MODELIRANJE I MANIPULIRANJE 3D OBJEKATA S POMOĆU IMGUI BIBLIOTEKE

Jelenčić, Kristijan

VELEUČILIŠTE U KARLOVCU
STROJARSKI ODJEL
STRUČNI PRIJEDIPLOMSKI STUDIJ MEHATRONIKA

KRISTIJAN JELENČIĆ

MODELIRANJE I MANIPULIRANJE 3D OBJEKATA S POMOĆU IMGUI BIBLIOTEKE

ZAVRŠNI RAD

KARLOVAC, 2024. godina.

VELEUČILIŠTE U KARLOVCU
STROJARSKI ODJEL
STRUČNI PRIJEDIPLOMSKI STUDIJ MEHATRONIKA

KRISTIJAN JELENČIĆ

MODELIRANJE I MANIPULIRANJE 3D OBJEKATA S POMOĆU IMGUI BIBLIOTEKE

ZAVRŠNI RAD

mr. sc. Vedran Vyroubal, v. pred.

KARLOVAC, 2024. godina.

SAŽETAK

Ovaj rad se bavi **modeliranjem i manipulacijom 3D objekata s pomoću ImGui biblioteke** koristeći OpenGL kao osnovni grafički set pravila i protokola za iscrtavanje. Projekt omogućava korisnicima stvaranje i upravljanje 3D objektima, kao što su kocke, s mogućnošću interakcije s njima kroz translaciju, rotaciju i skaliranje. Korištenjem GLFW i ImGui omogućeno je upravljanje korisničkim sučeljem i stvaranje prozora u kojem se odvija grafička manipulacija. GLM biblioteka se koristi za matematičke operacije nad 3D objektima, dok datoteke za sjenčanje pružaju fleksibilnost u stvaranju svjetlosnih i teksturnih efekata. Implementacija kamere omogućava slobodno kretanje kroz 3D prostor, što korisnicima daje potpunu kontrolu nad scenom i objektima unutar nje. Projekt pruža osnovne funkcionalnosti za razvoj aplikacija koje zahtijevaju dinamično i fleksibilno upravljanje 3D objektima, te je lako proširiv za složenije scene i interakcije.

Ključne riječi: C++, OpenGL, ImGui, programiranje, iscrtavanje

SUMMARY

This work focuses on the **modeling and manipulating 3D objects using the ImGui library** and OpenGL as the primary set of graphics rules and protocols for rendering. The project allows users to create and manage 3D objects, such as cubes, with the ability to interact with them through translation, rotation, and scaling. By using GLFW and ImGui, user interface management and window creation for graphical manipulation are facilitated. The GLM library is utilized for mathematical operations on 3D objects, while shader files offer flexibility in creating lighting and texture effects. The implementation of a camera allows for free movement through 3D space, providing users with full control over the scene and the objects within it. The project offers basic functionality for developing applications that require dynamic and flexible management of 3D objects, and it is easily extendable for more complex scenes and interactions.

Keywords: C++, OpenGL, ImGui, programming, rendering.

POPIS SLIKA

SADRŽAJ

1. UVOD

Modeliranje i manipuliranje 3D objektima predstavljaju ključne aspekte u svijetu računalne grafike i dizajna. Ova tema obuhvaća razne tehnike i alate koji omogućuju stvaranje, prikazivanje i interakciju s trodimenzionalnim objektima u virtualnom okruženju. Sa razvojem tehnologije, mogućnosti modeliranja postaju sve sofisticiranije, omogućujući dizajnerima, programerima i umjetnicima da izraze svoju kreativnost na načine koji su ranije bili nezamislivi.

U današnjem digitalnom dobu, 3D modeliranje nije samo rezervirano za industriju video igara ili animacije; koristi se i u arhitekturi, medicini, obrazovanju, pa čak i u znanstvenim istraživanjima. Kroz upotrebu alata kao što su OpenGL, Blender i Unity, stručnjaci mogu stvarati realistične prikaze i simulacije koje obogaćuju naše iskustvo svijeta.

Ova tema fokusira se na osnove modeliranja 3D objekata, tehniku manipulacije, kao i alate i tehnologije koje se koriste u ovom procesu. Također će se razmotriti važnost teksturiranja, što dodatno doprinosi realizmu i estetskom doživljaju. Istražit ćemo kako kombinacija kreativnosti i tehničkih vještina može dovesti do inovativnih rješenja u dizajnu i prezentaciji 3D objekata.

3D objekti se modeliraju s pomoću matematičkih izraza i algoritama koji definiraju njihovu geometriju, površinu i teksture [1]. Jedan od ključnih aspekata u manipulaciji 3D objektima je razumijevanje koordinatnog sustava i transformacija, kao što su translacija, rotacija i skaliranje. Te transformacije omogućuju korisnicima da postave objekte u željene pozicije, promijene njihove dimenzije i prilagode ih kutovima gledanja u virtualnom prostoru. Kroz ove osnovne operacije, korisnici mogu stvoriti dinamične i interaktivne scene koje se lako mogu vizualizirati i koristiti u raznim primjenama.

Osim osnovne geometrije, teksturiranje ima ključnu ulogu u stvaranju vjerodostojnog izgleda 3D objekata. Teksture dodaju detalje i realizam koje jednostavne geometrijske površine same po sebi ne mogu pružiti. Pomoću tekstura, objekti mogu imati složene

vizualne karakteristike poput drveta, metala ili tkanine, što povećava razinu vizualne privlačnosti i korisničkog iskustva. Kombinacija teksturiranja i osvjetljenja, koje se kontrolira kroz dokumente sjenčanja, omogućuje postizanje efekata poput sjene, refleksije i prozirnosti, čime se dodatno poboljšava realizam scena [2].

Tehnologije poput OpenGL-a i GLM biblioteke olakšavaju proces manipulacije objektima kroz pružanje alata za matematičke operacije i grafički prikaz. OpenGL je univerzalni set pravila i protokola koji omogućuje direktnu kontrolu nad hardverom za iscrtavanje, dok GLM biblioteka osigurava precizne funkcije za matematičke operacije potrebne za transformacije objekata u 3D prostoru. Korištenjem ovih alata, programeri mogu optimizirati rad s 3D objektima i jednostavno integrirati manipulaciju kamerom, što omogućuje slobodno kretanje kroz prostor i bolju preglednost modeliranih scena [3].

Napredak u modeliranju i manipulaciji 3D objektima također otvara vrata za složenije primjene u raznim industrijama. Na primjer, u arhitekturi se koristi za virtualne ture zgrada, dok u medicini pomaže u stvaranju preciznih modela ljudskih organa. Razvoj ovih tehnologija omogućuje stvaranje inovativnih rješenja koja mogu unaprijediti procese u brojnim disciplinama, istodobno omogućujući korisnicima veću fleksibilnost i kreativnu slobodu.

2. TEORETSKE OSNOVE

Kako bi prvo znali dali je ovakav pothvat moguć ili ne, moramo se upitati dali je sustav ili programski jezik Turingov-potpuni (eng. "Turing complete"). To znači da može simulirati svaki algoritam koji se može izračunati, pod uvjetom da ima dovoljno vremena i resursa. Kada usporedimo ovaj projekt s Turingovim strojem, možemo reći da su svi ključni aspekti razvoja aplikacija za manipulaciju 3D objekata izvedivi jer se oslanjamo na sustave koji su Turingovi-potpuni. To znači da, teoretski, sve što je moguće izračunati može biti programirano unutar okvira jezika i tehnologija koje koristimo, kao što su C++ i OpenGL. U ovom projektu, koristimo različite elemente računalne grafike i matematičkih transformacija, no sve te operacije teoretski možemo simulirati na Turingovom stroju, što potvrđuje izračunljivu prirodu svih zadataka koje izvodimo. Naprimjer, algoritmi za translaciju, rotaciju i skaliranje 3D objekata temelje se na matematičkim operacijama koje su unutar dosega Turingovog stroja, što znači da su sve ove operacije moguće i unutar okvira teorije izračunljivosti. Zaključno, sustavi koje koristimo u ovom projektu omogućuju kompleksne operacije i simulacije upravo zato što su Turingovi-potpuni, što znači da teoretski mogu riješiti bilo koji problem za koji postoji algoritam, čime je ovaj projekt u stanju izvršavati bilo koju grafičku operaciju unutar mogućnosti hardvera i softverskih resursa.

Teorijske osnove ovog projekta temelje se na razumijevanju ključnih principa računalne grafike, kao i matematičkih i tehničkih koncepata koji omogućuju modeliranje, prikazivanje i manipulaciju 3D objekata. Projekt koristi nekoliko osnovnih komponenti računalne grafike, uključujući OpenGL, GLM biblioteku, ImGui sučelje i koncepte iz područja računalnog prikazivanja slika i 3D transformacija.

OpenGL omogućuje izravan pristup grafičkom hardveru, što ga čini idealnim za interaktivne aplikacije poput ove. S mogućnošću korištenja shadera, framebuffera i tehnika poput mapiranja tekstura, OpenGL osigurava visokokvalitetnu grafiku i omogućuje složene transformacije u stvarnom vremenu. Jedna od glavnih prednosti OpenGL-a je njegova platformska neovisnost, što znači da se može koristiti na različitim operativnim sustavima (Windows, macOS, Linux) i uređajima (računala, mobilni uređaji). Također, OpenGL je

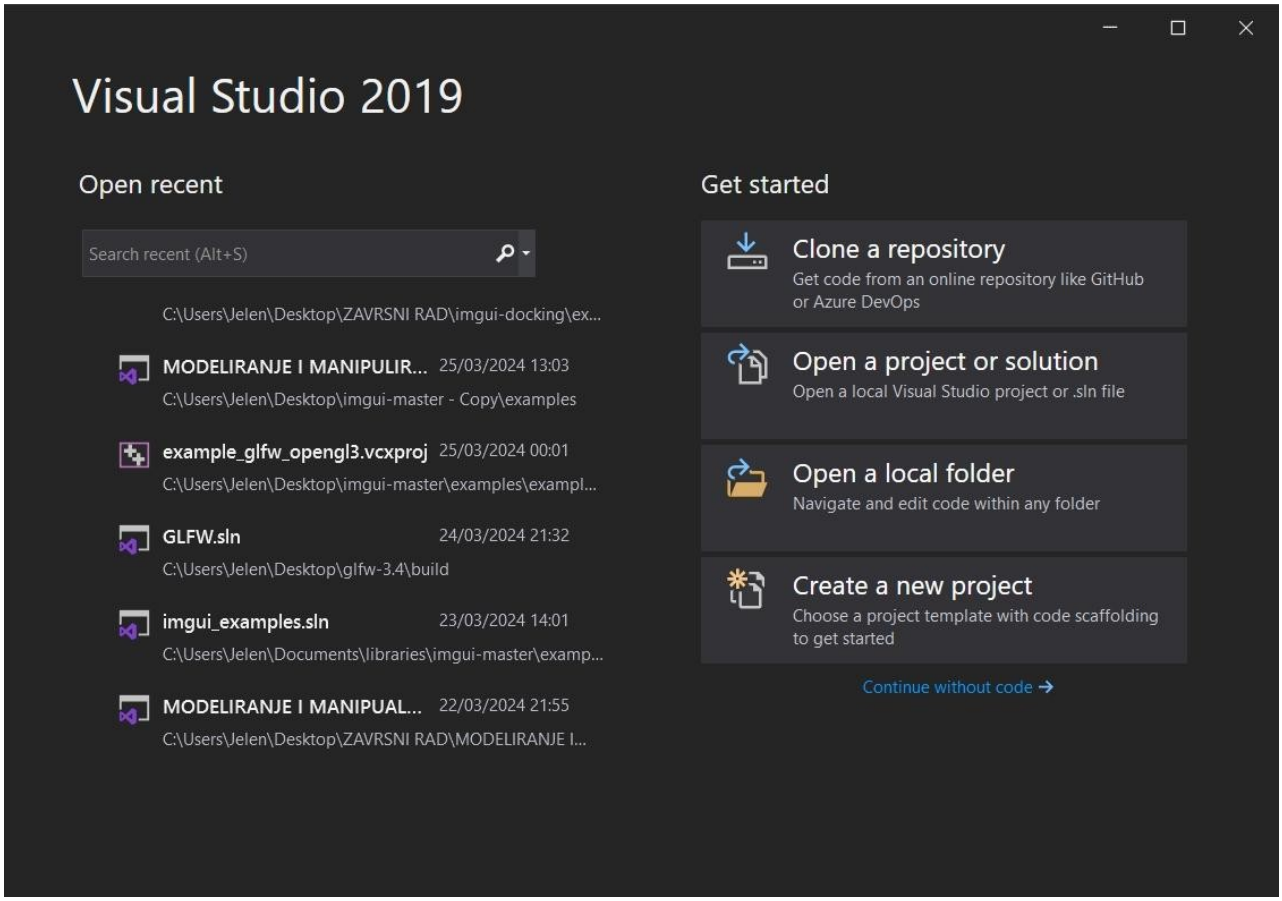
široko podržan u raznim industrijama, uključujući video igre, CAD softver, simulacije i druge aplikacije koje zahtijevaju visokokvalitetne grafičke prikaze [1].

OpenGL se često koristi zajedno s programskim jezicima poput C++ ili Python, a njegova arhitektura omogućuje prilagodbu različitim grafičkim procesorima, čineći ga fleksibilnim alatom za razvoj grafičkih aplikacija. U našem slučaju koristiti ćemo programski jezik C++ zbog učinkovitosti i brzine. Za programere, rad s OpenGL-om podrazumijeva razumijevanje pojmova poput koordinatnih sustava, shadera, framebuffera, te tehnika poput mapiranja tekstura i upravljanja perspektivom. Sveukupno, OpenGL predstavlja temeljni alat u svijetu računalne grafike, omogućujući stvaranje interaktivnih vizualnih iskustava s visokim stupnjem detalja i realizma [2].

3. POSTAVKA ZADATKA

3.1. INTEGRIRANO RAZVOJNO OKRUŽENJE

Kako bi mogli programirati potreban nam je prevoditelj, odnosno kompajler (eng. Compiler), jer za pisanje koda možemo u bilo kojem tekstualnom programu, a izbor integriranog razvojnog okruženja je veoma bitna odluka ovisno o potpori. Iako smo se odlučili koristiti Visual Studio 2019., svatko tko se osjeća ugodno s drugim razvojnim alatima, poput Code::Blocks, Eclipse ili čak terminala, može koristiti svoj preferirani alat. Bitno je razumjeti osnovne principe upravljanja projektima, organiziranja koda i kompilacije unutar vlastitog okruženja.



Slika 1: Visual Studio 2019

Za rad s vanjskim bibliotekama kao što su GLM, GLFW, GLEW i SOIL2, moramo postaviti naše projekte tako da ispravno povežu (eng. linking) ove biblioteke. Postoje dva načina povezivanja:

- Apsolutno povezivanje koristi apsolutne putanje na vašem računalu za lokaciju biblioteka. Ovo može biti nezgodno jer ako projekt preselite na drugo računalo ili promijenite lokaciju datoteka, putanja više neće biti valjana.

- Relativno povezivanje koristi putanje relativne prema lokaciji vašeg projekta, što omogućuje prenosivost projekta bez potrebe za prilagođavanjem putanja na svakom sustavu. Zbog praktičnosti i prenosivosti projekta, mi smo se odlučili za relativno linkanje biblioteka. Ovo znači da su sve biblioteke smještene unutar mapa unutar projekta (primjerice, unutar foldera 'libs') te je time omogućeno jednostavno dijeljenje projekta s drugim korisnicima ili korištenje na različitim računalima.

3.2. IZRADA PROZORA

GLFW je besplatna i otvorena biblioteka namijenjena upravljanju prozorima, OpenGL kontekstima te unosnim uređajima, poput tipkovnice i miša, u grafičkim aplikacijama. Zbog svoje jednostavnosti i efikasnosti, GLFW je osobito popularan u razvoju aplikacija temeljenih na OpenGL-u jer značajno olakšava upravljanje prozorima i unosnim uređajima u odnosu na druge grafičke okvire [4]. Dodatno, multiplatformska priroda GLFW-a omogućava jednostavno kreiranje aplikacija na različitim operativnim sustavima, kao što su Windows, macOS i Linux.

Ključne značajke GLFW-a uključuju:

• Upravljanje prozorima i OpenGL kontekstima.

• Podrška za višestruke monitore i način rada preko cijelog ekrana (fullscreen).

• Upravljanje unosom putem tipkovnice, miša i drugih uređaja, uključujući gamepad.

• Sinkronizaciju s vertikalnim osvježavanjem (V-Sync) radi izbjegavanja "trganja" slike.

• Fleksibilno upravljanje OpenGL kontekstom, omogućujući programerima potpunu kontrolu nad verzijama OpenGL-a.

Proces izrade prozora s GLFW-om:

1. **Inicijalizacija GLFW-a**
   Prije korištenja biblioteke potrebno je inicijalizirati GLFW kako bi se osiguralo odgovarajuće okruženje za stvaranje prozora i OpenGL konteksta.

2. **Postavljanje verzije OpenGL-a i opcija za prozor**
   Prije kreiranja prozora, postavljaju se parametri kao što su verzija OpenGL-a i odabir profila, poput "core" ili "compatibility".

3. **Kreiranje prozora**
   Nakon definiranja parametara, prozor se stvara pomoću funkcije glfwCreateWindow, koja vraća pokazivač na objekt prozora. U slučaju neuspjeha, potrebno je osloboditi sve alocirane resurse. Prilikom kreiranja, specificira se veličina prozora te naziv programa.

4. **Postavljanje OpenGL konteksta**
   Kada je prozor kreiran, potrebno je povezati OpenGL kontekst s njim kako bi se omogućilo izvršavanje OpenGL naredbi.

5. **Inicijalizacija GLEW-a (ili GLAD-a)**
   GLFW ne učitava OpenGL funkcije automatski, stoga je potrebno koristiti biblioteke poput GLEW-a ili GLAD-a za učitavanje potrebnih funkcija. Ovo je ključno za rad s modernim OpenGL funkcijama.

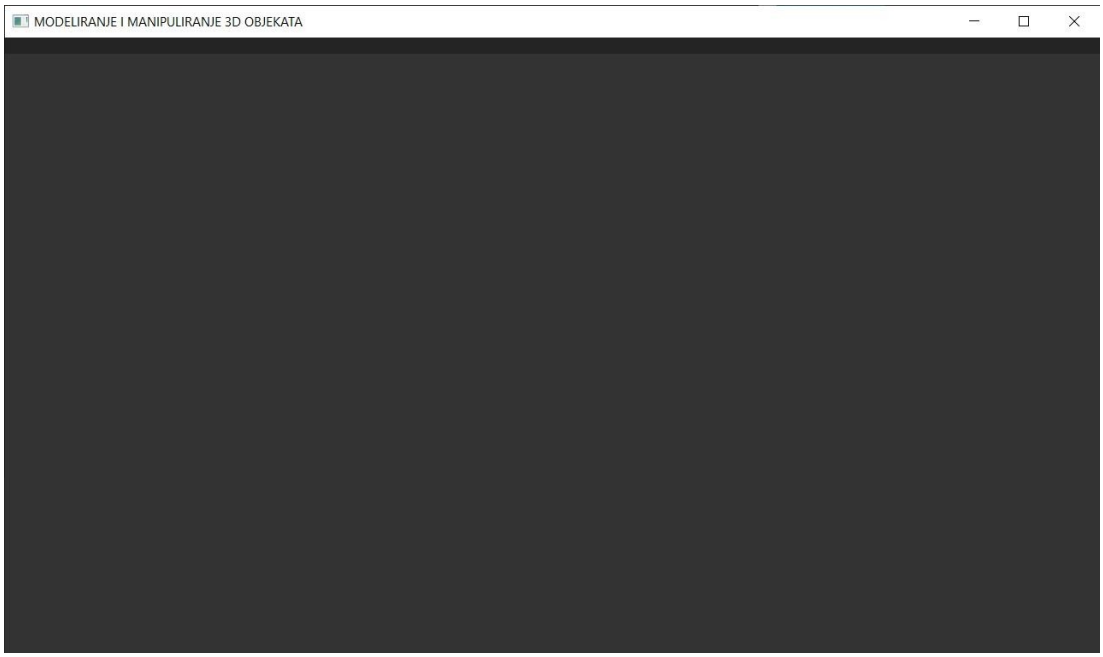
6. **Postavljanje petlje za iscrtavanje**

   Svaki program mora imati petlju za iscrtavanje kako bi se slika kontinuirano ažurirala. Funkcija glfwWindowShouldClose provjerava zatvara li korisnik prozor, dok glfwSwapBuffers zamjenjuje stražnji i prednji bafer radi glatkog prikaza.

7. **Zatvaranje i čišćenje resursa**

   Nakon što korisnik zatvori prozor, potrebno je očistiti sve resurse i završiti GLFW sesiju.

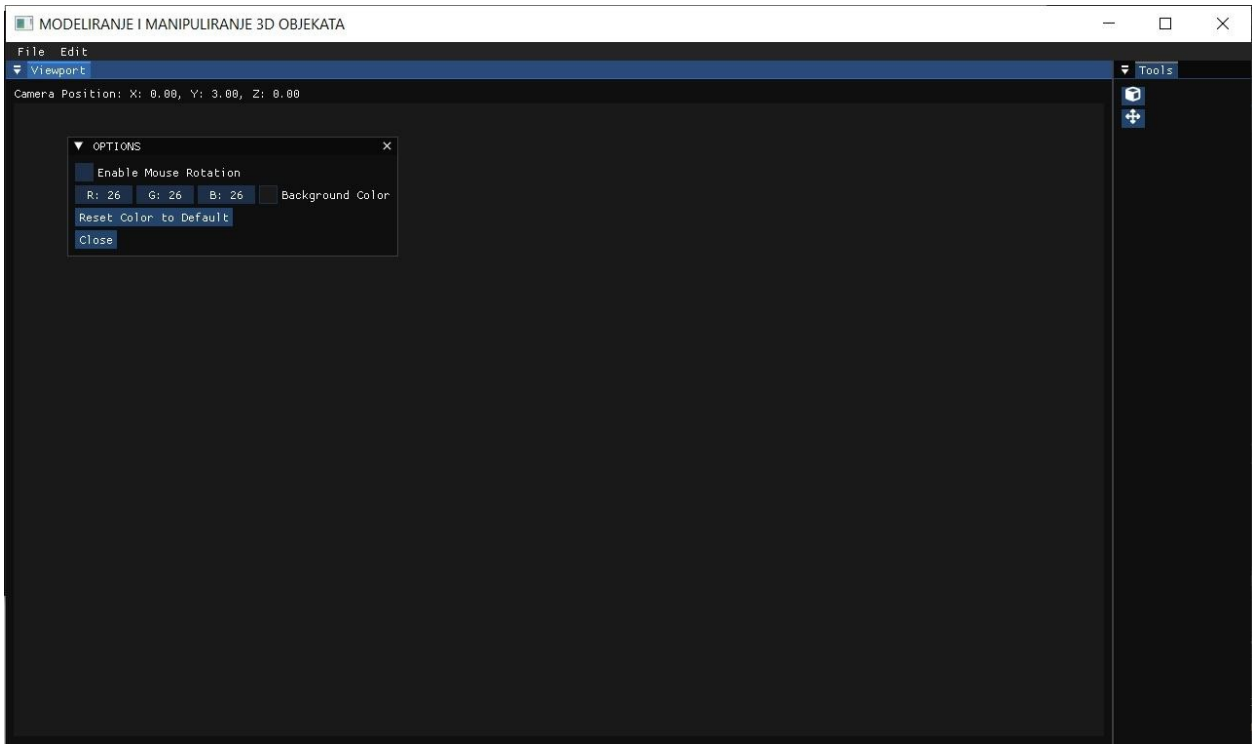
U našem slučaju, za postizanje funkcionalnog korisničkog sučelja, koristimo osnovni kod preuzet s GitHub-a. Stražnji dio koda zadužen je za pohranu funkcija i njihovo pozivanje kada je potrebno, te omogućuje pravilno funkcioniranje sučelja. Budući da smo odabrali GLFW kao glavni alat za upravljanje prozorima, sve ostale mape i metode iscrtavanja specifične za različite platforme (kao što su Windows ili mobilni uređaji) možemo ukloniti. Vulkan, kao modernija i složenija alternativa, za naše potrebe nije potrebna. Tako uspješno smo dobili prazan prozor za daljnji rad. Na takav prozor moguće je nadodati korisne funkcije.



Slika 2: Izrada prozora

3.3. IMGUI SUČELJE

ImGui je popularna biblioteka koja omogućava jednostavno i efikasno stvaranje grafičkih sučelja za aplikacije. Dizajniran za brz razvoj alata, omogućavajući programerima da s lakoćom dodaju grafičke elemente kao što su izbornici, gumbi, klizači, tekstualna polja i drugi widgeti, bez potrebe za složenim strukturama koje se koriste u tradicionalnim sučeljima. Jednostavnost ovog alata proizlazi iz toga da svaka komponenta GUI-ja postoji samo dok je iscrtavanje u trenutnoj petlji [5]. Također, moguće je dodati font koji omogućuje prikaz ikona umjesto teksta. Font “fa-solid-900” ćemo koristiti u našem projektu.



Slika 3: Prozor s ImGui grafičkim elementima

Ključne značajke ImGui-a:

- Intuitivan API: Lako je stvarati prozore, gumbe, tekstualna polja i druge elemente bez puno koda.

- Sidravanje (eng. Docking): Omogućava dinamičko postavljanje prozora u različite dijelove sučelja.

- Višestruki prozori: Moguće je otvarati manje prozore unutar glavnog prozora, poput dijaloških okvira ili kontrolnih panela.

- Interaktivni gumbi i alati: Lako se dodaju gumbi za akcije i kontrolne elemente poput klizača ili izbornih okvira.

- Prilagodba stila: Može se prilagoditi izgled prozora, boje i veličine elemenata, kako bi sučelje izgledalo profesionalno.

- Jednostavna integracija s OpenGL-om: Korisno u aplikacijama koje koriste OpenGL, jer ImGui može lako prikazivati sučelje preko prikazanih scena.

U ImGui-u se manji prozori mogu kreirati pomoću funkcije ImGui::Begin() i ImGui::End(). Svaki put kada želite prikazati novi prozor ili panel, možete pozvati ove funkcije kako biste definirali sadržaj i izgled tog prozora [5].

4. RAZRADA ZADATKA

Svaki program trebao bi biti zaštićen od neznanih i zlonamjernih korisnika. To su korisnici koji svojim postupcima, bilo namjerno ili nenamjerno, mogu izazvati probleme. To može uključivati prekid rada programa, netočne vrijednosti ili čak nemogućnost zatvaranja programa. Posebnu pažnju posvetit ćemo zaštiti od ovakvih problema. Korištenjem GLFW funkcija koje omogućuju unos s tipkovnice, možemo osigurati zatvaranje prozora pritiskom na tipku. Odabrali smo tipku 'Esc' za zatvaranje prozora zbog jednostavnosti korištenja.

4.1. DATOTEKE SJENČANJA

Nakon izrade prozora i sučelja, sada se možemo baviti grafikom. Koristimo datoteke za sjenčanje, poznate kao *shaderi* koje su ključni dio modernog OpenGL-a i koriste se za programiranje grafičkog procesora (GPU) kako bi se postigle specifične grafičke efekte. Datoteke za sjenčanje sadrže kod koji GPU izvršava za svaku fazu iscrtavanja objekata. *Shaderi* su napisani u jeziku GLSL (OpenGL Shading Language), a koriste se za prilagodbu načina na koji se vrhovi i pikseli obrađuju i prikazuju na ekranu [2].

U OpenGL-u se koriste različite vrste datoteka za sjenčanje, a najčešće su:

1. *Vertex shader*
    - Prvi korak u obradi objekta na GPU-u.
    - Operira na pojedinačnim vrhovima 3D objekata.
    - Njegov zadatak je transformirati pozicije vrhova iz lokalnog prostora objekta u odsječeni prostor.
    - Također može vršiti operacije poput transformacija, postavljanja teksturnih koordinata, normala i boja vrhova.

```
#version 330 core
layout (location = 0) in vec3 position;
//layout (location = 1) in vec3 color;
layout(location = 2) in vec2 texCoord;

uniform mat4 model; //local object coordinates to camera coordinates
uniform mat4 view; //normalized coordinates to window coordinates
uniform mat4 projection; //camera coordinates to normalized coordinates between 0 and 1

//out vec3 ourColor;
out vec2 TexCoord;

//uniform mat4 transform;

void main()
{
    gl_Position = projection * view * model * vec4(position, 1.0f);
    //gl_Position = transform * vec4(position, 1.0f);
    //ourColor = color;
    TexCoord = vec2(texCoord.x, 1.0 - texCoord.y);
}
```

Slika 4: Vertex shader

2. *Fragment shader*

- Operira na svakom pojedinačnom pikselu ili fragmentu, što omogućuje kontrolu nad bojom svakog piksela koji će biti iscrtavan na ekranu.

- Često se koristi za određivanje boja, primjenu tekstura, svjetlosnih efekata i slično.

```
#version 330 core
in vec2 TexCoord;

out vec4 color;

uniform sampler2D ourTexture1;

void main()
{
    color = texture (ourTexture1, TexCoord);
}
```

Slika 5: Fragment shader

Proces rada sa *shader* datotekama:

1. Pisanje *shader* koda: *Shaderi* se pišu zasebno, obično u datotekama s ekstenzijama ".vs" (za *vertex shader*) i ".frag" (za *fragment shader*). GLSL sintaksom podsjeća na programski jezik C, ali je prilagođen grafičkoj obradi.

2. Kompilacija i povezivanje: Nakon što su *shaderi* napisani, treba ih učitati u aplikaciju, kompilirati i povezati u *shader* program. *Shader* program je objekt u OpenGL-u koji sadrži oba *shadera* (*vertex* i *fragment shader*) te se koristi za crtanje objekata.

   Proces obično uključuje sljedeće korake:

   - Učitavanje koda shadera iz datoteka.
   - Kompilacija svakog shadera zasebno.
   - Povezivanje shadera u jedan program.

3. Upotreba *shader* programa: Nakon što su *shaderi* uspješno kompilirani i povezani, koristi se funkcija glUseProgram() kako bi se aktivirao shader program prije iscrtavanja objekta

4. Uniformi: *Shaderi* mogu primati ulazne podatke iz CPU-a putem uniformi. To su varijable čije vrijednosti ostaju iste za svaki vrh ili *fragment*, primjerice matrice za transformacije, boje ili teksture.

Kada koristimo ".png" teksture u OpenGL-u, često se koristi alfa kanal kako bismo postigli efekte transparentnosti. Alfa vrijednost određuje razinu prozirnosti svake točke (piksela) u teksturi, gdje vrijednost 0 znači potpuno prozirno, a 1 potpuno neprozirno [6].

Koristimo biblioteku poput SOIL2 za učitavanje ".png" datoteka koje podržavaju alfa kanal. SOIL2 automatski prepoznaje i učitava alfa kanal kada se radi o ".png" slikama.

Slika 6: Učitavanje alfa vrijednosti tekstura

Kako bismo postigli efekt transparentnosti, potrebno je uključiti alpha blending u OpenGL-u:

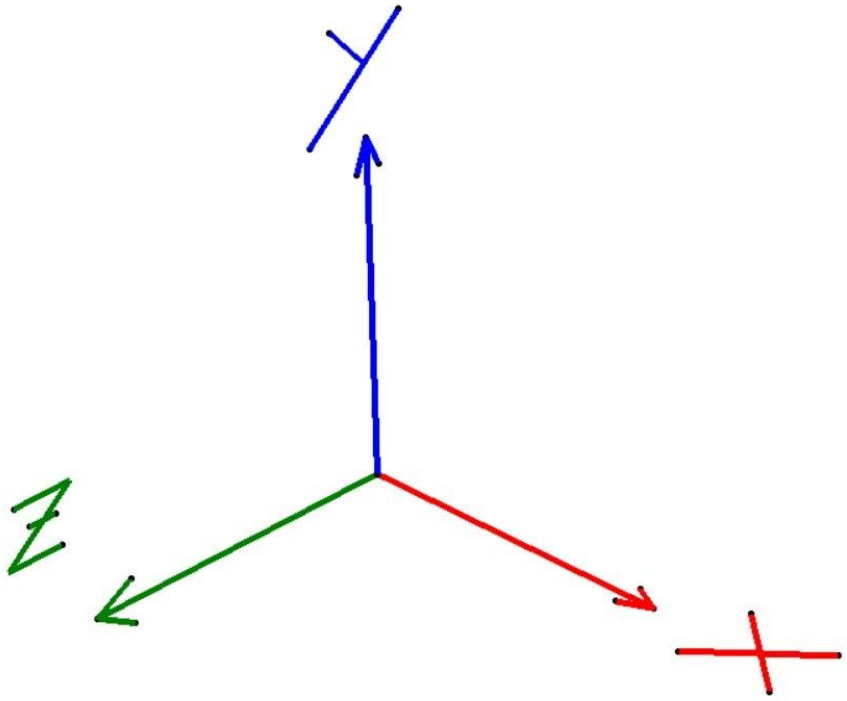
- GL_SRC_ALPHA: koristi alfa vrijednost teksture.
- GL_ONE_MINUS_SRC_ALPHA: koristi inverznu vrijednost alfa kanala za pozadinsku boju.

4.2. KOORDINATNI SUSTAV

U OpenGL-u koordinatni sustav igra ključnu ulogu u prikazu objekata u trodimenzionalnom prostoru. Kako bi se pravilno prikazali 3D objekti na dvodimenzionalnom zaslonu, koriste se razni matematički transformacijski procesi koji pretvaraju koordinate objekata iz njihovog lokalnog prostora u svjetski prostor, a zatim u prostor kamere i konačno na

zaslon [7]. OpenGL koristi kartezijanski trodimenzionalni koordinatni sustav s osima X, Y i Z, pri čemu:

- X-os predstavlja vodoravnu liniju (lijevo-desno),
- Y-os vertikalnu liniju (gore-dolje),
- Z-os dubinsku liniju (naprijed-nazad).



Slika 7: OpenGL koordinatni sustav

Transformacije i projekcije na koordinatni sustav:

1. Lokalni koordinatni sustav

Lokalni koordinatni sustav definira položaje vrhova objekta u njegovom vlastitom prostoru. Ovdje su koordinate objekta definirane relativno na njegov centar ili neku drugu referentnu točku. Ovaj sustav koordinata koristi se prije nego što se objekt postavi u scenu. Primjerice, kocka može imati vrhove definirane od -0.5 do 0.5, gdje su svi vrhovi relativni na središte kocke.

2. Svjetski koordinatni sustav

Svjetski koordinatni sustav odnosi se na cjelokupni prostor scene. U ovom sustavu svaki objekt ima svoje pozicije u globalnom prostoru. Da bi se objekt iz lokalnog koordinatnog sustava postavio u scenu, koristi se modelna matrica koja transformira koordinate objekta u svjetski prostor, omogućujući njegovo pozicioniranje, rotaciju i skaliranje unutar cijele scene.

3. Gledani koordinatni sustav

Gledani koordinatni sustav koristi se za prikazivanje objekata iz perspektive kamere. Kamera se u OpenGL-u definira kao virtualna "točka gledanja", a sve se transformacije objekata moraju izvršiti tako da odgovaraju perspektivi kamere. To se postiže s pomoću pogledne matrice (eng. view matrix), koja transformira sve objekte tako da izgledaju kao da ih gledamo s određene pozicije i orijentacije.

4. Projekcijski koordinatni sustav

Projekcijski koordinatni sustav koristi se za pretvaranje 3D koordinata u 2D prikaz na ekranu. Ovdje se koristi projekcijska matrica koja primjenjuje perspektivnu ili ortogonalnu projekciju. Perspektivna projekcija simulira kako ljudsko oko percipira svijet, gdje udaljeni objekti izgledaju manji, dok ortogonalna projekcija uklanja efekt perspektive i koristi se za tehničke prikaze gdje je važna točnost proporcija.

5. Normalizirani uređajni koordinatni sustav

Nakon projekcijske transformacije, svi objekti su unutar prostora nazvanog normalizirani uređajni prostor. Koordinate su sada između -1 i 1 po X, Y i Z osi, gdje su svi objekti izvan

ovog raspona odrezani (eng. clipping). Ovaj prostor predstavlja ekran i sve što nije unutar njega neće biti prikazano.

6. Prostor ekrana

Na kraju, nakon svih transformacija, koordinate se preslikavaju na prostor ekrana, gdje su vrijednosti X i Y skalirane na piksele na ekranu. Ovdje se objekti konačno prikazuju na 2D zaslonu [7].

U OpenGL-u, koordinate prolaze kroz niz transformacija pomoću različitih matrica. Tipičan niz transformacija je sljedeći:

1. Modelna matrica: transformira objekte iz lokalnog prostora u svjetski prostor.
2. Pogledna matrica: transformira objekte iz svjetskog prostora u gledani prostor.
3. Projekcijska matrica: transformira objekte iz gledanog prostora u projekcijski prostor.

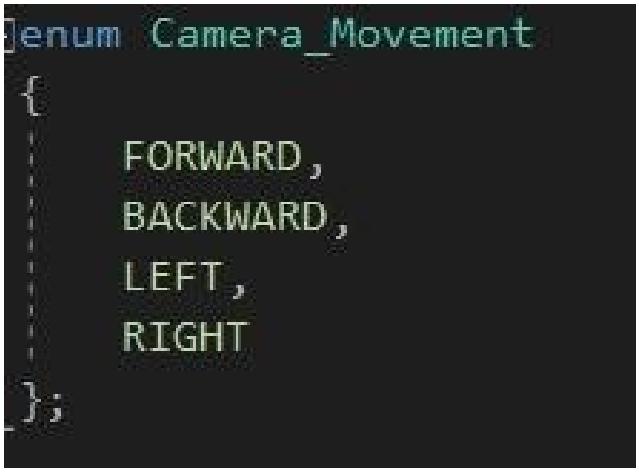
Kombiniranjem svih ovih matrica dobiva se potpuna transformacija objekta, koja se često naziva MVP matrica (Model-View-Projection):

```
glm::mat4 mvp = projection * view * model;
```

4.3. KAMERA

Kamera u 3D prostoru je ključna komponenta za manipulaciju i interakciju s objektima u virtualnom svijetu. Implementacijom kamere omogućavamo korisniku da se slobodno kreće kroz prostor, čime stječe osjećaj dubine i perspektive unutar scene [8]. Ova funkcionalnost je izuzetno korisna za pregledavanje, analizu i uređivanje složenih 3D modela.

Definiramo enumeraciju pod nazivom Camera_Movement koja predstavlja razne smjerove kretanja kamere. Ova enumeracija pruža jasan i organiziran način za upravljanje ulazima kamere:



Slika 8: Pomicanje kamere

- *FORWARD*: Pomiče kameru u smjeru u kojem trenutno gleda.
- *BACKWARD*: Pomiče kameru u suprotnom smjeru od trenutnog gledanja.
- *LEFT*: Pomiče kameru ulijevo od trenutnog gledanja.
- *RIGHT*: Pomiče kameru udesno od trenutnog gledanja.

Također definiramo skup zadanih vrijednosti za inicijalne postavke kamere, koje kontroliraju njezinu orijentaciju, brzinu, osjetljivost i razinu zumiranja:

- *YAW*: Horizontalna rotacija kamere. Vrijednost -90.0 stupnjeva osigurava da kamera počinje gledati prema negativnoj osi Z, što je uobičajeno za prednju stranu u mnogim aplikacijama.

- *PITCH*: Vertikalna rotacija kamere. Vrijednost 0.0 stupnjeva znači da je kamera poravnata s horizontom. Podesivši ovu vrijednost, omogućujemo kameri da gleda gore ili dolje.

- *SPEED*: Ova vrijednost definira koliko brzo kamera se kreće kada se izdaje naredba za kretanje. Viša vrijednost brzine omogućuje brže kretanje.

- *SENSITIVITY*: Ova vrijednost utječe na to koliko se kamera rotira na temelju pomaka miša. Viša osjetljivost znači da će mala pomicanja miša rezultirati većim rotacijama kamere.

- *ZOOM*: Ova vrijednost predstavlja kut gledanja kamere, koji utječe na to koliko široko ili usko vidimo scenu. Veća vrijednost zumiranja omogućava širi pogled na okolinu.

```
// Default camera values
const GLfloat YAW = -90.0f;
const GLfloat PITCH = 0.0f;
const GLfloat SPEED = 6.0f;
const GLfloat SENSITIVTY = 0.25f;
const GLfloat ZOOM = 45.0f;
```

Slika 9: Zadane vrijednosti kamere

Ove postavke omogućuju korisnicima da jednostavno i intuitivno upravljaju kamerom, pružajući bogato iskustvo prilikom interakcije s 3D objektima. Uz nadodamo i pomak miša da možemo ići točno tamo gdje hoćemo. Kako pokazivač ne bi smetao ili izlazio kada hoćemo rotirati kameru, moguće ga je isključiti. To stvara probleme koji onemogućuju pritisak na gumb sučelja i slično. Upravo zato moramo implementirati da je rotacija kamere moguća samo dok držimo lijevu tipku miša.

4.4. KLASA OBJEKTA

Kreiramo klasu pod nazivom Cube, koja predstavlja 3D objekt u sceni, u ovom slučaju kocku. Ova klasa omogućuje definiranje ključnih parametara objekta, poput pozicije, veličine (skale), osi rotacije, kuta rotacije i teksture koja će biti primijenjena na objekt. Koordinate za ovaj objekt, kao i ostale njegove atribute, bit će pohranjeni unutar klase, a proces iscrtavanja objekta se odvija pomoću OpenGL naredbi.

Prvo, teksture koje će biti primijenjene na kocku učitavamo iz odabranog direktorija. Koristimo funkciju LoadTextureFiles koja prolazi kroz sve datoteke u zadanom direktoriju i filtrira one koje imaju ekstenziju “.png” ili “.jpg”. Ove teksture kasnije možemo dodijeliti različitim objektima prilikom njihova kreiranja.

Klasa Cube sadrži nekoliko atributa koji definiraju stanje svakog pojedinog objekta: VAO (Vertex Array Object) i VBO (Vertex Buffer Object) za pohranu podataka o geometriji objekta, ID teksture, poziciju, skalu, os rotacije te kut rotacije. Na početku se ti atributi inicijaliziraju kroz konstruktor, koji prima poziciju, skalu, os rotacije, kut rotacije i ID teksture kao parametre.

U funkciji InitResources, OpenGL resursi za kocku se inicijaliziraju. To uključuje omogućavanje dubinskog testiranja (eng. depth testing) i miješanje transparentnosti, kao i generiranje VAO i VBO objekata. Podaci o kocki, uključujući njene vrhove i odgovarajuće teksturne koordinate, smještaju se u niz točaka. Ovi podaci zatim se učitavaju u VBO, a VAO se koristi za povezivanje vrhova i tekstura za iscrtavanje.

Funkcija Render koristi povezani *shader* program za iscrtavanje kocke. Tekstura se aktivira i veže na odgovarajući teksturno mjesto (u ovom slučaju GL_TEXTURE0), a zatim se kocka iscrtava s pomoću OpenGL naredbe glDrawArrays, koja iscrtava objekt kao set trokuta.

Kako bismo kocki dodali teksturu, koristimo funkciju LoadTexture. Ova funkcija prima putanju do datoteke teksture, generira ID teksture, postavlja parametre za teksturu (poput načina filtriranja i ponavljanja) te učitava sliku pomoću biblioteke SOIL2. Ako je učitavanje uspješno, tekstura se koristi za mapiranje na površinu kocke.

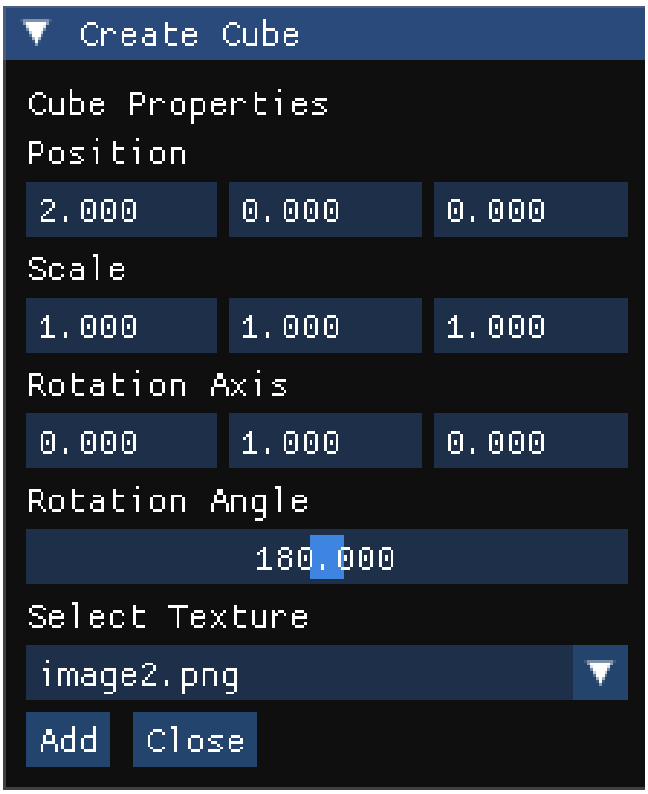
Kada želimo dodati novi objekt u scenu, pozivamo funkciju AddCube, koja prima poziciju, skalu, os rotacije, kut rotacije i naziv teksture. Ova funkcija učitava odgovarajuću teksturu, kreira novu instancu objekta Cube, inicijalizira njene resurse i dodaje objekt u vektor cubes, koji čuva sve kocke koje će biti iscrtane u sceni.

Također, osigurano je pravilno oslobađanje OpenGL resursa u funkciji Cleanup, koja briše VAO, VBO i teksture nakon što više nisu potrebni, kako bi se izbjeglo curenje memorije.

Ovakav pristup omogućuje jednostavno i fleksibilno dodavanje novih 3D objekata s različitim teksturama, skalama i rotacijama, te njihovo lako manipuliranje unutar scene, čineći proces kreiranja i iscrtavanja 3D objekata vrlo efikasnim.

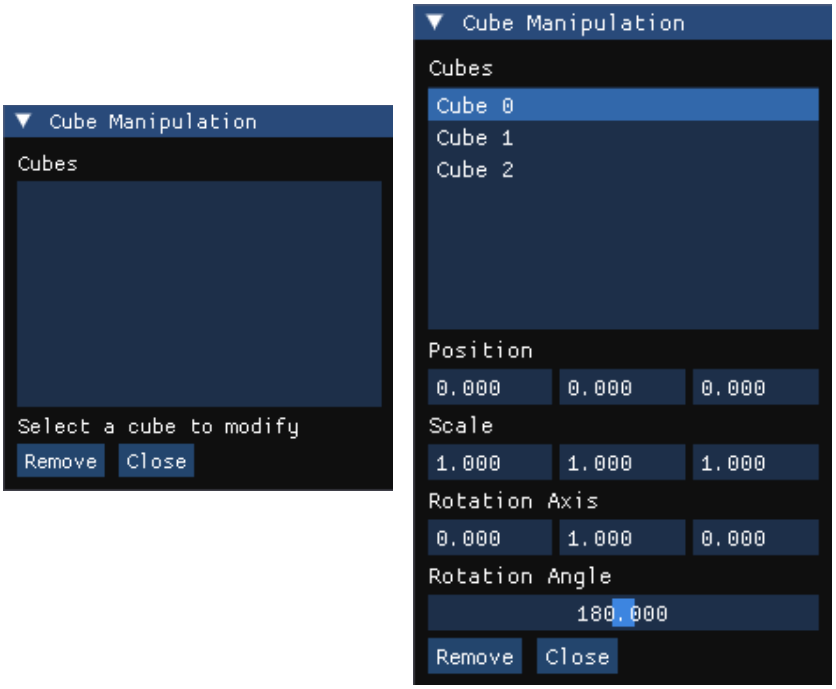
5. ANALIZA REZULTATA

U okviru našeg projekta implementirali smo napredni sustav za upravljanje 3D kockama koji korisnicima omogućuje jednostavno stvaranje, brisanje i manipulaciju objekata u virtualnom prostoru. Ovaj sustav značajno poboljšava interaktivnost korisničkog sučelja, omogućujući korisnicima da prilagode svoj 3D svijet prema vlastitim željama. Proces stvaranja kocke započinje otvaranjem prozora “Create Cube” unutar ImGui sučelja. Ovdje korisnici mogu unijeti različite atribute nove kocke, uključujući položaj, skalu, os rotacije i kut rotacije. Ova prilagodba omogućava korisnicima da definiraju točnu poziciju kocke u 3D prostoru, kao i njezine dimenzije, što doprinosi raznolikosti i dinamičnosti scene. Uz to, korisnici imaju mogućnost odabrati teksturu iz unaprijed definiranog popisa, čime se dodatno obogaćuje vizualna estetika objekta. Kada su svi parametri postavljeni, pritiskom na gumb "Add" korisnici mogu lako dodati novu kocku u svoju scenu.



Slika 10: Izrada 3D objekata

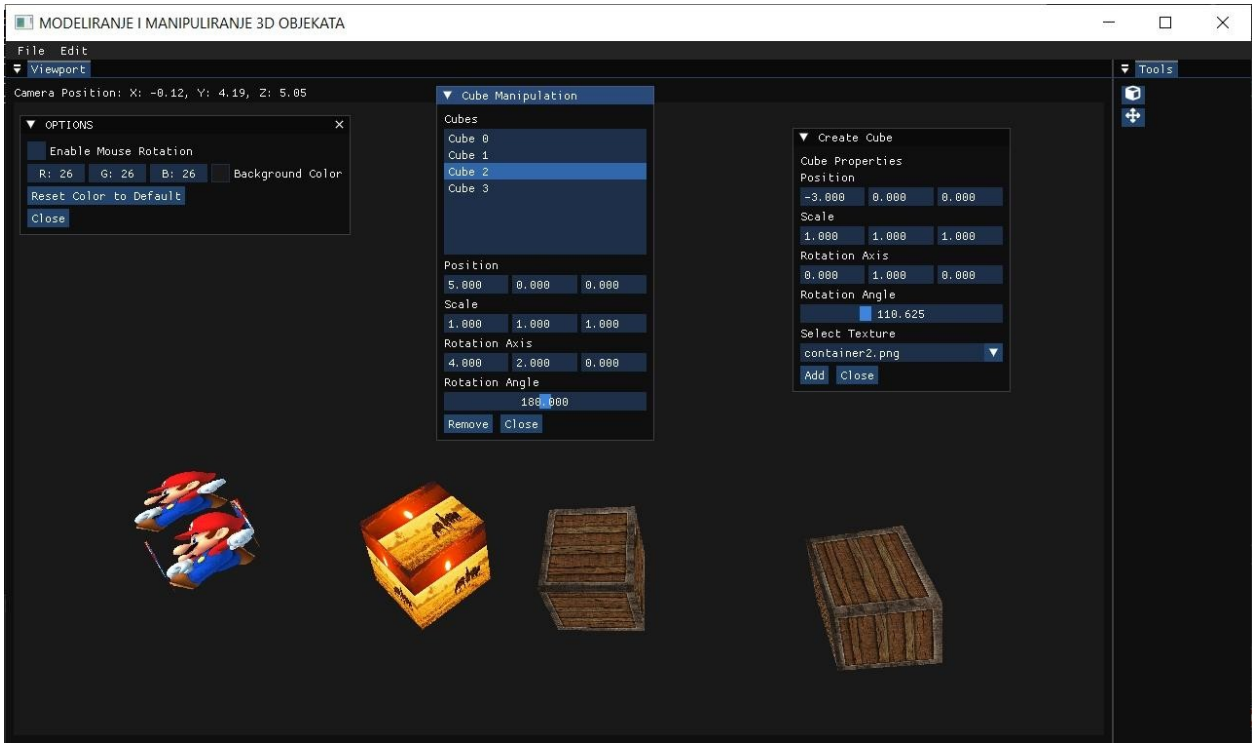
Osim stvaranja novih objekata, sustav omogućuje i manipulaciju postojećim kockama. Otvaranjem prozora “Cube Manipulation”, korisnici mogu pregledavati listu svih trenutnih kocki u sceni. Ova lista, označena indeksima kocki, olakšava prepoznavanje i odabir objekta koji korisnik želi modificirati. Kada je kocka odabrana, korisnici mogu mijenjati njezine atribute, poput položaja, skale, osi rotacije i kuta rotacije. Ova fleksibilnost omogućuje korisnicima da precizno prilagode svoje objekte prema vlastitim potrebama, čime se poboljšava njihovo iskustvo i kontrola unutar 3D okruženja. Konačno, sustav uključuje funkcionalnost za uklanjanje kocki, što korisnicima omogućuje da se riješe neželjenih objekata iz svoje scene. Nakon što odaberu kocku iz liste, jednostavno pritiskom na gumb "Remove" mogu ukloniti odabrani objekt. Ova opcija je od ključne važnosti za održavanje reda unutar radnog prostora, omogućujući korisnicima da organiziraju svoj okoliš i usmjere fokus na relevantne objekte. Ukupno gledajući, implementirani sustav za upravljanje 3D kockama s ImGui sučeljem nudi korisnicima intuitivno i interaktivno iskustvo modeliranja, pružajući im alate potrebne za stvaranje, manipulaciju i prilagodbu objekata unutar virtualnog prostora.



Slika 11: Upravljanje 3D objektima

6. ZAKLJUČAK

U ovom projektu razvijen je napredni sustav za prikazivanje i manipulaciju 3D objektima, koji korisnicima omogućuje stvaranje dinamičnih i interaktivnih scena. Implementacija OpenGL-a u kombinaciji s ImGui sučeljem omogućila je jednostavno korisničko iskustvo prilikom upravljanja 3D objektima, poput kocki, kroz intuitivne prozore za unos podataka. Korisnici sada imaju mogućnost prilagoditi atribute objekata kao što su položaj, skala, os rotacije i kut rotacije, čime se poboljšava njihova kreativnost i fleksibilnost u oblikovanju virtualnog prostora.



Slika 12: Prikazivanje više objekata

Osim mogućnosti stvaranja novih objekata, sustav omogućuje i lako upravljanje postojećim kockama, uključujući njihovu modifikaciju i uklanjanje. Ova funkcionalnost ne samo da poboljšava iskustvo korištenja, već i olakšava održavanje reda unutar scene, omogućujući korisnicima da se fokusiraju na relevantne elemente.

Projekt također istražuje važnost uporabljivosti i intuitivnosti u razvoju grafičkih sučelja, što je ključno za angažiranje korisnika i poticanje njihove interakcije s aplikacijom. Budući radovi mogli bi se usmjeriti na daljnje poboljšanje performansi prikazivanja, dodavanje složenijih 3D modela, kao i implementaciju dodatnih funkcija poput animacija i fizičkih simulacija. Tako, naš sustav može postati još robusniji alat za kreativne projekte i profesionalne primjene u području 3D modeliranja i vizualizacije.

LITERATURA

1. Khronos Group: The OpenGL Graphics System: A Specification, 4.0 (Core Profile), ožujak 11, 2010 (PDF).

2. Khronos Group: GLSL Language Specification, Version 3.30.6 (PDF)

3. GLM Team: GLM Documentation, dostupno na: https://github.com/g-truc/glm/blob/master/manual.md, pristupljeno 15.10.2024.

4. Khronos Group: A list of GLUT alternatives, dostupno na: https://www.opengl.org/resources/libraries/windowtoolkits/, pristupljeno 15.10.2024.

5. Službena dokumentacija ImGui-a. Dostupno na: https://github.com/ocornut/imgui

6. SpartanJ: SOIL2, dostupno na: https://github.com/SpartanJ/SOIL2, pristupljeno 15.10.2024.

7. Eh Chua: 3D Graphics with OpenGL - The Basic Theory, dostupno na: https://personal.ntu.edu.sg/ehchua/programming/opengl/CG_BasicsTheory.html, pristupljeno 15.10.2024.

8. LearnOpenGL: Getting started with Camera, dostupno na: https://learnopengl.com/Getting-started/Camera, pristupljeno 15.10.2024.