Virtualni kokpit - prikaz brzinometra

Knežević, Dino

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

Sveučilišni diplomski studij Informacijske Tehnologije i Podatkovne Znanosti

Virtualni kokpit – prikaz brzinometra

Diplomski rad

Dino Knežević

Osijek, 2024.

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

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Dino Knežević
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1293R, 07.10.2022.
JMBAG:	0165081812
Mentor:	doc. dr. sc. Tomislav Galba
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Tomislav Keser
Član Povjerenstva 1:	doc. dr. sc. Tomislav Galba
Član Povjerenstva 2:	izv. prof. dr. sc. Alfonzo Baumgartner
Naslov diplomskog rada:	Virtualni kokpit - prikaz brzinometra
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Napisati program koristeći OpenGL ili drugu biblioteku za generiranje grafičkog sadržaja koji će simulirati i prikazati analogni brzinometar. Također, potrebno je implementirati i mogućnost promjene stanja kazaljke ovisno o inputu (automatska promjena - demo prikaz ili putem nekog input-a kao npr. tipkovnica).

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	17.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	30.09.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	30.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 30.09.2024.

Ime i prezime Pristupnika:	Dino Knežević
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1293R, 07.10.2022.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Virtualni kokpit - prikaz brzinometra**

izrađen pod vodstvom mentora doc. dr. sc. Tomislav Galba

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 pristupnika:

SADRŽAJ

1. Uvod

U svakom dijelu modernog života, računala, razne tehnologije i uređaji su sve potrebniji i sve češća svakodnevna pojava. Jedan od ključnih dijelova je i računalna grafika te joj svakodnevno raste potreba za napretkom u njenim performansama, mogućnostima i učinkovitosti. Glavni cilj računalne grafike je iz godine u godinu bio fotorealistični prikaz, ali sada je veliki fokus na pronalazak novih i efikasnih načina, odnosno pristupa računalnoj grafici. Razne djelatnosti su ovisne o računalnoj grafici poput arhitekture, simulacije, industrija video igara i brojne druge uključujući automobilsku industriju. Kako bi grafička sučelja bila moguća i efikasna, koriste se programska sučelja, poput OpenGL-a, koji je jedno od najpoznatijih aplikacijskih sučelja za grafiku.

Za prikaz grafičkih elemenata, odnosno računalne grafike u realnom vremenu potrebno je izvršiti niz matematičkih operacija za određivanje boje i položaja svake točke na ekranu. Ove operacije nisu dovoljno brze na CPU-u, pa je potrebno koristiti grafičke kartice. OpenGL omogućava pristup grafičkoj kartici i omogućujuće izvođenje raznih ili vlastitih programa na grafičkim karticama, zahvaljujući paralelnom načinu obrade podataka.

U ovom radu opisan je proces korištenja OpenGL-a za simulaciju i prikaz analognog brzinomjera. U prvoj polovici rada se detaljno opisuje način kreiranja računalne grafike. U drugoj polovici opisuje se implementacija interaktivnog sučelja koje omogućuje kontrolu kazaljke brzinomjera putem automatskog prikaza ili korisničkog unosa.

1.1. Zadatak diplomskog rada

Cilj je razviti aplikaciju koristeći OpenGL ili sličnu biblioteku za grafiku koja će simulirati i prikazati analogni brzinomjer. Potrebno je implementirati mogućnost promjene stanja kazaljke ovisno o inputu, bilo kroz automatsku promjenu kao dio demo prikaza ili putem korisničkog unosa putem tipkovnice.

2. Pregled područja

2.1. Povijest brzinometra

Prvi električni brzinometar izumio je Hrvat Josip Belušić 1888. godine [1], nazvan velociometar. Ovaj uređaj je označio važan korak u preciznom mjerenju brzine. Također, Charles Babbage je razvio rani tip brzinomjera za lokomotive. Kasnije, 1902. godine, Otto Schulze je patentirao brzinomjer s rotirajućim fleksibilnim kabelom, koji se obično pogonio preko zupčanika povezanih s prijenosom vozila. Ovakav tip brzinometra je također prepoznatljiv kao analogni, odnosno mehanički brzinometar. Primjer analognog prikaza je na slici 2.1.

Sl. 2.1. Analogni kokpit u automobilu Audi RS6 C5

Kako je automobil industrija rasla, potreba za preciznijim mjerenjima dovela je do evolucije brzinomjera. Mehanički sustavi postali su složeniji, no s vremenom su ih zamijenili elektronski sustavi koji su nudili bolju točnost i pouzdanost. Brzinomjer je uređaj koji služi za mjerenje i prikaz trenutne brzine kretanja vozila, u automobilima se iskazuje u kilometrima po satu ili u miljama po satu, ovisno koristi li se metrički ili imperijalistički sustav mjernih jedinica.

2.2. Povijest digitalnih brzinomjera

Digitalni brzinomjeri su se razvili kao odgovor na potrebu za preciznijim i pouzdanijim mjernim uređajima. Prvi modeli su se pojavili u kasnim desetljećima 20. stoljeća [2], zamjenjujući mehaničke sustave koji su bili podložni habanju i nepravilnostima u mjerenju. S napretkom

tehnologije, digitalni brzinomjeri su postali standard u automobilskoj industriji, pružajući ne samo točnija mjerenja brzine, već i dodatne funkcionalnosti kao što su integracija s navigacijskim sustavima i prikaz podataka o potrošnji goriva.

Razvoj digitalnih brzinomjera također je bio potaknut napretkom u mikroelektronici i senzorima, što je omogućilo smanjenje veličine i cijene komponenti, te povećanje njihove pouzdanosti. Ovi uređaji su postali vrlo važni ne samo u automobilima, već i u raznim industrijskim primjenama, uključujući zrakoplovstvo i željeznički transport, gdje su točnost i pouzdanost mjerenja ključni za sigurnost i učinkovitost.

2.3. Prednosti i nedostatci digitalnih brzinomjera

Digitalni brzinomjeri donose brojne prednosti u usporedbi s tradicionalnim mehaničkim sustavima. Njihova preciznost i pouzdanost osiguravaju točnija očitanja brzine. Iako su brzinomjeri vrlo precizni, proizvođači automobila namjerno uvode toleranciju [3] kako bi vozač imao dojam da vozi brže, čime se povećava sigurnost na cesti. Dodatne funkcionalnosti poput integracije s navigacijskim sustavima pružaju korisne informacije u stvarnom vremenu.

Digitalni sustavi mogu biti složeniji za održavanje i popravak te su često skuplji u proizvodnji. Također, oslanjanje na elektroničke komponente može dovesti do problema s pouzdanošću u ekstremnim uvjetima, gdje su mehanički sustavi robusniji.

2.4. Tehnološki trendovi

U razvoju digitalnih brzinomjera uočavaju se ključni tehnološki trendovi. Jedan od najznačajnijih je integracija s pametnim sustavima vozila, koja omogućuje povezivanje s aplikacijama i internetom, pružajući vozačima dodatne informacije poput uvjeta na cesti i navigacijskih uputa.

Daljnji razvoj uključuje upotrebu heads-up display (HUD) tehnologije, koja projicira podatke izravno na vjetrobransko staklo, čime se smanjuje potreba za skretanjem pogleda s ceste. Tu je i povećana upotreba senzora za preciznije praćenje brzine i analizu vožnje, što doprinosi sigurnosti i učinkovitosti.

Ovi trendovi pokazuju kako digitalni brzinomjeri postaju sve sofisticiraniji i poboljšavaju cjelovito iskustvo vožnje. Ovo uvodi dodatnu kompleksnost, povećava troškove i vrijeme razvoja jer uz prisutne tehnologije u automobilu jer je potrebno osigurati što bolje performanse i učinkovitost što ukazuje na potrebu za daljnjim istraživanjem i razvojem.

2.5. Primjene digitalnih brzinomjera

Digitalni brzinomjeri nalaze široku primjenu u raznim industrijama zbog svoje preciznosti i pouzdanosti. Digitalni brzinomjeri u zrakoplovstvu igraju ključnu ulogu u praćenju brzine letjelice gdje su preciznost i pouzdanost od iznimnog značaja za sigurnost. U željezničkom prometu [4] brzinomjeri osiguravaju da vlakovi operiraju unutar sigurnih brzinskih granica što je ključno za sprječavanje nesreća i osiguranje pravovremenog dolaska.

U industrijskim postrojenjima digitalni brzinomjeri se koriste za praćenje brzine rotacijskih strojeva i turbina. To omogućuje precizno upravljanje i održavanje, smanjujući rizik od kvarova i poboljšava operativnu učinkovitost. Primjena je također moguća i izvan vozila u svrhu sigurnosti prometa na način da se brzinometar u kombinaciji s radarom postavi uz stranu ceste [5].

Brzinometri u vodenom prometu [6] pomažu u navigaciji i praćenju brzine plovila. Različiti uvjeti poput struja i valova zahtijevaju prilagodbu brzine za optimalnu plovidbu. Digitalni brzinometri pružaju točne podatke potrebne za donošenje informiranih odluka čime poboljšavaju sigurnost i učinkovitost operacija. Ovisno o primjeni i području izvedbe se razlikuju ali uloga brzinometra je jednako značajna u svakom području. U svim ovim industrijama digitalni brzinometri su postali standard zbog svojih sposobnosti integracije s modernim sustavima i prilagodljivosti raznim uvjetima. Primjer digitalnog brzinomjera u automobilu prikazan je na slici 2.2.

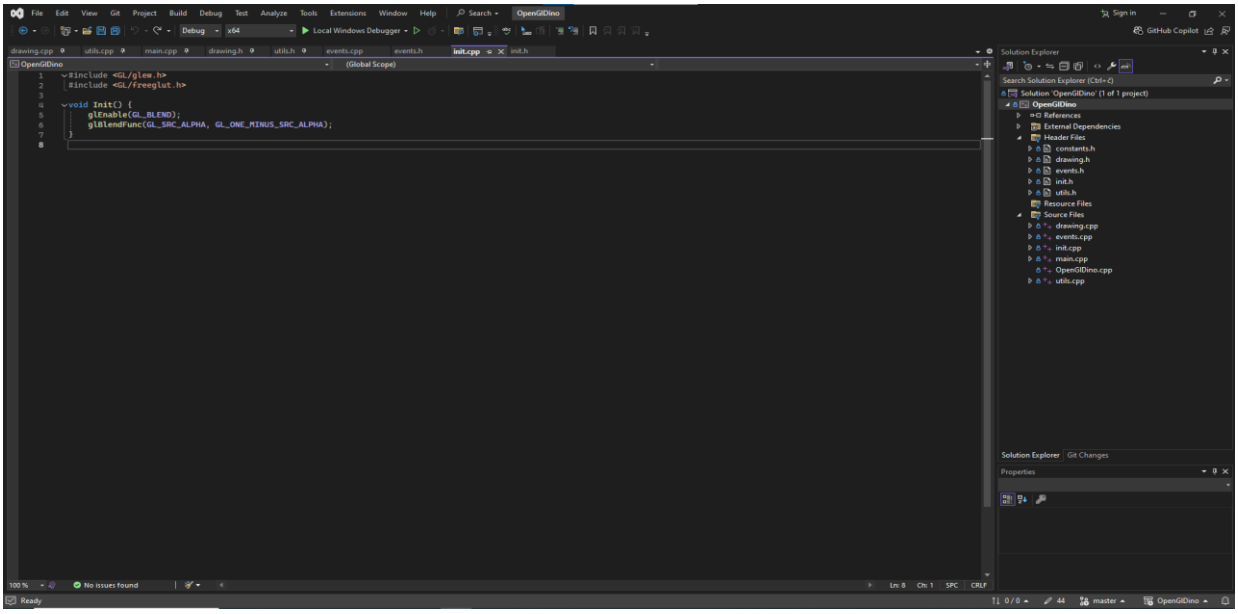
Sl. 2.2. Digitalni kokpit u automobilu *Audi R8*

3. Korištene tehnologije

Za realizaciju ovog rada bilo je potrebno koristiti nekoliko tehnologija čija će uloga biti detaljnije opisana u nastavku.

3.1. Microsoft Visual Studio 2022

Microsoft Visual Studio [7] je jedno od najpopularnijih integriranih razvojnih okruženja (IDE) koje se koristi za razvoj softvera. Razvijen od strane Microsofta Visual Studio podržava više programskih jezika uključujući C++, C#, Python i ostale čime omogućuje fleksibilnost i prilagodljivost za različite projekte. Ovaj IDE je odabran za implementaciju rada zbog velikog broja mogućnosti i materijala. Na slici 3.1 je prikazano njegovo sučelje.



Sl. 3.1. Sučelje Microsoft Visual Studio 2022

Nudi širok spektar funkcionalnosti koje olakšavaju razvoj složenih projekata. Jedna od ključnih značajki je njegov napredni alat za otklanjanje pogrešaka koji omogućuje programerima identificiranje i ispravljanje pogrešaka na jednostavan i učinkovit način. Alat za otklanjanje pogrešaka uključuje mogućnost postavljanja točaka prekida (engl. *breakpoints*), promatranje varijabli u realnom vremenu te analizu izvršavanja koda liniju po liniju. Visual Studio pruža alate za profiliranje koda koji pomažu programerima u optimizaciji performansi aplikacije. Ovi alati omogućuju analizu potrošnje resursa, identificiranje uskih grla i poboljšanje brzine izvršavanja aplikacija. Alati za refaktoriranje koda također su dostupni omogućujući programerima prestrukturiranje postojećeg koda bez promjene njegove vanjske funkcionalnosti čime se poboljšava čitljivost i održavanje programskog koda. Integracija sa sustavima za kontrolu verzija

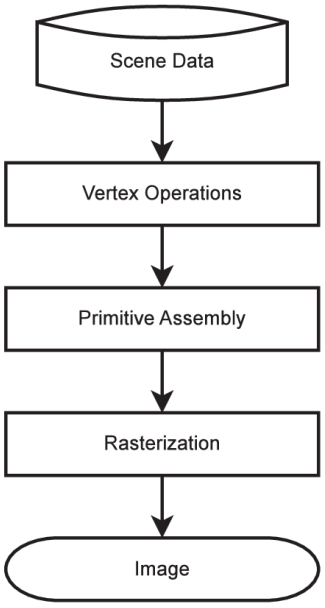
poput GitHub-a [8] je još jedna važna značajka Visual Studija. Ova integracija omogućuje programerima jednostavno upravljanje promjenama u kodu, kolaboraciju s timovima i praćenje povijesti promjena. Značajno olakšava razvoj jer pruža mogućnost bezbrižne promjene i eksperimentiranje pri radu te ako dođe do pogreške u implementaciji jednostavno je vratiti se na originalno ispravno stanje. Visual Studio ima bogat ekosustav dodataka (engl. *extensions*) koji proširuju njegovu funkcionalnost omogućujući prilagodbu razvojnog okruženja specifičnim potrebama projekata.

3.2. Programski jezik C++

Računalna grafika zahtijeva dobre performanse te je zato potrebno odabrati ispravne alate za njen razvoj poput odabira programskog jezika koji može biti ključan kada se u obzir uzme veličina i zahtjev projekta. C++ [9] je moćan višenamjenski programski jezik koji je poznat po svojoj učinkovitosti i fleksibilnosti. Razvijen kao proširenje C jezika, C++ dodaje objektno orijentirane značajke omogućujući programerima stvaranje složenih i modularnih aplikacija. Korištenje C++ jezika raširenо je u raznim industrijama uključujući razvoj video igara, aplikacija, sustava u stvarnom vremenu i u automobilskoj industriji. Nudi visoku razinu kontrole nad očvrsjem što ga čini odličnim izborom za razvoj aplikacija koje zahtijevaju maksimalno optimizirane performanse. Razvojni programeri imaju mogućnost upravljanja memorijom što omogućuje preciznu kontrolu nad resursima. Iako to povećava složenost, vrijeme i troškove razvoja pruža mogućnost stvaranja vrlo učinkovitih aplikacija te dolazi do posebnog izražaja u okruženjima gdje su resursi ograničeni. Dolazi s bogatom standardnom bibliotekom koja pruža niz funkcionalnosti uključujući rad s podacima, algoritme i manipulaciju stringovima. Standardna biblioteka omogućuje programerima korištenje već definiranih funkcija i klasa čime se smanjuje vrijeme razvoja i povećava pouzdanost aplikacija. Jedna od najvažnijih karakteristika C++ jezika je objektno orijentirano programiranje (OOP). OOP omogućuje razvojnim programerima organizaciju koda u objekte, što poboljšava modularnost, ponovnu upotrebu koda i olakšava održavanje. Osnovni principi OOP-a kao što su enkapsulacija, nasljeđivanje i polimorfizam omogućuju programerima stvaranje složenih sustava s jasnom strukturom i hijerarhijom. C++ je poznat po svojoj prenosivosti što znači da se programi mogu pokrenuti na različitim platformama s minimalnim promjenama. Ova značajka čini C++ idealnim za razvoj aplikacija koje trebaju raditi na različitim operativnim sustavima. C++ omogućuje interoperabilnost s drugim jezicima poput C-a što nudi mogućnost korištenja postojećeg koda i biblioteka unutar C++ projekata.

3.3. OpenGL

OpenGL [10] (*Open Graphics Library*) je snažan platformno neovisan API koji se koristi za renderiranje 2D i 3D vektorske grafike. Razvijen je za pružanje standardiziranog sučelja koje omogućava programerima stvaranje grafičkog sadržaja visoke kvalitete. OpenGL je korišten u raznim industrijama uključujući video igre, simulacije, CAD aplikacije i druge grafičke aplikacije,zbog svoje fleksibilnosti i brojnih i korisnih funkcionalnosti. Nije ovisan o programskom jeziku u kojem se vrši implementacija. Služi za komunikaciju s grafičkim očvrsjem (engl. *hardware*), a dio očvrsja zaslužan za iscrtavanje je grafička kartica s čime se postiže ubrzano iscrtavanje uz pomoć očvrsja (engl. *hardware-accelerated rendering*). Implementacija ovog rada je na razini softvera(engl. *software*) odnosno programske podrške. Način rada, tj. slijed iscrtavanja se može prikazati u nekoliko koraka koji su prikazani na slici 3.2 te zatim detaljno opisani.



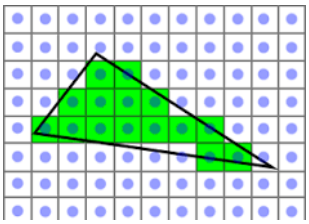
Sl. 3.2. Pojednostavljeni tijek iscrtavanja

Tijek iscrtavanja:

1. Unos podataka: Podaci o geometriji poput koordinata vrhova, normala i teksturnih koordinata, unose se u program. Zatim se pohranjuju u *vertex buffer* objekte (VBO) i povezuju s *vertex array* objektima (VAO).
2. Obrada vrhova (engl. *Vertex shader*): Ovdje se vrši transformacija vrhova, primjena projekcijskih matrica, te izračun svjetlosnih efekata i drugih atributa. Zbog izvođenja na grafičkoj kartici izvodi se paralelno i za n *vertexa, vertex shader* će se pozvati n puta. Za prikaz na ekran potrebno je svaki vertex transformirati i prebaciti u poseban OpenGL koordinatni

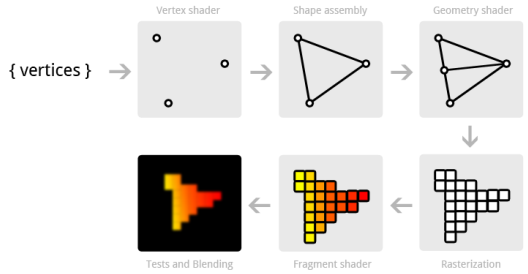
sustav. Definirani raspon [-1,1] u svakoj od 3 koordinatne osi. Dodatan potkorak je naknadna obrada vrhova (engl. *Vertex Post-Processing*). Transformira se povratna informacija i sprema u međuspremnik te time jednostavni oblici i podatci vrhova ne prolaze ranije korake i dobiva se poboljšanje u performansama i štednja resursa. Nakon toga pokreće se isijecanje (engl. *Clipping*) gdje se izbacuju oblici koji su jednostavni i nisu vidljivi jer se time štedi na resursima tako što se neće iscrtavati ono što nije vidljivo.

3. Sabiranje jednostavnih oblika (engl. *Primitive Assembly*): Prvi potkorak je taj da se sve definira(matematički) kao jednostavan oblik. Za primjer oblika iscrtavanja se može istaknuti trokut gdje će niz od šest vrhova postati niz od dva trokuta. Slijedeći korak je odstrjel lica (engl. *Face culling*), nakon grupiranja svih vrhova u jednostavne oblike posljedično tome je potrebo odrediti površinu geometrijskog lika koja je vidljiva i označiti površinu koja nije vidljiva kako se ne bi iscrtavala. Primjenom navedenog koraka, odnosno njegovih potkoraka se štedi na resursima i ubrzava iscrtavanje.
4. Rasterizacija (engl. Rasterization): Proces s kojim se koristeći interpolaciju određuje koja vrijednost se pridružuje kojem pikselu. Sklopljene plohe se razdvajaju na fragment kako bi im se odredila boja tj. tekstura. Svaki fragment nosi informacije o položaju, boji i teksturnim koordinatama. Kao ulazni parametar prima jednostavan oblik proslijeđen iz prošlog koraka i pretvara ih u fragmente ili dijelove. Izlazni parametar su fragmenti i za svaki jednostavan oblik se stvara minimalno jedan fragment ali moguće je stvoriti i više fragmenata. Shader fragmenata (engl. *Fragment Shader*) obrađuje fragmente i nije obavezan korak ali se jako često koristi jer je neophodan za osvjetljenje i sjene. Ovdje se određuje konačna boja piksela, primjenjuju teksture i sjene, te izvode složeni efekti poput *bump mappinga* ili *anti-aliasinga* [11]. Prikaz dobivanja fragmenta iz oblika je na slici 3.3.



Sl. 3.3. Dobivanje fragmenta iz oblika

5. Konačni prikaz: Nakon svih koraka konačni rezultat se prikazuje na zaslonu. Prije konačnog prikaza izvode se testovi nad fragmentima koji odlučuju hoće li se pojedini fragment prikazati na zaslonu. Test dubine (engl. *depth test*) je ključna operacija koja osigurava da se samo vidljivi fragmenti iscrtavaju na zaslonu. Tijekom ovog testa dubina svakog fragmenta uspoređuje se s već pohranjenim vrijednostima u z-bufferu. Fragmenti koji se nalaze iza drugih fragmenata su odbačeni čime se sprječava nepotrebno iscrtavanje odnosno korištenje resursa i poboljšavaju performanse. Stapljanje (engl. *blending*) određuje konačnu boju piksela kod fragmenata s prozirnošću. Kada su fragmenti prozirni njihova boja se miješa s bojom fragmenata ispod njih prema određenim pravilima. Ova operacija omogućava stvaranje efekata poput stakla, magle ili dima, gdje se vidi više slojeva kroz prozirne dijelove. Primjer nastanka konačnog prikaza je na slici 3.4.



Sl. 3.4. Nastanak konačnog prikaza

3.3.1. Glew

OpenGL *Extension Wrangler Library* (GLEW) [12] je biblioteka otvorenog koda za C/C++ programski jezik dizajnirana da pojednostavi korištenje OpenGL ekstenzija. Prilikom razvoja računalne grafike potrebno je koristiti napredne funkcionalnosti koje vrlo često nisu dostupne u osnovnoj verziji OpenGL-a te je GLEW često dobar izbor. GLEW automatizira proces prepoznavanja i pristupa ovim ekstenzijama omogućujući razvojnim programerima korištenje najnovijih grafičkih mogućnosti odnosno funkcionalnosti. GLEW upravlja pokazivačima na funkcije i inicijalizira ih prilikom pokretanja programa i uvelike olakšava razvojni proces i skraćuje vrijeme implementacije ali također smanjuje i šansu za progreškom. Nakon inicijalizacije funkcija sprema ih u datoteku zaglavlja kako bi se mogle dalje koristiti. GLEW provjerava koje su ekstenzije dostupne na trenutnom grafičkom hardveru i operativnom sustavu. Ovaj proces osigurava kompatibilnost i omogućuje upotrebu novih značajki čim postanu dostupne uključujući funkcionalnosti poput *framebuffer* objekata, shaderskih programa i različitih vrsta teksturiranja. Time se omogućuje razvoj složenih grafičkih aplikacija s visokom razinom vizualne kvalitete. Ključna prednost GLEW-a je ta što je dizajniran da radi na više platformi uključujući Windows,

Linux i macOS. Ovo omogućava razvojnim programerima da razvijaju aplikacije koje su prenosive i mogu raditi na različitim sustavima bez potrebe za velikim izmjenama koda.

3.3.2. Glut(FreeGLUT)

GLUT(OpenGL Utility Toolkit) je biblioteka koja pojednostavljuje rad s OpenGL-om pružajući jednostavno sučelje za upravljanje prozorima unosom s tipkovnice i miša te drugim osnovnim funkcionalnostima. Razvijena kao platformno neovisna biblioteka GLUT omogućuje jednostavnu izradu OpenGL aplikacija bez potrebe za upravljanjem specifičnim detaljima operativnog sustava. Nažalost zbog lošeg stanja održavanja biblioteke(neodržavano već 20 godina) nastala je potreba za alternativom koja će pružati iste ili proširene i dodatne funkcionalnosti uz redovito održavanje i napredak. FreeGLUT je besplatna biblioteka otvorenog koda i moderna alternativa originalnom GLUT-u. Razvijen kako bi nadoknadio ograničenja izvornog GLUT-a freeGLUT nudi proširene funkcionalnosti i poboljšane mogućnosti što ga je učinilo prilično popularnim izborom među razvojnim programerima. Ima veći broj značajki poput:

• Upravljanje prozorima: Omogućuje jednostavno stvaranje i upravljanje prozorima za prikaz OpenGL sadržaja. Podržava različite stilove prozora, poput prozora s okvirom, bez okvira ili preko cijelog zaslona. Interakcija s korisničkim sučeljem je intuitivna, omogućavajući jednostavno upravljanje i prilagodbu prozora.

• Unos i događaji: Pruža jednostavne načine za rukovanje unosom s tipkovnice, miša i drugih uređaja. Ova funkcionalnost omogućuje stvaranje interaktivnih aplikacija bez dodatne složenosti. Na primjer, razvojni programeri mogu definirati funkcije koje će se pozivati prilikom pritiska tipki na tipkovnici, kretanja miša ili drugih korisničkih akcija, čime se olakšava razvoj aplikacija s bogatim korisničkim iskustvom.

• Podrška za više platformi: FreeGLUT je kompatibilan s različitim operativnim sustavima, uključujući Windows, macOS i Linux čime aplikacije budu prijenosive i konzistentne, što je ključno za razvoj aplikacija koje trebaju raditi na različitim platformama bez značajnih izmjena u kodu.

• Poboljšanja i ekstenzije: U usporedbi s originalnim GLUT-om, freeGLUT nudi podršku za višestruke monitore, omogućujući aplikacijama prikaz na više ekrana. Osim toga, pruža kontrolu nad kontekstom prikaza, što omogućuje precizniju kontrolu nad procesom renderiranja i koristi dostupne moderne grafičke funkcionalnosti. Korištenje freeGLUT-a je jednostavno i često

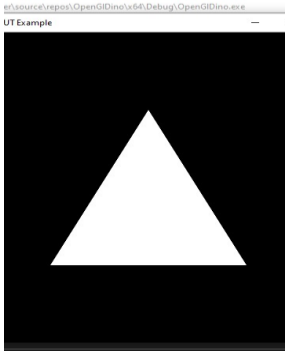
se koristi za brzi razvoj prototipa. Na primjer, osnovni program za kreiranje prozora i prikazivanje jednostavnog geometrijskog oblika može se napisati u nekoliko linija koda, koristeći funkcije kao što su `glutInit`, `glutCreateWindow`, `glutDisplayFunc` i `glutMainLoop`. Primjer koda s kojim se renderira trokut u središtu prozora je na slici 3.5. dok je prikaz na slici 3.6.

```
#include <GL/freeglut.h>

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glVertex2f(-0.5f, -0.5f);
    glVertex2f(0.5f, -0.5f);
    glVertex2f(0.0f, 0.5f);
    glEnd();
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("FreeGLUT Example");
    glutDisplayFunc(display);
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glutMainLoop();
    return 0;
}
```

Sl. 3.5. Primjer programskog koda za trokut u središtu prozora



Sl. 3.6. Prikaz trokuta u središtu prozora

Funkcija „*glutInit*“ inicijalizia GLUT i obrađuje sve argumente naredbenog retka. „*GlutInitDisplayMode*“ postavlja način prikaza uključujući jedno međuspremanje i RGB boje. Funkcije „*glutInitWindowSize*“ i „*glutCreateWindow*“ su zaslužne za postavljanje veličine prozora i njegovo stvaranje. Zatim se poziva funkcija „*glutDisplayFunc*“ koja registrira funkciju za iscrtavanje sadržaja na prozoru. Funkcija „*glutMainLoop*“ ulazi u glavnu petlju programa čekajući događaje i pozivanju registrirane funkcije.

3.4. Cmake

CMake[13] je popularni besplatni program otvorenog koda za upravljanje procesom izgradnje softverskih projekata te je neovisan o prevoditelju. Pristup izvornom kodu omogućuje proširenja prema specifičnim potrebama projekta. Razvijen kako bi pojednostavio kompilaciju i konfiguraciju CMake omogućuje razvojnim programerima definiranje procesa izgradnje softvera u prenosivom i sustavno neovisnom formatu. Može generirati izvorno okruženje za izgradnju koje kreira bibilioteke i prevodi programski kod te također može generirati programske omotače oko nekog programskog koda (eng. *wrappers*). Izvršne datoteke može graditi u proizvoljnim kombinacijama koje ovise o odabiru razvojnog programera koji razvija programsko rješenje. Podržava statičku i dinamičku izgradnju biblioteka. CMake podržava različite strukture direktorija koje mogu biti složene koje ovise o jednoj ili više biblioteka. Omogućuje razvojnim programerima definiranje projekta pomoću CMakeLists.txt datoteka, gdje se specificiraju izvori, zaglavlja, biblioteke i druge konfiguracijske opcije. Primjer osnovne konfiguracije prikazan je na slici 3.7.

```
1    cmake_minimum_required(VERSION 3.10)
2    project(MyProject)
3
4    set(CMAKE_CXX_STANDARD 11)
5
6    add_executable(MyExecutable main.cpp)
7    S
```

Sl. 3.7. Prikaz osnovne CMake konfiguracije

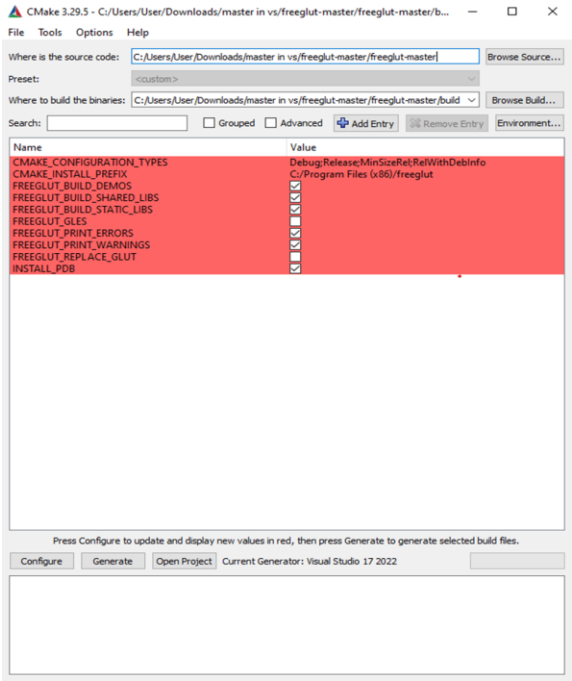
Ovaj jednostavni primjer pokazuje osnovnu konfiguraciju CMake projekta gdje se definira minimalna verzija CMake-a, postavlja naziv projekta i definira izvršni program s glavnim izvorom „*main.cpp*“. CMake automatski pronalazi i upravlja ovisnostima projekta olakšavajući integraciju vanjskih biblioteka i modula. Podržava in source i out of source izgradnju. In source izgradnja je opcija izgradnje gdje su sve datoteke potrebne za projekt poput C++ zaglavlja i ostalih datoteka unutar jednog direktorija. Konfiguracija *CmakeLists.txt* se također mora nalaziti unutar istog direktorija i ovaj pristup je dobar za jednostavne projekte i upravo zato se koristi za potrebe ovog

diplomskog rada za generiranje datoteka određenih biblioteka poput „*freeglut.dll*“ koji služi kao dinamička biblioteka. Out of source način izgradnje projekta je takav da su direktoriji poput build direktorija i ostalih odvojeni jedni od drugih te svaki može sadržavati jedan ili više direktorija unutar sebe kao i datoteka. Ovaj način je kompleksniji jer svaki direktorij mora sadržavati „*CmakeLists.txt*“ konfiguraciju. Omogućuje različite načine izgradnje koj su neovisni o platformi te mogu biti razvojni, testni ili produkcijski. CMake se može koristiti pomoću komandne linije ili korištenjem grafičkog sučelja. Primjer korištenja putem komandne linije nalazi se na slici 3.8.

```
PS C:\d disk\Diplomski WebGL> mkdir build
>> cd build
>> cmake -G "Make sme files" ../source
```

Sl. 3.8. Korištenje CMake funkcionalnosti putem komandnog prozora

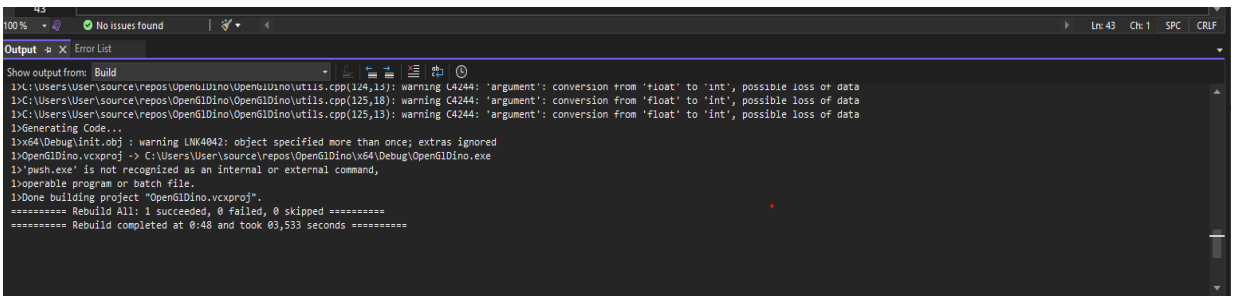
Naredbom „*mkdir build*“ kreira se direktorij pod nazivom „build“, te se naredbom „*cd*“ navigira unutar njega. Zatim se pokreće naredba „*cmake*“ s nekoliko parametara gdje „-G“ označava odabir generatora, te se navodi putanja do source direktorija u kojoj će projektne datoteke biti generirane. Unutar direktorija je obavezno imati datoteku(konfiguraciju) „*CMakeLists.txt*“. Za potrebe diplomskog rada i jednostavniji i pregledniji proces kreiranja dinamičke biblioteke korišteno je grafičko sučelje prikazano na slici 3.9.



Sl. 3.9. Korištenje CMake grafičkog sučelja

3.5. MsBuild

MsBuild (*Microsoft Build Engine*) [14] je alat za izgradnju aplikacija razvijenih u Microsoft Visual Studio IDE-u. Omogućuje automatizaciju procesa kompilacije i upravljanja projektima što je ključno za učinkovito upravljanje velikim softverskim projektima. Za potrebe ovog diplomskog rada zbog jednostavnosti i dostupnosti materijala koristi se umjesto CMake-a koji je bio korišten samo za kreiranje dinamičke biblioteke. Koristi se za izgradnju projekata, generiranje izvršnih datoteka i biblioteka, te za provođenje testova osiguravajući dosljednost i kvalitetu u cijelom procesu razvoja programskog rješenja. Neke od ključnih značajki su: integracija s Visual Studiom, prilagodljivost, upravljanje ovisnostima i podrška za više platformi. MsBuild je integriran s Visual Studiom omogućujući jednostavno upravljanje projektima i rješavanje ovisnosti unutar integriranog razvojnog okruženja. Razvoj je ubrzan jer je moguće prilično jednostavno i brzo kreirati projekt i krenuti sa implementacijom. Podržava prilagodbu skripti za izgradnju omogućavajući korisnicima definiranje specifičnih zadataka kao što su kopiranje datoteka, izvršavanje testova i pokretanje drugih alata. MsBuild automatski upravlja ovisnostima među projektima,osiguravajući da se svi potrebni moduli pravilno kompiliraju u ispravnom redoslijedu. Za pokretanje izgradnje izvršne datoteke potrebno je pritisnuti kombinaciju tipki „*alt* + *shift* + *b*“ na tipkovnici ali također je moguće koristiti komandnu liniju za Visual Studio i pristup pomoću grafičkog sučelja pritiskom na „*Build*“ te zatim odabirom „*Build Solution*“ ili „*Rebuild Solution*“. Tijekom izrade diplomskog rada najčešće korištena opcija je bila kombinacija tipki na tipkovnici. Ako je uspješno izgrađena izvršna datoteka tada ispis u okviru „Output“ bude kao na slici 3.10. Također prikazuje upozorenja te u slučaju neuspjelog pokušaja u kartici „Eror List“ su navedeni problemi zbog kojih izgradnja nije bila moguća.



Sl. 3.10. Prikaz uspješne izgradnje izvršne datoteke

3.6. Vcpkg

Vcpkg [15] je alat za upravljanje paketima koji olakšava instalaciju i upravljanje bibliotekama u C++ projektima. Razvijen od strane Microsofta vcpkg omogućava jednostavnu integraciju vanjskih biblioteka u projekte automatizirajući proces preuzimanja, kompilacije i konfiguracije. Omogućava brzo preuzimanje i instalaciju širokog spektra C++ biblioteka pomoću jednostavnih naredbi. Podržava različite platforme poput Windows-a, macOS-a i Linux-a i zato omogućuje da projekti budu prenosivi i lako konfigurabilni na različitim sustavima. Poput CMake-a automatski upravlja ovisnostima između paketa olakšavajući integraciju složenih biblioteka bez potrebe za ručnim rješavanjem ovisnosti. Također se lako integrira s alatima kao što su CMake i msBuild tako da razvojni programeri ne moraju voditi brigu o sukobima ovih alata. Vcpkg se često koristi u razvoju C++ projekata gdje je potrebno brzo integrirati vanjske biblioteke. Primjerice dodavanje biblioteka za grafiku, mrežno programiranje ili analizu podataka može se obaviti jednostavnim naredbama čime se smanjuje vrijeme potrebno za postavljanje i konfiguraciju razvojnog okruženja. Instalacija Vcpkg-a je prilično jednostavna i moguće je koristiti Git Bash komandnu liniju da bi se klonirao repozitorij alata te se uz par naredbi može integrirati u Visual Studio. Poželjno je putanju Vcpgk instalacije odnosno direktorija postaviti u „*varijable okoline*“ kako bi Visual Studio mogao koristiti Vcpkg. Također je moguće instalirati alat kroz visual studio ali češća praksa je koristiti komandnu liniju tako da nije direktno vezano uz Visual Studio.

Prvi korak u postavljanju alata je korištenje naredbe prikazane na slici 3.11.

```
git clone https://github.com/microsoft/vcpkg.git
```

Sl. 3.11. Naredba za kloniranje repozitorija na lokalno računalo

Zatim je potrebno pokrenuti „*bootstrap-vcpkg.bat*“ skiptu koja automatski priprema i konfigurira vcpkg alat za korištenje. Ona provjerava preduvjete i osigurava da su potrebne postavke i alati dostupni na trenutnom sustavu, preuzima potrebne datoteke i postavlja izvršnu datoteku te zatim kompajlira vcpkg alat kako bi bio spreman za korištenje. S ovom skriptom se postiže brza instalacija bez potrebe za ručnom konfiguracijom. Zatim je potrebno instalirati instancu vcpkg za korisnika kako bi ju MSBuild mogao pronaći. Opisane naredbe su prikazane na slici 3.12.

```
PS C:\d disk\Diplomski WebGL> cd vcpkg && bootstrap-vcpkg.bat
>> .\vcpkg.exe integrate install
```

Sl. 3.12. Naredbe za integraciju vcpkg s MSBuidl-om

4. Implementacija programskog rješenja

Ovo poglavlje sadrži detaljno opisan razvoj programskog rješenja za prikaz digitalnog brzinomjera.

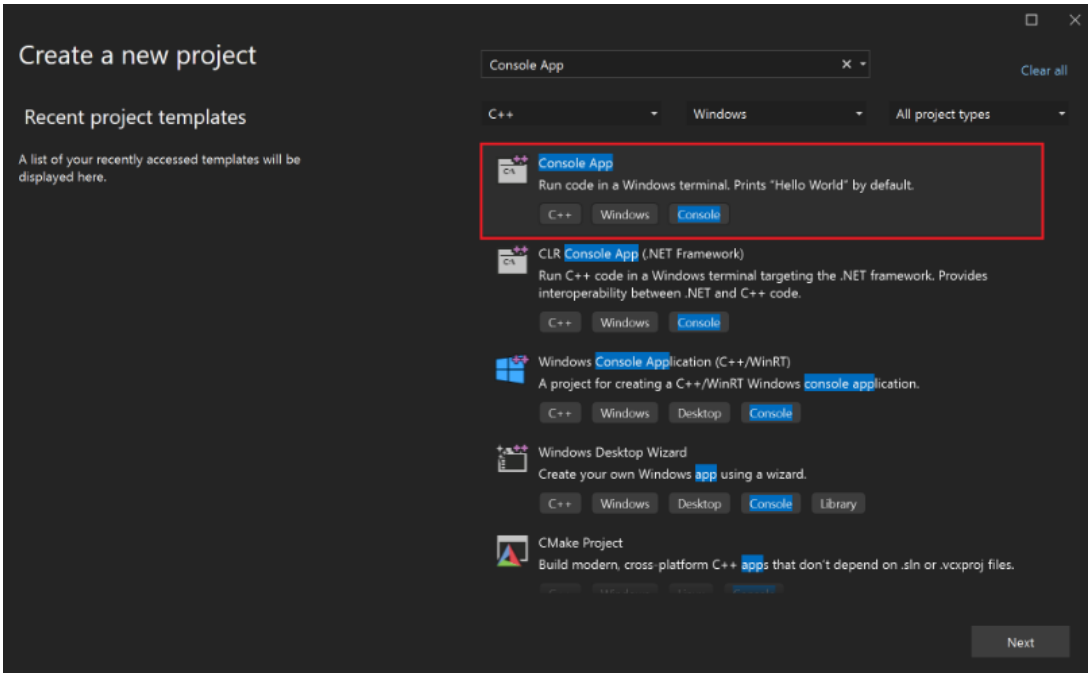
4.1. Postavljanje projekta

Za potrebe diplomskog rada odabrano integrirano razvojno okruženje je Microsoft Visual Studio opisano u prošlom poglavlju. Odabran je zbog svojih naprednih mogućnosti, jednostavnosti integracije i korištenja. Projekt nije jednostavan za postaviti zbog nekoliko korištenih biblioteka koje nisu mogle biti dodane koristeći samo CMake, MSBuild ili Vcpkg, te ih je potrebno koristiti pojedinačno kako bi biblioteke bile dostupne u projektu.

4.1.1. Kreiranje projekta

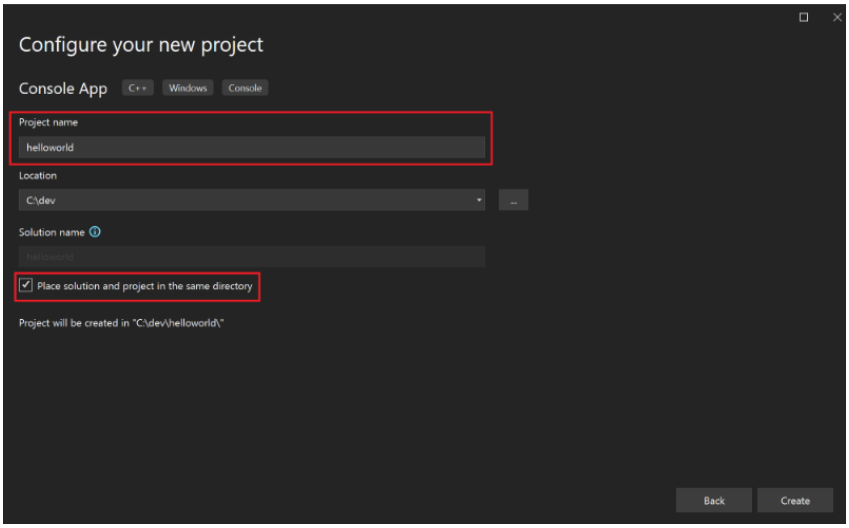
Za ispravno postavljanje projekta potrebno je izvesti konfiguraciju u nekoliko koraka:

1. Kreirati novi Visual Studio projekt. Ovaj korak uključuje odabir vrste projekta. Za ovaj diplomski rad odabrana je konzolna aplikacija (slika 4.1) samo kako bi mogli lakše pokrenuti izvršnu datoteku.



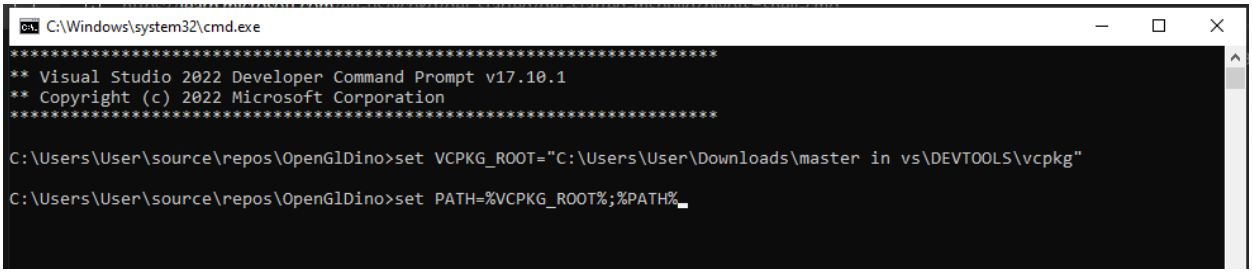
Sl. 4.1. Kreiranje nove konzolne aplikacije u C++ jeziku

2. Odabrati naziva projekta, lokaciju projekta i označiti izbor postavljanja projekta i Visual Studio solution-a u isti direktorij, prikazano na slici 4.2.



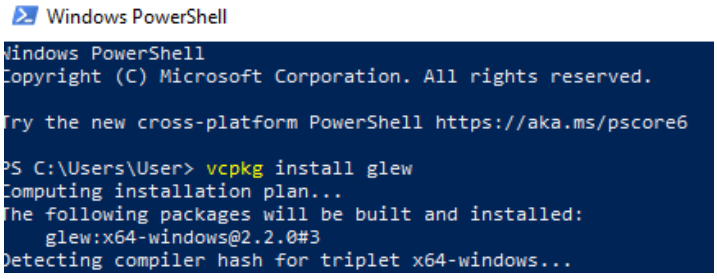
Sl. 4.2. Konfiguracija konzolne aplikacije

3. Nakon uspješnog kreiranja projekta Visual Studio automatski pokreće projekt te je potrebno u alatnoj traci izabrati „*Tools*“ zatim „*Command Line*“ i odabrati opciju „*Developer Command Prompt*“ i postaviti putanju instalacije vcpkg alata. Ovaj korak nije obavezan ako je vcpkg alat instaliran načinom opisanim u prošlom poglavlju, ali ako Visual studio ne prepozna vcpkg potrebno je upisati i pokrenuti naredbe prikazane na slici 4.3. i provjeriti je li putanja ispravno postavljena u varijablama sustava. Tako možemo alat pozivati u npr. Windows Powershell-u.



Sl. 4.3. Konfiguracija vcpkg alata u Visual Studio komandnoj liniji

4. Završni korak je pokrenuti projekt, odnosno izvršnu datoteku ali da bi projekt funkcionirao kako je zamišljen, potrebno je instalirati biblioteku kako bi verificirali da alat radi ispravno. Primjer naredbe za instalaciju „*GLEW*“ biblioteke je na slici 4.4.



Sl. 4.4. Instalacija *Glew* biblioteke

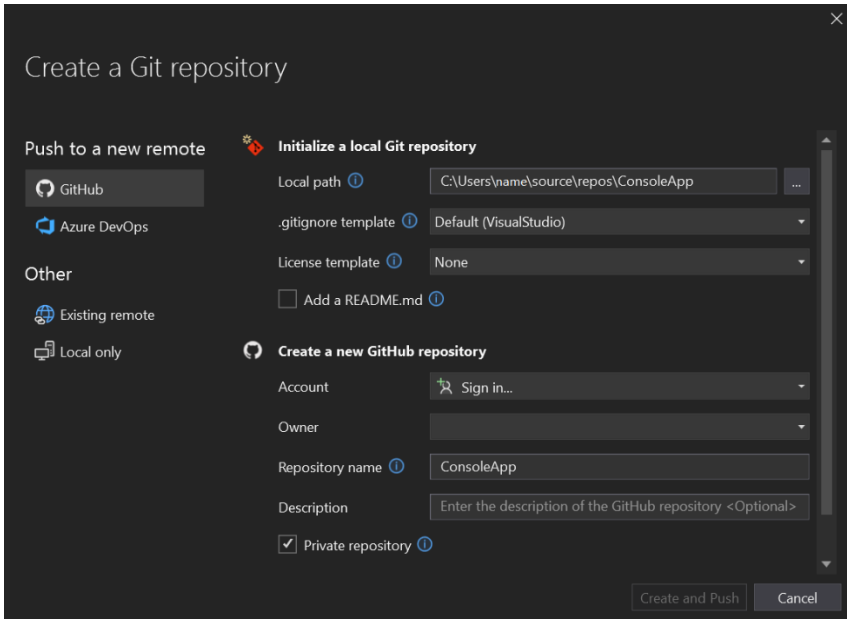
4.1.2. Instalacija dinamične biblioteke freeGLUT

Zbog problema prilikom pokušaja instalacije biblioteke sa vcpkg alatom, korišten je alat CMake koji je opisan u prošlom poglavlju. Za instalaciju biblioteke bilo je potrebno preuzeti datoteku sa službene stranice [16] te raspakirati kompresiranu datoteku i pokrenuti projekt u Visual Studio-u. Zatim je potrebno pokrenuti projekt odnosno pokrenuti izgradnju izvršne datoteke kako bi se dinamička biblioteka „*freeglut.dll*“ generirala. Cilj ovakvog pristupa je ponovna upotreba koda jer dinamička biblioteka(engl. *Dynamic Link Library*) omogućuje aplikacijama pristup funkcionalnostima freeGLUT-a bez potrebe za statičkim povezivanjem u vrijeme kompilacije. Više programa može koristiti istu DLL datoteku, što smanjuje ukupnu memoriju potrebnu za pokretanje. Funkcionalnosti su odvojene u vanjsku datoteku olakšavajući održavanje i ažuriranje bez ponovnog kompajliranja aplikacija. Cijeli postupak izrade diplomskog rada odvijao se u „*Debug*“ načinu kompilacije u Visual Studio-u jer uključuje dodatne informacije poput otkrivanja grešaka jer omogućuje lako praćenje i pronalazak problema u programskom kodu. Pruža mogućnost izvršavanja programskog koda korak po korak te praćenje varijabli i postavljanje točaka prekida(engl. *breakpoints)*. Nakon uspješnog pokretanja odnosno generiranja potrebno je pokrenuti alat CMake i postaviti putanju do izvornog koda i direktorij u kojem želimo generirane datoteke. Nakon pokretanja konfiguracije potrebno je pregledati i potvrditi postavke te kliknuti na „*Generate*“. Nakon uspješnog generiranja potrebno je pronaći dinamičku biblioteku u direktoriju s generiranim datotekama i dodati ju u Visual Studio projekt.

4.1.3. Dodavanje sustava za kontrolu verzije

Visual Studio ima puno funkcionalnosti koje također uključuju integraciju sa sustavom kontrole verzije „*Github*“, tako da je praktično koristiti integrirani alat da bi projekt imao mogućnost sigurnosnih kopija(engl. *backup*) i praćenje promjena što omogućuje povratak na određeno odabrano stanje. Implementacija je poprilično jednostavna jer je u bočnoj alatnoj traci dovoljno odabrati karticu „*Git Changes*“ te kliknuti na „*Create Git Repository...*“ prikazano na slici 4.5.

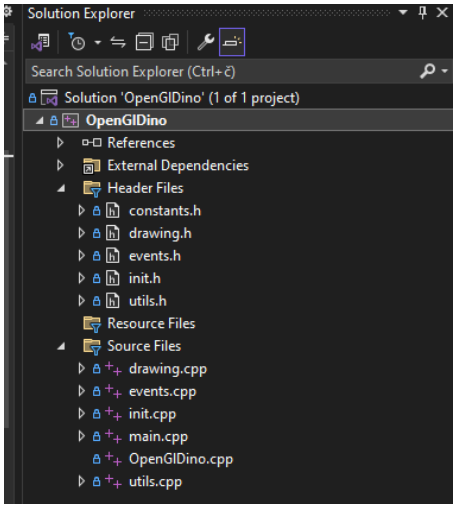
Kako bi imali kvalitetno konfiguriran sustav kontrole verzije potrebno je kreirati „*.gitignore*“ datoteku kako bi izbjegli praćenje nevažnih datoteka jer spriječava dodavanje privremenih ili generiranih datoteka koje nisu relevantne za projekt. Održava repozitorij organiziranim i čistim te je lakše pregledati programski kod. Također smanjuje veličinu repozitorija i ubrzava operacije. Još jedna bitna prednost je što štiti privatnost jer spriječava slučajno dijeljenje osjetljivih informacija, ovaj projekt ne sadrži takve informacije ali ako bi se kasnije htio proširiti korištenjem vanjskih API-a može zaštititi privatni ključ za pristup.



Sl. 4.5. Inicijalizacija GIT repozitorija

4.2. Struktura projekta

Radi bolje preglednosti, projekt je strukturiran u nekoliko direktorija, prikazano na slici 4.6.

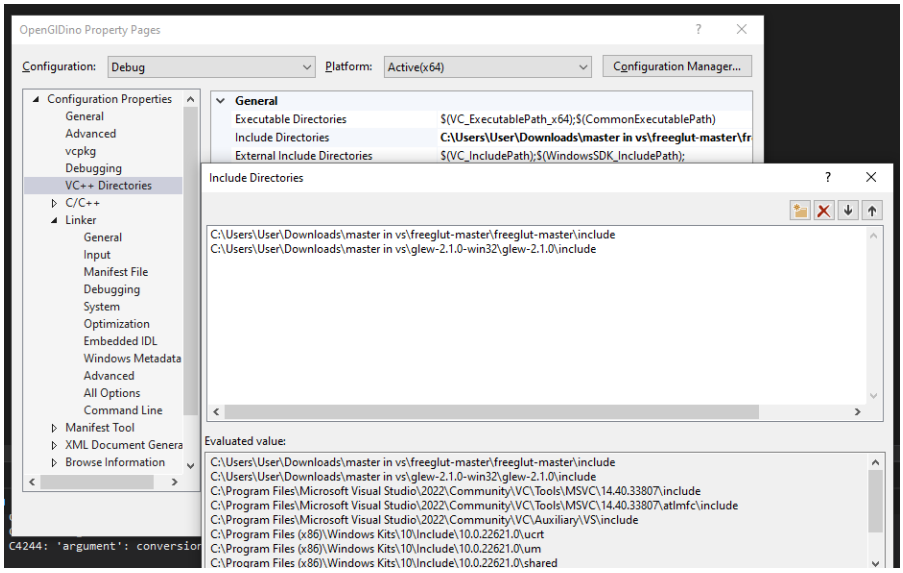


Sl. 4.6. Struktura Visual Studio projekta

Datoteke s ekstenzijom „*.h*“ su zaglavne datoteke i sadrže deklaracije funkcija, klasa i struktura. Omogućuju dijeljenje deklaracija između različitih dijelova koda. Datoteke s „*.cpp*“ ekstenzijom nazivaju se izvorne datoteke i sadrže implementacije funkcija i metoda te izvršavaju stvarni kod programa. Razdvajanjem je postignut organiziraniji kod, odnosno kompletan projekt.

Jasno definirana struktura olakšava pronalaženje i modificiranje datoteka i čini projekt i njegove datoteke čitkijim. Odvajanje koda u module i zaglavne datoteke omogućava jednostavno ponovno

korištenje funkcionalnosti u različitim dijelovima projekta. Strukturiranje pomaže u održavanju konzistentnosti u kontroli verzija i uvelike olakšava praćenje promjena. Strukturirani projekti se bolje prilagođavaju proširivanju odnosno dodavanju novih funkcionalnosti i omogućuju dodavanje na pregledniji i jednostavniji način. Potencijalni nedostatak je široka struktura gdje je razvojnom programeru teško upamtiti lokaciju funkcionalnosti ali zbog korištenja Visual Studia razvojni programer će dobiti povratnu informaciju u obliku pogreške (engl. *error*) ili upozorenja. Razvojno okruženje također nudi poboljšanja tijekom pisanja koda te može umjesto razvojnog programera uključiti potrebne datoteke. Iako je razvojno okruženje sposobno predlagati i samostalno povezivati funkcionalnosti, biblioteke i datoteke ono ponekad ne može pronaći neke biblioteke. Za potrebe ovog diplomskog rada potrebno je dodatno uključiti freeGLUT i GLEW biblioteke u postavkama projekta. To omogućuje kompajleru da pronađe potrebne zaglavne datoteke za te biblioteke tijekom kompilacije i osigurava da se deklaracije funkcija i struktura iz tih biblioteka mogu koristiti u programskom kodu bez pogrešaka. Postupak dodavanja biblioteka je prikazan na slici 4.7.

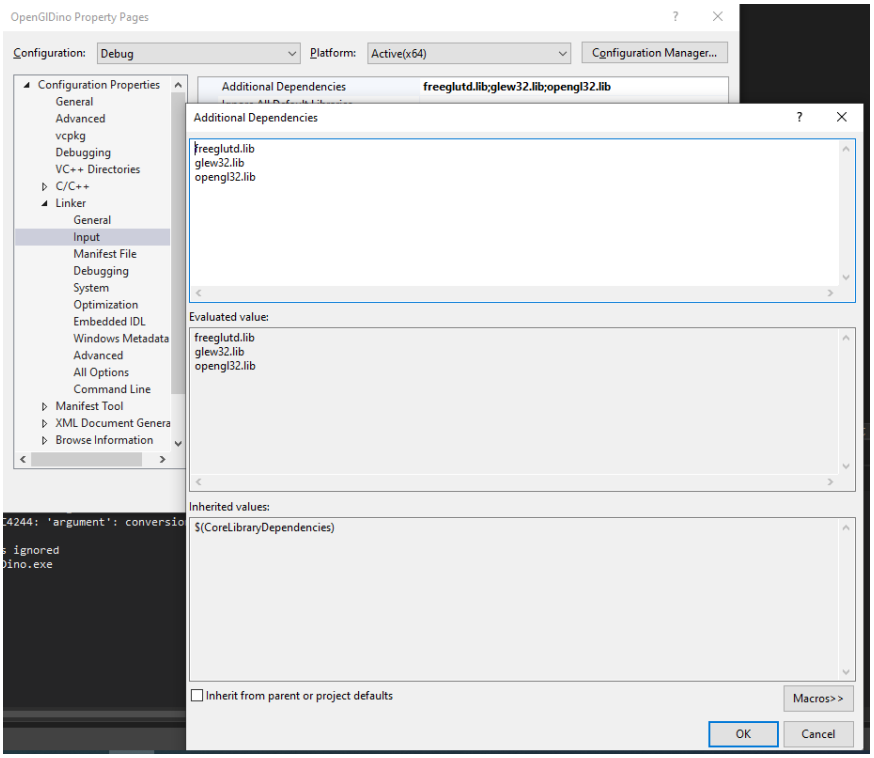


Sl. 4.7. Dodavanje vanjskih biblioteka u projekt

Potrebno je u postavkama projekta odabrati postavku „*VC++ Directories*“ te zatim kliknuti na „*Include Directories*“ i dodati putanje biblioteka.

Kako bi se biblioteke mogle koristiti, potrebno je koristiti alat „*Linker*“ koji kombinira različite objekte i biblioteke u jednu izvršnu datoteku ili biblioteku. Povezuje vanjske funkcije i varijable čime omogućuje da su svi pozivi funkcija pravilno povezani s njihovim definicijama. Također može optimizirati kod tako da uklanja neiskorištene funkcije i time smanjuje veličinu krajnje izvršne datoteke. Ako biblioteke nisu pravilno uključene u projekt razvojnom programeru je

prikazana greška poput „*unresolved external symbol*“ koja se javlja kada Linker ne može pronaći implementacije funkcija. Potrebno je u postavkama projekta odabrati postavku „*Linker*“ zatim kliknuti na „*Input*“ te u „*Additional Dependencies*“ dodati putanje „*Include*“ direktorija nedavno dodanih biblioteka. Primjer unosa je prikazan na slici 4.8. Zatim je potrebno ponovno izgraditi izvšnu datoteku kako bi se promjene primjenile i razvojni programer može koristiti nove funkcije. Prilikom daljnjeg rada na projektu, ako Vcpkg ne uspije instalirati novu biblioteku, potrebno je slijediti navedene korake za uspješno dodavanje u projekt.



Sl. 4.8. Dodavanje *include* direktorija u *Additional Dependencies*

4.3. Struktura i opis komponenata

Svaka komponenta igra ključnu ulogu u cjelokupnoj funkcionalnosti projekta i omogućuje organiziran i skalabilan razvoj, a njihovom jasnom podjelom olakšava se održavanje i proširenje programskog rješenja.

4.3.1. Glavna datoteka (main.cpp)

Ova datoteka služi kao ulazna točka aplikacije gdje se izvršava glavni tok programa. „*Main*“ funkcija je standardna ulazna točka u C++ programima [17]. Kada se aplikacija pokrene,operativni sustav prvo poziva ovu funkciju što omogućuje kontrolu nad cijelim tijekom programa od samog početka. Postavlja osnovne postavke aplikacije uključujući inicijalizaciju OpenGL-a i ostalih

potrebnih komponenti. Koristi se za stvaranje glavnog prozora aplikacije u kojem se prikazuju grafički elementi. Registrira funkcije odgovorne za obradu događaja kao što su unos s tipkovnice i miša. Pokreće beskonačnu petlju koja upravlja renderiranjem i obradom događaja osiguravajući da se grafičko sučelje stalno osvježava i reagira na korisnikove interakcije poput pritiska tipke na tipkovnici. Također sadrži pozive funkcija iz drugih modula (npr. crtanje i obrada događaja) što omogućuje modularni dizajn i razdvajanje odgovornosti.

4.3.2. Komponenta za crtanje

Uloga ove komponente je renderiranje i upravljanje grafičkim elementima koji su na zaslonu. Sastoji se od zaglavne (*drawing*.h) i izvorne datoteke (*drawing*.cpp) te implementira funkcije koje definiraju kako se objekti iscrtavaju na zaslonu. To uključuje postavljanje boja, pozicije i oblika objekata. Modul koristi OpenGL funkcije za kreiranje i prikazivanje grafike.

4.3.3. Komponenta za upravljanje događajima

Sastoji se od zaglavne (*events*.h) i izvorne (*events*.cpp) datoteke. Njena uloga je upravljanje korisničkim unosima koji mogu biti s tipkovnice ili miša računala na kojem se programsko rješenje pokreće. Omogućuje interakciju korisnika s tako što detektira korisnikov unos neovisno je li se dogodio s miša ili tipkovnice.

4.3.4. Inicijalizacijska komponenta

Sastoji se od zaglavne (*init*.h) i izvršne (*init*.cpp) datoteke. Uloga komponente je postavljanje početnih postavki aplikacije. Funkcionalnosti su joj konfiguracija OpenGL konteksta i postavljanje početnih parametara kao što su boja pozadine i dimenzije prozora. Također osigurava da su svi potrebni resursi i postavke ispravno postavljeni prije pokretanja glavne petlje programa.

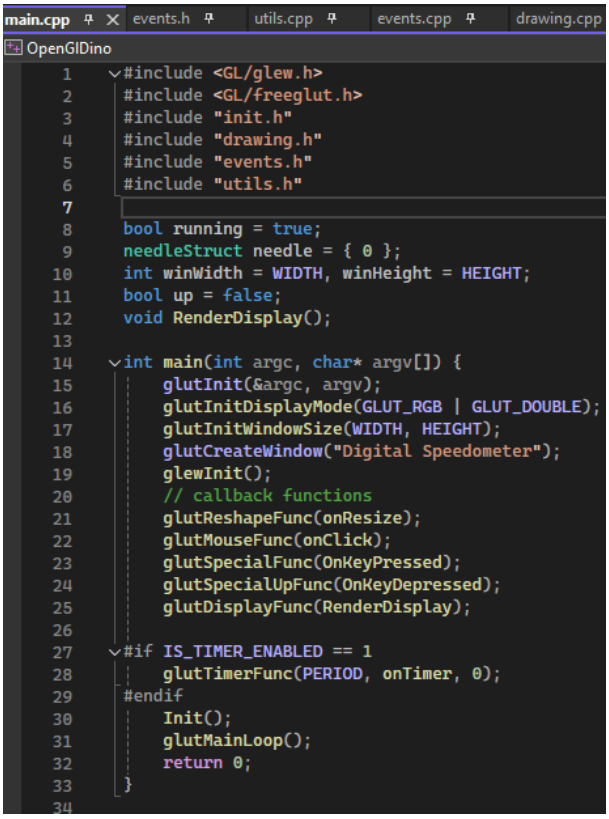
4.3.5. Komponenta pomoćnih funkcija

Ova komponenta sadrži razne pomoćne funkcije. Sastoji se od zaglavne (*utils*.h) i izvršne (*utils*.cpp) datoteke. Ove funkcije olakšavaju različite operacije u kodu poput matematičkih proračuna ili upravljanja memorijom. Pomažu u modularnosti koda i ponovnoj upotrebi omogućavajući čišći i organiziraniji razvoj.

4.4. Programski kod

4.4.1. Kod glavne komponente

Početni dio programskog rješenja prikaz koda nalazi se na slici 4.9.



Sl. 4.9. Programski kod glavne komponente (*main*.cpp)

Programski kod se može podijeliti na nekoliko dijelova počevši od uključivanja zaglavnih datoteka odnosno biblioteka (*glew*.h i *freeglut*.h) koje omogućuju korištenje OpenGL funkcionalnosti i upravaljanje prozorima. Preostale zaglavne datoteke sadrže deklaracije za funkcije i strukture koje su korištene u cijelom programu i već u samom početku ovakva struktura pridonosi čistijem i preglednijem kodu. Sadrži globalne varijable koje se koriste u cijelom programskom rješenju za iscrtavanje i kalkulacije dok varijabla „*running*“ održava stanje aplikacije tako što omogućuje interakciju korisnika s programom putem miša klikom na gumb „*Start*“ ili „*Stop*“ ovisno o stanju programa. Sadrži deklaraciju funkcije koja će služiti za iscrtavanje na zaslonu i koja je implementirana u komponenti za crtanje.

Glavna (*main*) funkcija inicijalizira i registrira callback funkcije. Da bi program mogao funkcionirati potrebno je inicijalizirati GLUT i GLEW biblioteku što se postiže pozivom funkcija „*glutInit*“ i „*glewInit*“. Zatim je potrebno postaviti prikaz na zaslonu i konfigurirati veličinu prozora, način prikaza i stvoriti prikaz s naslovom. Zbog razdvojenosti koda potrebno je registrirati

callback funkcije koje su definirane i implementirane u drugim datotekama odnosno dijelovima programa.

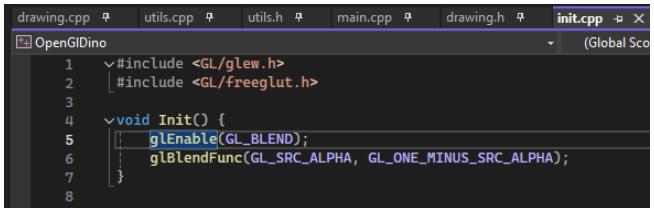
Registrirane funkcije i njihova uloga:

1. glutDisplayFunc(*RenderDisplay*) registrira funkciju za prikaz (iscrtavanje) na zaslonu.
2. glutReshapeFunc(*onResize*) registrira funkciju za promjenu veličine.
3. glutSpecialFunc(*onKeyPressed*) registrira funkciju za unos s tipkovnice.
4. glutSpecialUpFunc (*onKeyDepressed*) registrira funkciju za otpuštanje tipke s tipkovnice.
5. glutMouseFunc(*onClick*) registrira funkciju za klik mišem.

Važna funkcija koja postavlja timer ako je definiran odnosno postavljen je „glutTimerFunc“ koja prima pozitivnu cjelobrojnu vrijednost koja određuje vrijeme ponovnog okidanja funkcije vezane za timer u milisekundama. Također prima i funkciju koja će se izvoditi pri svakom okidanju timera. Poziv funkcije „*Init*“ služi za postavljanje početnog stanja programa te ne vraća vrijednost. Konačno je potrebno pokrenuti beskonačnu petlju koja upravlja događajima i renderiranjem pozivom „glutMainLoop“.

4.4.2. Kod komponente za inicijalizaciju

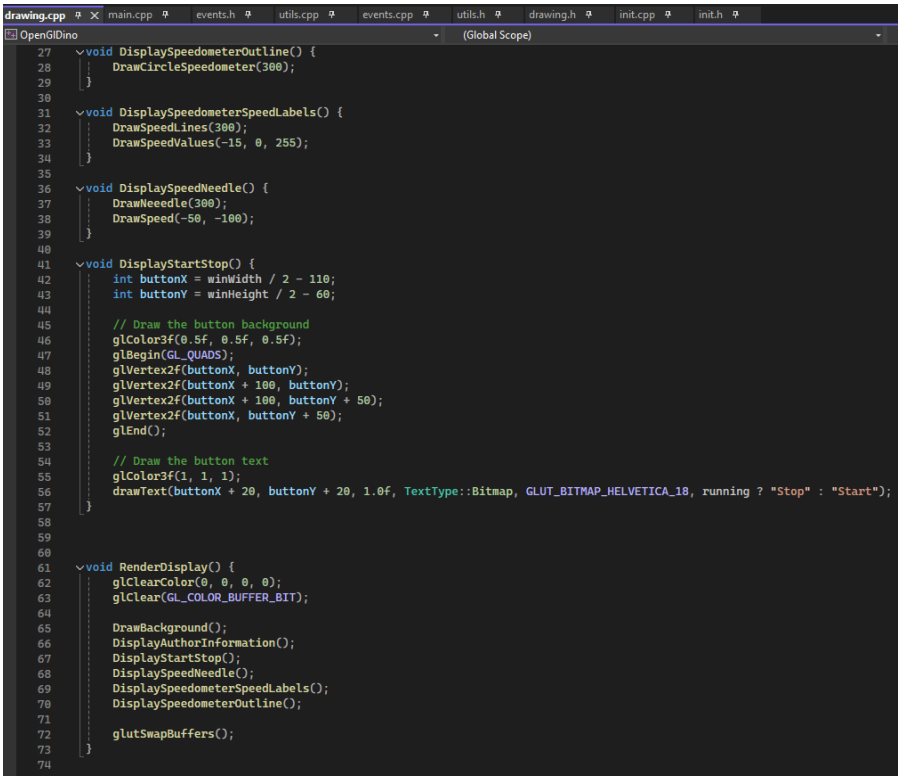
Datoteka uključuje zaglavne datoteke biblioteka „*freeGLUT*“ i „*Glew*“ pomoću naredbe „*include*“. Kako bi miješanje boja bilo moguće što je ključno za prikaz prozirnosti u objektima poziva se funkcija „*glEnable*“ koja prima parametar „GL_BLEND“ koji označava da se ta funkcionalnost želi uključiti. Slijedeća funkcija „*glBlendFunc*“ postavlja način miješanja boja gdje „GL_SRC_ALPHA“ definira faktor prozirnosti izvora dok „GL_ONE_MINUS_SRC_ALPHA“ definira faktor prozirnosti cilja. Odabrani način miješanja omogućuje stvaranje efekata prozirnosti i glatkih prijelaza između objekata, prikaz koda je na slici 4.10. Parametri koji se koriste u obje funkcije su tipa „*Glenum*“ koji je zapravo cjelobrojni tip podatka, te je definiran u zaglavnoj datoteci „*GLEW.h*“.



Sl. 4.10. Programski kod komponente za inicijalizaciju

4.4.3. Kod komponente za iscrtavanje

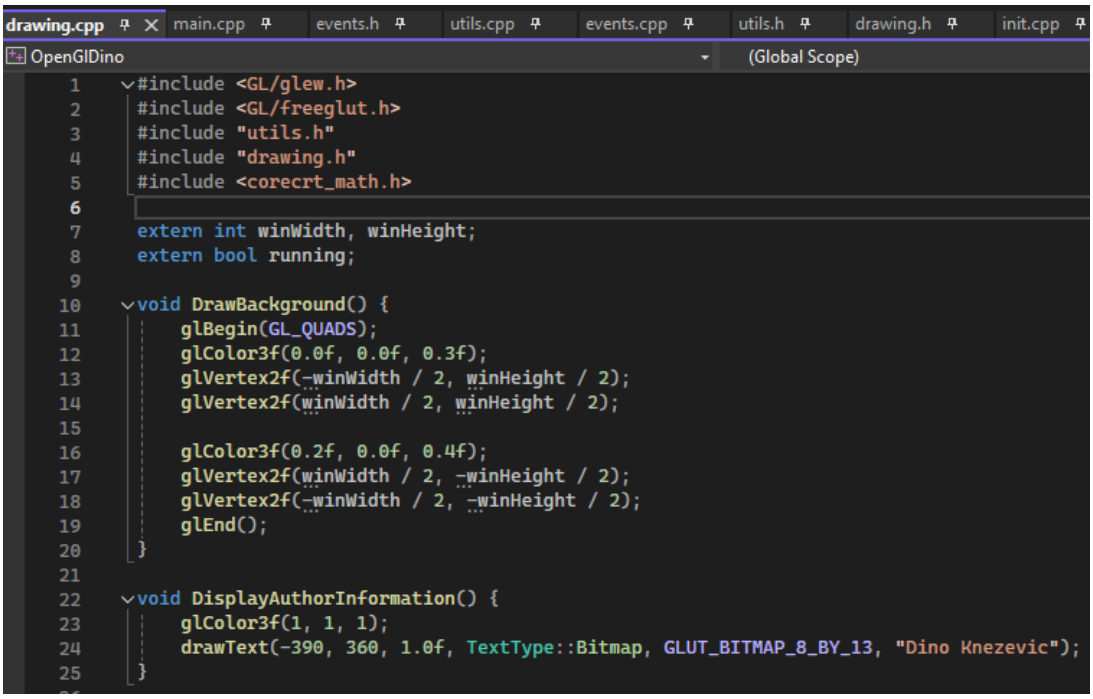
Komponenta koja u sebi sadrži implementaciju i pozive funkcija koje služe za iscrtavanje na zaslonu glavna funkcija koja se u ovoj komponenti implementira je „*RenderDisplay*“ te su u njoj objedinjeni pozivi ostalih funkcija čija je svrha iscrtavanje na zaslonu. Navedena funkcija se također poziva pri korisnikovoj interakciji u komponenti za događaje odnosno ako korisnik odluči promijeniti veličinu prozora tada je potrebno osvježiti prikaz kako bi korisničko iskustvo bilo ugodnije. Funkcije u ovoj komponenti zajedno omogućuju crtanje i prikaz svih grafičkih elemenata potrebnih za prikaz na zaslonu uključujući pozadinu, informacije o autoru, brzinomjer i gumb za pokretanje/zaustavljanje izvođenja programa. Na slici 4.11. su prikazane implementacije funkcija koje su potrebne za iscrtavanje brzinomjera i prikaza gumba za pokretanje/zaustavljanje izvođenja programa.



Sl. 4.11. Programski kod za iscrtavanje brzinometra u komponenti za iscrtavanje

Implementacija funkcije zaslužne za ispis na zaslonu („*RenderDisplay*“) se zasniva na principu korištenja prednjeg i stražnjeg međuspremnika. Prilikom svakog poziva funkcija „*glClear*“ čisti cijeli prikaz odnosno zaslon te poziva sve funkcije koje su zaslužne za crtanje različitih grafičkih elemenata na zaslonu. Završni korak funkcije je zamjena međuspremnika pozivom funkcije(„*glutSwapBuffers*“)[18]. Korišteni princip naziva se dvostruko međuspremanje jer koristi dva međuspremnika za prikaz slike na zaslonu, a sastoji se od prednjeg koji služi za prikaz slike

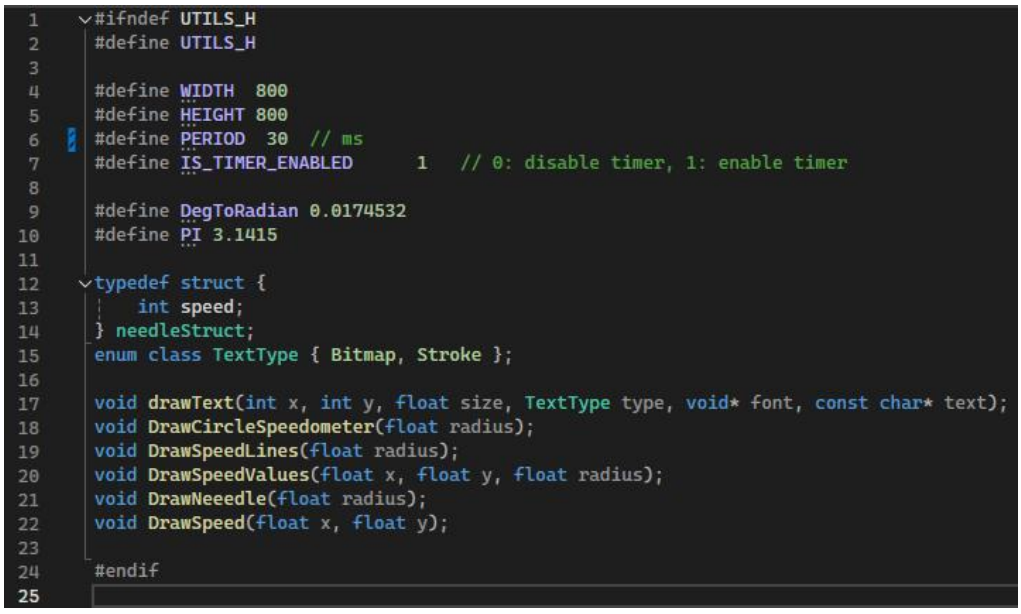
te stražnjeg spremnika u kojem se vrši crtanje odnosno renderiranje slike. Dok korisnik gleda sliku u prednjem međuspremniku nova slika se crta u stražnjem međuspremniku. Nakon što je crtanje završeno funkcija „*glutSwapBuffers*“ zamjenjuje stražnji međuspremnik s prednjim. Prednji međuspremnik sada prikazuje novu sliku dok stražnji međuspremnik postaje prazan i spreman je za novo crtanje. Ovakav način rada smanjuje treperenje jer korisnik nikada ne vidi proces crtanja već samo gotovu sliku. Kako bi neke funkcije za iscrtavanje mogle obavljati svoju funkcionalnost potrebno je naredbom za uključivanje uključiti zaglavne datoteke (*utils*.h i *drawing*.h) koje sadrže deklaracije funkcija i struktura koje se koriste ili implementiraju. Funkcije za iscrtavanje ne vraćaju vrijednost i njihova implementacija nije kompliciranа. Za osnovnu funkcionalnost potrebne su funkcije za iscrtavanje obrisa brzinomjera, pokazivača brzine, oznaka za brzinu i vrijednosti brzine. Funkcija za pokretanje/zaustavljanje („*DisplayStartStop*“) je u potpunosti implementirana u ovoj komponenti jer ne zahtjeva dodatne funkcionalnosti poput funkcija koje su zaslužne za ispis brzinomjera. Prvi korak je izračun položaja gumba i njihov izračun se zasniva na jednostavnim računicama. Koordinata za x-os se računa kao polovica širine prozora od čega se oduzima 110 piksela kako bi se gumb pozicionirao blizu desnog ruba prozora. Za izračun koordinate na y-osi potrebno je od polovice visine prozora oduzeti 60 piksela i gumb je pozicioniran u gornjem desnom kutu prozora. Za crtanje pravokutnika je potrebno pozvati funkciju „*glBegin*“ koja prima parametar „*GL_QUADS*“ koji je definiran u „*GLEW*“ biblioteci i označava crtanje pravokutnika. Zatim je za svaki vrh potrebno pozvati funkciju „*glVertex2f*“ koja prima dva parametra koji sadrže vrijednosti x i y koordinata. Kako bi označili kraj niza primitiva koje se žele iscrtati na zaslonu potrebno je pozvati funkciju „*glEnd*“. Na slici 4.12. je prikazan programski kod za prikaz informacija o autoru i iscrtavanje pozadine prozora u boji. Funkcija „*DrawBackground*“ koristi dva različita poziva „*glColor3f*“ kako bi postavila različite boje za gornje i donje vrhove četverokuta. Rezultat je gradijent boje koji prelazi iz tamno plave (gornji dio zaslona) u ljubičastu (donji dio zaslona). Rezultat je glatki prijelaz boja kako bi grafičko sučelje odnosno zaslon bio privlačniji korisniku i kako bi brzinomjer bio istaknutiji i u fokusu korisnika. Funkcije za crtanje brzinomjera u sebi pozivaju funkcije koje su implementirane u komponenti pomoćnih funkcija zbog složenije implementacije i preglednosti.



Sl. 4.12. Programski kod za iscrtavanje pozadine i informacija o autoru

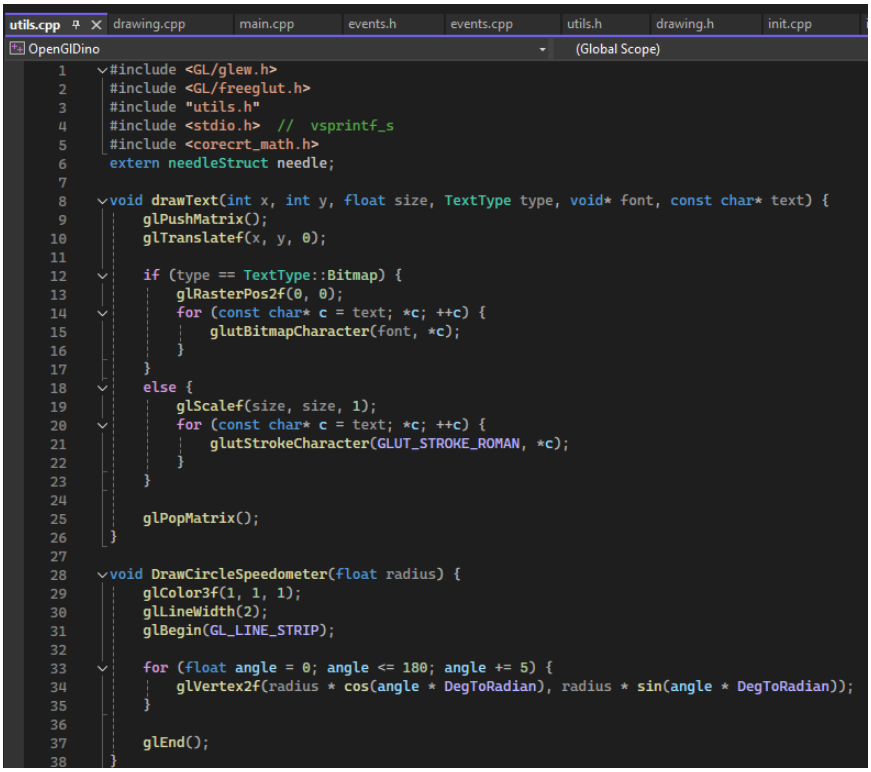
4.4.4. Kod komponente pomoćnih funkcija

U ovom dijelu su prikazane i objašnjene funkcionalnosti i implementacija pomoćnih funkcija i funkcija za iscrtavanje koje se pozivaju u komponenti za iscrtavanje. Sastoji se od zaglavne datoteke „*utils.h*“ i izvorne datoteke „*utils.cpp*“. Zaglavna datoteka definira pomoćne funkcije, deklaraciju struktura, konstanti te definira funkcije za iscrtavanje koje se koriste u drugim mjestima u programskom kodu. Njen kod je prikazan na slici 4.13.



Sl. 4.13. Programski kod zaglavne datoteke komponente pomoćnih funkcija

Konstante su definirane kako bi kod bio pregledniji i ako bude potrebno prilagođavati vrijednosti može se promijeniti na jednom mjestu. Konstanta „*DegToRadian*“ služi za pretvaranje stupnjeva u radijane. Struktura „*needleStruct*“ je novi tip podatka koji predstavlja pokazivač koji u osnovnoj implementaciji sadrži cjelobrojni podatak o brzini. Programski kod pomoćnih funkcija za ispis teksta i funkcije za iscrtavanje oblika brzinomjera je prikazan na slici 4.14.



Sl. 4.14. Programski kod pomoćnih funkcija i funkcije iscrtavanja brzinomjera

Funkcija „drawText“ je odgovorna za ispisivanje tekstualnog sadržaja na zaslonu i omogućuje upotrebu dviju vrsti teksta: rasteriziranog(bitmapnog) i vektorskog. Prima koordinate gdje će tekst biti prikazan, parametar „*size*“ koji se koristi za skaliranje teksta ako je odabran vektorki tip teksta. Također prima oznaku za tip teksta koji će biti prikazan(„*enum*“ tip podatka) te pokazivač na font koji će se koristiti za prikaz teksta i niz znakova koji predstavljaju tekst koji će biti prikazan. Pozivom funkcije „*glRasterPos2f*“, tekst se postavlja na poziciju za ispisivanje te je kroz petlju koja će proći kroz svaki znak u formatiranom nizu i pomoću funkcije „glutBitmapCharacter“ ispisati taj znak. Kako bi funkcija mogla fleksibilno i precizno pozicionirati i skalirati tekst potrebno je koristiti funkcije za manipulaciju matricom OpenGL-a te funkcije za pozicioniranje i skaliranje teksta. Korištenjem funkcija „*glPushMatrix“* i „*glPopMatrix*“ bilo koje transformacije koje se primjenjuju unutar ovih poziva su izolirane i osigurano je da promjene transformacija ne utječu na ostatak scene odnosno prikaza jer želimo utjecati samo na promjenu teksta. Funkcije „*glTranslatef*“i „*glScalef*“ omogućuju precizno pozicioniranje i skaliranje teksta. Bez ovih

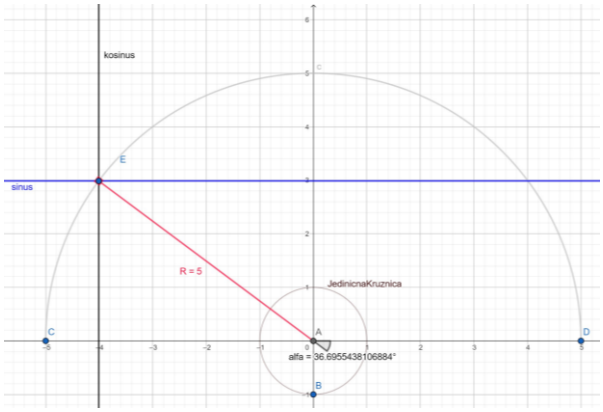
transformacija tekst se ne bi mogao ispisati na zadanoj poziciji u zadanoj veličini. Kako bi se izbjegao gubitak kvalitete unutar petlje za ispisivanje znakova svaki znak se ispisuje funkcijom „*glutStrokeCharacter*“ koja omogućuje skaliranje jer se temelji na vektorskim definicijama.

Iscrtavanje brzinomjera se odvija pomoću funkcije „*DrawCircleSpeedometer*“ koji prima realni broj za radijus. Za iscrtavanje je potrebno postaviti boju koristeći „*glColor3f*“ koja prima parametre po redoslijedu za crvenu, zelenu i plavu boju te ih miješa. Potrebno je označiti početak povezanog niza linijskih segmenata za iscrtavanje gdje je svaki segment povezan s prethodnim predajom parametra „*GL_LINE_STRIP*“. Za crtanje kružnog oblika potrebno je koristiti petlju koja će proći kroz kutove od 0 do 180 stupnjeva u koracima od 5 stupnjeva. Za potrebe rada odabran je polukrug kao oblik analognog prikaza. OpengGL koordinatni sustav u svom središtu ima koordinate (0,0) te je pomoću izračuna kuta između radijusa i x-osi moguće crtati polukrug koristeći trigonometrijske funkcije za izračun koordinata točaka brzinomjera. Korištenjem jedinične kružnice moguće je odrediti x i y koordinate koje će služiti za iscrtavanje. Korištenjem funkcije kosinus koja na temelju kuta koji je preračunat u radijane se izračunava koordinata x dok se za koordinatu y koristi trigonometrijska funkcija sinus. Da bi brzinomjer mogao biti prikazan u proizvoljnoj skali, odnosno na različitim zaslonima koordinate je potrebno pomnožiti s radijusom. Slika 4.15. prikazuje jednostavno korištenje trigonometrijskih funkcija za određivanje koordinata pomoću kuta. Funkciji za crtanje vrha potrebno je predati izračunate koordinate x i y. Koordinata x se računa prema formuli:

$$x = r * cos(angle * D2R) + x \tag{4-1}$$

Koordinata y se računa prema formuli:

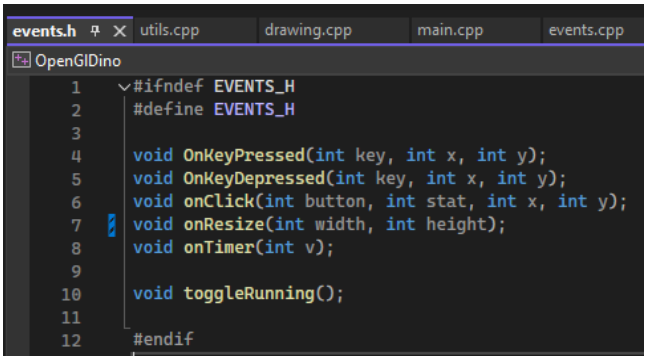
$$y = r * sin(angle * D2R) + y \tag{4-2}$$



Sl. 4.15. Prikaz izračuna koordinata pomoću trigonometrijskih funkcija

4.4.5. Kod komponente za upravljanje događajima

Deklaracije funkcija u zaglavnoj datoteci „*events.h*“ su prikazane na slici 4.16.



Sl. 4.16. Deklaracija funkcija za upravljanje događajima

Za postizanje kvalitetnog korisničkog iskustva potrebno je implementirati funkciju koja se poziva kada se promijeni veličina prozora, implementacija prikazana na slici 4.17.

```
void onResize(int width, int height) {
    winWidth = width;
    winHeight = height;
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-width / 2, width / 2, -height / 2, height / 2, -1, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    RenderDisplay();
}
```

Sl. 4.17. Programski kod funkcije odgovorne za prilagodbu prikaza

Naredbom „*glViewport*“ postavlja se prikazni prozor tako da se proteže od kuta (0,0) do novih koordinata (w,h). Zatim se poziva „*glMatrixMode*“ koja se koristi za definiranje karakteristika kamere i postavlja ju u način rada s projekcijom te se slijedećom naredbom „*glLoadIdentity*“ uklanjaju sve prethodne transformacije. Naredba „*glOrtho*“ postavlja ortografsku projekciju gdje su granice postavljene tako da centriraju prikaz na (0,0) sa širinom „*width*“ i visinom „*height*“. Nakon što je prikaz ispravno postavljen potrebno je vratiti matricu na način rada „*model-matrica*“ i resetirati trenutnu matricu. Za simuliranje kretanja kazaljke i prikaza brzine koristi se navigacijska tipka „*gore*“(engl. „*up*“) na tipkovnici te se za prikaz usporavanja detektira otpuštanje tipke. Funkcije za upravljanje navedenim događajima su prikazane na slici 4.18.

```
12
13  void OnKeyPressed(int key, int x, int y) {
14      switch (key) {
15      case GLUT_KEY_UP: up = true; break;
16      }
17      glutPostRedisplay();
18  }
19
20  void OnKeyDepressed(int key, int x, int y) {
21      switch (key) {
22      case GLUT_KEY_UP: up = false; break;
23      }
24      glutPostRedisplay();
25  }
26
```

Sl. 4.18. Programski kod funkcija za detekciju pritiska tipki odgovornih za simulaciju

Funkcija prima cjelobrojnu vrijednost koja označava pritisnutu tipku te ako je pritisnuta odgovarajuća tipka postavlja vrijednost varijable „*up*“ na istinito i označava da trenutni zaslon treba ponovno iscrtavanje kako bi se vidile promjene. Analogno vrijedi i kada se tipka otpusti.

Funkcija odgovorna za pauziranje/nastavljanje izvođenja simulacije je prikazana na slici 4.19.

```
26
27  void onClick(int button, int stat, int x, int y) {
28      // Convert window coordinates to OpenGL coordinates
29      int oglX = x - winWidth / 2;
30      int oglY = winHeight / 2 - y;
31
32      // Coordinates for upper right corner
33      int buttonX = winWidth / 2 - 110;
34      int buttonY = winHeight / 2 - 60;
35
36      // Check if the click is within the start/stop button area
37      if (button == GLUT_LEFT_BUTTON && stat == GLUT_DOWN) {
38          if (oglX > buttonX && oglX < buttonX + 100 && oglY > buttonY && oglY < buttonY + 50) {
39              toggleRunning();
40          }
41      }
42      glutPostRedisplay();
43  }
44
```

Sl. 4.19. Programski kod funkcije odgovorne za pauziranje/nastavljanje izvođenja programa

Potrebno je pretvoriti koordinate prozora u OpenGL koordinate i odrediti lokaciju gumba koji se nalazi u gornjem desnom kutu zaslona. Zatim je potrebno provjeriti je li pritisnuta lijeva tipka miša i je li se klik dogodio unutar područja gumba. Pomoću funkcije „*onTimer*“ koja se poziva na svaki periodični vremenski interval („*TIMER_PERIOD*“) definiran u zaglavnoj datoteci pomoćnih funkcija. Ako je program u stanju „*running*“(vrijednost varijable je istinita) tada se povećava kut igle što označava promjenu brzine. Za realističniji prikaz ubrzanja i usporavanja kut povećanja igle je ograničen izračunom koji ovisno radi li se o ubrzanju ili usporavanju ograničava kut na 8 odnosno -4 stupnja. Izračun povećanja kuta pri ubrzanju je definiran sljedećom formulom:

$$y = (316 - currentAngle)/20 \tag{4-3}$$

Izračun smanjenja kuta pokazivača pri usporavanju je definiran sljedećom formulom:

$$y = (-currentAngle)/80 \quad (4\text{-}3)$$

Iznosi s kojima se dijeli trenutni kut(brzina) su dobiveni eksperimentalnim načinom. Za ograničenje iznosa promjene kuta su korištene funkcije „*std::min*“ i „*std::max*“ koje vraćaju manju od dvije proslijeđene vrijednosti, odnosno veću ako se koristi „*std::max*“. Programski kod za prethodno opisane funkcionalnosti je prikazan na slici 4.20.

```
67  float calculateIncrement(bool accelerating, float currentAngle) {
68      if (accelerating) {
69          // simulate acceleration
70          return std::min(8.0f, (316.0f - currentAngle) / 20.0f);
71      }
72      else {
73          //  simulate deceleration
74          return std::max(-4.0f, -currentAngle / 80.0f);
75      }
76  }
77
78
79  #if TIMER_ON == 1
80  void onTimer(int v) {
81      glutTimerFunc(TIMER_PERIOD, onTimer, 0);
82
83      if (running) {
84          float increment = calculateIncrement(up, needle.speed);
85          needle.speed += increment;
86
87          if (needle.speed < 0) needle.speed = 0;
88          if (needle.speed > 316) needle.speed = 316 +60;
89      }
90
91      glutPostRedisplay();
92  }
93   #endif
94
95  void toggleRunning() {
96      running = !running;
97  }
98
```

Sl. 4.20. Programski kod funkcije odgovorne za pomak pokazivača

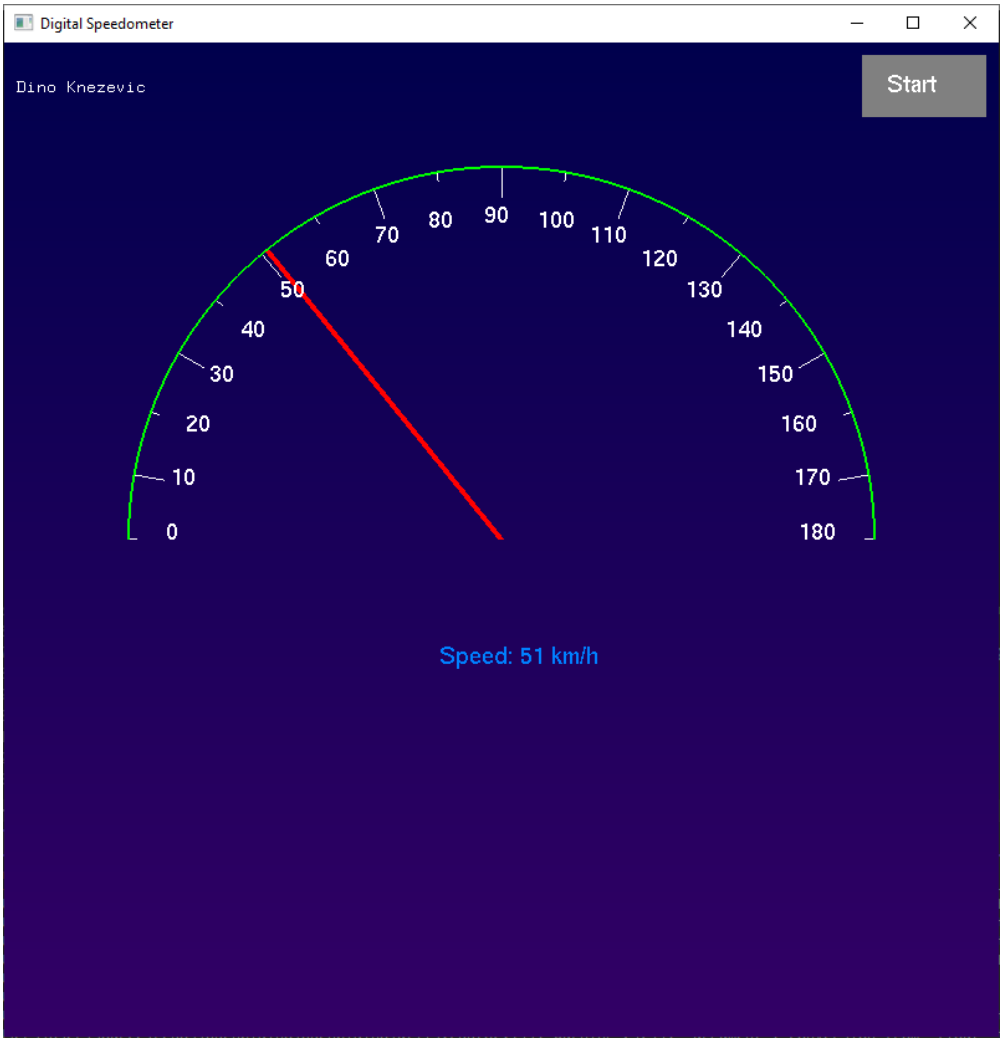
Za crtanje alternirajućih linija na brzinomjeru ispod kojih je prikazan iznos brzine koje služe za lakši odnosno ugodniji prikaz čovjeku potrebno je odrediti početnu točku na polukrugu. Također je potrebno odrediti dužinu crte za što se koristi ternarni operator „?“ te je nakon odabira početne točke linije potrebno od radijusa oduzeti izračunati dužinu crte kako bi se dobile krajnje koordinate za iscrtavanje linije. Programski kod funkcije za iscrtavanje prikazn je na slici 4.21.

```
40  void DrawSpeedLines(float radius) {
41      glColor3f(1, 1, 1);
42      glLineWidth(1);
43      glBegin(GL_LINES);
44      for (float angle = 0; angle <= 180; angle += 10) {
45          float lineLength = (static_cast<int>((angle +10) / 10) % 2 == 0) ? 25 : 7.5; // Alternate line lengths
46          glVertex2f(radius * cos(angle * DegToRadian), radius * sin(angle * DegToRadian));
47          glVertex2f((radius - lineLength) * cos(angle * DegToRadian), (radius - lineLength) * sin(angle * DegToRadian));
48      }
49      glEnd();
50  }
```

Sl. 4.21. Programski kod funkcije odgovorne za pomak pokazivača

4.5. Grafičko sučelje programa

Grafičko sučelje programa (engl. kratica „GUI“) služi kao most između korisnika i aplikacije te omogućuje korisnicima interakciju s programom putem vizualnih elemenata poput gumba i pritiska tipki s tipkovnice. Za potrebe diplomskog rada koristi se „FreeGLUT“ i „GLEW“ kao osnovne alate za upravljanje OpenGL kontekstom i renderiranjem te su također dobar odabir za izradu sučelja s kojim korisnik može upravljati programom. Zaslon se sastoji od prikaza informacija o autoru, gumba s kojim se može zaustavljati ili pokretati program te brzinomjera koji je prikazan kao analogan i promjenjivog teksta koji sadrži informaciju o trenutnoj brzini. Izgled programa je prikazan na slici 4.22.



Sl. 4.22. Prikaz digitalnog brzinomjera

5. Zaključak

U ovom radu izrađena je računalna aplikacija koja omogućuje vizualizaciju brzine vozila putem grafičkog prikaza brzinomjera. Aplikacija je implementirana korištenjem OpenGL-a, C++ programskog jezika te GLEW i freeGLUT biblioteka koje su omogućile jednostavniju integraciju i razvoj grafičkih elemenata. U teorijskom dijelu rada opisani su alati i tehnologije korištene pri implementaciji kao i osnovne matematičke i grafičke transformacije potrebne za prikaz kazaljke brzinomjera i interakciju korisnika s aplikacijom.

Aplikacija omogućuje korisniku simulaciju ubrzavanja i usporavanja dok kazaljka brzinomjera prikazuje trenutnu brzinu. Glavne funkcionalnosti uključuju iscrtavanje brzinomjera, kontrolu simulacije putem tipkovnice i grafički prikaz relevantnih informacija na zaslonu. Korišteni su osnovni koncepti rada s dvodimenzionalnom grafikom te trigonometrijske funkcije za pravilno postavljanje kazaljke na brzinomjeru.

Ovaj projekt može se dodatno unaprijediti kroz dodavanje naprednih grafičkih efekata kao što su glatke animacije kazaljke i realistično sjenčanje. Također je moguće proširiti funkcionalnosti aplikacije dodavanjem različitih stilova brzinomjera, dodavanjem brojača okretaja motora kao i poboljšanjem korisničkog sučelja kako bi bilo intuitivnije i vizualno atraktivnije.

Literatura

[1] E., Sobey, E. J. C., Sobey, *A Field Guide to Automotive Technology*. Chicago Review Press, 2009.

[2] D., Sellin, „Visions of progress: the past and future of digital car dashboards“ [online], 10-ruj-2020. Dostupno na: https://rightware.com/blog/visions-of-progress-the-past-and-future-of-digital-car-dashboards/. [Pristupljeno: 29.6.2024.].

[3] „Ask the experts: Speedometers and speed cameras“ [online], 11-velj-2016. Dostupno na: https://web.archive.org/web/20160211080229/http://www.drive.com.au/motor-news/ask-the-experts-speedometers-and-speed-cameras-20120411-1wsqp.html. [Pristupljeno: 29.6.2024.].

[4] M. T., Naing, T., Tun, C., Saldanha, „Design of Speedometer and Recording System for Diesel Electric Locomotive“, MERAL Portal.

[5] J., Ondruš, M., Gogola, K., Čulík, R., Kampf, L., Bartuška, „Speedometer reliability in regard to road traffic sustainability“, *Open Eng.*, izd. 1, sv. 11, str. 1059–1068, sij. 2021.

[6] „Boat Speedometer: The Essentials“ [online], 20-srp-2024. Dostupno na: https://themarineking.com/blogs/news/boat-speedometer. [Pristupljeno: 12.9.2024.].

[7] „Visual Studio 2022 | Download for free“ [online]. Dostupno na: https://visualstudio.microsoft.com/vs/. [Pristupljeno: 29.6.2024.].

[8] „Build software better, together“ [online]. Dostupno na: https://github.com. [Pristupljeno: 29.6.2024.].

[9] Bjarne Stroustrup, *The C programming language*. Addison-Wesley, 1997.

[10] „OpenGL - The Industry Standard for High Performance Graphics“ [online]. Dostupno na: https://www.opengl.org/. [Pristupljeno: 29.6.2024.].

[11] „LearnOpenGL - Anti Aliasing“ [online]. Dostupno na: https://learnopengl.com/Advanced-OpenGL/Anti-Aliasing. [Pristupljeno: 29.6.2024.].

[12] „GLEW: The OpenGL Extension Wrangler Library“ [online]. Dostupno na: https://glew.sourceforge.net/index.html. [Pristupljeno: 29.6.2024.].

[13] „CMake - Upgrade Your Software Build System“ [online]. Dostupno na: https://cmake.org/. [Pristupljeno: 29.6.2024.].

[14] ghogen, „MSBuild Tutorial: Install and create a project - MSBuild“ [online], 17-lis-2023. Dostupno na: https://learn.microsoft.com/en-us/visualstudio/msbuild/walkthrough-using-msbuild?view=vs-2022. [Pristupljeno: 30.6.2024.].

[15] „vcpkg - Open source C/C++ dependency manager from Microsoft“ [online]. Dostupno na: https://vcpkg.io/en/. [Pristupljeno: 30.6.2024.].

[16] „freeglut“ [online], 11-lip-2024. Dostupno na: https://sourceforge.net/projects/freeglut/. [Pristupljeno: 30.6.2024.].

[17] „Main function - cppreference.com“ [online]. Dostupno na: https://en.cppreference.com/w/cpp/language/main_function. [Pristupljeno: 30.6.2024.].

[18] „4.6 glutSwapBuffers“ [online]. Dostupno na: https://www.opengl.org/resources/libraries/glut/spec3/node21.html. [Pristupljeno: 1.7.2024.].

Sažetak

U ovom radu razvijena je aplikacija koja simulira digitalni brzinomjer koristeći OpenGL tehnologiju. U teorijskom dijelu rada objašnjene su osnovne tehnologije korištene u razvoju aplikacije uključujući rad s grafičkim bibliotekama poput OpenGL-a , GLUT-a i GLEW-a. Poseban naglasak stavljen je na implementaciju grafičkog korisničkog sučelja (GUI) i upravljanje događajima kako bi se omogućila interakcija korisnika s aplikacijom.

Aplikacija prikazuje osnovne elemente digitalnog brzinomjera uključujući brzinsku skalu, kazaljku koja pokazuje trenutnu brzinu te start/stop dugme koje omogućava pokretanje i zaustavljanje simulacije. Kroz analizu koda objašnjeni su ključni dijelovi implementacije uključujući inicijalizaciju, iscrtavanje te upravljanje događajima. Implementacija je demonstrirala upotrebu OpenGL-a za stvaranje vizualnih efekata i interaktivnog korisničkog iskustva. U procesu razvoja identificirani su izazovi povezani s preciznim upravljanjem grafičkim elementima i optimizacijom performansi aplikacije. Iako su svi funkcionalni zahtjevi ispunjeni mogućnosti za daljnja poboljšanja uključuju optimizaciju renderiranja i proširenje funkcionalnosti aplikacije kako bi podržala dodatne opcije prilagođavanja korisničkog sučelja i napredne vizualne efekte.

Ključne riječi: brzinometar, GLEW, GLUT, OpenGL

Abstract

In this thesis, an application that simulates a digital speedometer using OpenGL technology was developed. The theoretical part of the work explains the basic technologies used in the development of the application, including working with graphical libraries such as OpenGL, GLUT, and GLEW. Special emphasis was placed on the implementation of the graphical user interface (GUI) and event handling to enable user interaction with the application.

The application displays the basic elements of a digital speedometer, including a speed scale, a needle indicating the current speed, and a start/stop button that allows the simulation to start and stop. Through code analysis, key parts of the implementation were explained, including initialization, rendering, and event handling. The implementation demonstrated the use of OpenGL for creating visual effects and an interactive user experience. During development, challenges related to precise control of graphical elements and optimizing application performance were identified. Although all functional requirements were met, opportunities for further improvement include optimizing rendering and expanding the application's functionality to support additional customization options for the user interface and advanced visual effects.

Keywords: GLEW, GLUT, OpenGL, speedometer

Životopis

Dino Knežević rođen je 13. srpnja 2000. godine u Našicama. Pohađao je osnovnu školu u Zdencima i školi Ivana Gorana Kovačića u razdoblju od 2007. do 2015 godine. Zatim upisuje smjer tehničar za elektroniku u Srednjoj školi Isidora Kršnjavoga u Našicama koju pohađa od 2015. do 2019. godine. Nakon završene srednje škole upisuje sveučilišni preddiplomski studij računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku 2019. godine. Preddiplomski studij završava 2022. godine te iste godine upisuje sveučilišni diplomski studij računarstva izborni blok informacijske tehnologije i podatkovne znanosti. Od 2023. godine zaposlen je u Mono d.o.o kao software developer.

_____

Potpis autora

Prilozi

Prilozi na USB-u:

- Prilog 1: programski kod aplikacije
- Prilog 2: knezevic_dino_diplomski.pdf