Cjelokupan proces implementacije izveden je unutar *Microsoft Visual Studio 2017* integriranog razvojnog okruženja. Stvaranje preglednika zahtijevalo je određena znanja s područja matematike i fizike.

Završni rad bi se mogao unaprijediti dodatnim vizualnim efektima kao što su realistične sjene i bolja kalkulacija svijetla. Također bi bilo dobro kada bi preglednik imao bolje vizualno korisničko sučelje i podržavao učitavanje različitih formata 3D modela.

LITERATURA

[1] OpenGL, The Khronos Group Inc. – 21.9.2018.

Dostupno na: https://www.khronos.org/opengl/wiki/

[2] GLEW – 23.6.2018.

Dostupno na: http://glew.sourceforge.net/

[3] GLFW - 24.6.2018.

Dostupno na: http://www.glfw.org/

[4] Slika fragmenta – 20.9.2018.

Dostupno na https://gamedev.stackexchange.com/questions/8977/what-is-a-fragment-in-3d-graphics-programming

[5] Slika toka iscrtavanja – 20.9.2018.

http://www.songho.ca/opengl/gl_pipeline.html

[6] Prostori u OpenGLu – 15.9.2018.

Dostupno na https://learnopengl.com/Getting-started/Coordinate-Systems

[7] GLM knjižnica - 21.9.2018.

Dostupno na https://glm.g-truc.net/0.9.9/index.html

[8] Slika usporedbe osvjetljenja -20.9.2018.

Dostupno na: https://clara.io/learn/userguide/lighting_shading/materials/material_types/webgl_materials

[9] Assimp – 24.9.2018.

Dostupno na : http://www.assimp.org/index.php

SAŽETAK

U završnom radu dan je prikaz izrade preglednika za 3D modele. U tu svrhu korišteni su OpenGL i C++ programski jezik te nekoliko knjižnica otvorenog koda poput GLEW, GLFW i GLM knjižnice. Kako bi se program napravio bilo je potrebno vrlo dobro se upoznati sa načinom rada OpenGL-a i svim koracima tijeka iscrtavanja. Programski jezik C++ omogućio je programsku implementaciju OpenGL-a te integraciju raznih knjižnica otvorenog koda, dok je OpenGL omogućio komunikaciju programa s grafičkom karticom i time i samo iscrtavanje 3D modela na zaslon računala. Kroz samu izradu preglednika primjenjivana su razna znanja s područja matematike, fizike i linearne algebre usvojena na fakultetu. Izradom ovog preglednika dobivena je mogućnost učitavanja 3D modela raznih formata te njihov pregled unutar samog programa. Za pregled se koristi kamera kontrolirana korisničkim unosom i pokazivačem miša.

Ključne riječi: preglednik, 3D, trodimenzionalni, model, OpenGL, C++

DEVELOPMENT OF 3D MODEL VIEWER USING OPENGL AND C++ PROGRAMMING LANGUAGE

ABSTRACT

This final paper follows the development of the 3D model viewer. It was developed using OpenGL and C++ programming language and a few open source libraries like GLEW, GLFW and GLM libraries. To realize this program, it was necessary to familiarize with the inner workings of OpenGL and its rendering pipeline. Programming language C++ provided the means of implementation of OpenGL and the ability to include many popular open source libraries, while OpenGL provided tools to efficiently communicate with graphics card hardware enabling rendering 3D models on computer screen. Through the process of developing viewer there was a need to apply knowledge of mathematics, physics and linear algebra acquired throughout college education. By developing viewer, we received ability to load 3D models of different formats into the program and closer inspection of said models, once loaded. User controlled camera is used to inspect 3D models.

Keywords: viewer, 3D, model, OpenGL, C++

ŽIVOTOPIS

Antun Kakuk rođen je 10. ožujka 1993. godine u Požegi. Od 1999. do 2007. pohađa Osnovnu školu fra Kaje Adžića u Pleternici. Nakon završetka osnovne škole upisuje Katoličku klasičnu gimnaziju s pravom javnosti u Požegi od 2007. do 2011. godine. Te iste 2011. godine upisuje prvi studij, preddiplomski studij biologije na Odjelu za biologiju sveučilišta Josipa Jurja Strossmayera u Osijeku kojeg napušta nakon godinu dana studiranja. Nakon dvije godine upisuje preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku kojeg trenutno pohađa.