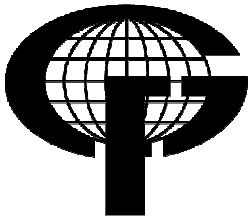


Sveučilište u Zagrebu – Geodetski fakultet
University of Zagreb – Faculty of Geodesy

Katedra za geoinformatiku
Chair of Geoinformation Science

Kačićeva 26, 10000 Zagreb, Croatia
Web: http://geoinfo.geof.hr; Tel.: +385 (1) 46 39 227; Fax: +385 (1) 48 26 953

Diplomski rad

Vizualizacija geoprostornih podataka OpenGL modulom

Izradio: Ranko Marić

Mentor: prof. dr. sc. Damir Medak

Zagreb, srpanj 2011.

Prije svega zahvaljujem se svom mentoru prof. dr. sc. Damiru Medaku na primjedbama, prijedlozima i savjetima, te asistentu dipl. ing. Mariu Mileru.

Najveću zahvalu dugujem svojoj obitelji i Nikolini, koji su me podržavali i bili uz mene tijekom cijelog studija.

I. Autor

Ime i Prezime: Ranko Marić

Datum i mjesto rođenja: 18. prosinca 1982., Zabok, Hrvatska

II. Diplomski rad

Predmet: Analiza prostornih podataka

Naslov: Vizualizacija geoprostornih podataka OpenGL modulom

Mentor: Prof. dr. sc. Damir Medak

Voditelj: Mario Miler, dipl. ing. geod.

III. Ocjena i obrana

Datum zadavanja zadatka: 20. travnja 2011.

Datum obrane: 08. srpnja 2011.

Sastav povjerenstva pred kojim je branjen diplomski rad:

Prof. dr. sc. Damir Medak

Prof. dr. sc. Drago Špoljarić

Dr. sc. Ivan Medved

Vizualizacija geoprostornih podataka OpenGL modulom

Sažetak: *Ovaj rad opisuje princip rada OpenGL-a i postupak izrade preglednika 3D Wavefront OBJ datoteka. Za funkcioniranje preglednika i prikaz modela koristi se OpenGL API grafika. Ovaj projekt je započet s ciljem učenja C++ programskog jezika i OpenGL grafike. Kao rezultat, razvijena je aplikacija kojom se mogu pregledavati 3D objekti sa vezanom teksturom, učitani iz OBJ datoteke. Spomenuta aplikacija može se koristiti za vizualizaciju prostornih objekata kao samostalni program.*

Ključne riječi: *OpenGL, API, Wavefront OBJ, Visual Studio, C++.*

Visualization of geospatial data with OpenGL module

Abstract: *This paper describes the working principle of OpenGL and the process of making 3D Wavefront OBJ files viewer. For functioning of the viewer and model display it is used OpenGL API graphics. This project was initiated in order to learn C++ programming language and OpenGL graphics. As a result, it is developed an application that can be used to view 3D objects with bound textures, loaded from OBJ files. Mentioned application can be used to visualize spatial objects as a standalone program.*

Keywords: *OpenGL, API, Wavefront OBJ, Visual Studio, C++.*

Sadržaj

1. Uvod

Današnjim metodama mjerenja prikupljaju se velike količine podataka. Ti podaci su teško čitljivi bez grafičkog prikaza, bilo dvodimenzionalnog ili trodimenzionalnog. Grafički prikaz izmjerenih podataka olakšava rad nad tim podacima, omogućuje lakše pronalaženje i ispravljanje pogrešaka, potrebno je kraće radno vrijeme, te pruža nevjerojatne vizualizacije. U svemu tome veliko značenje ima *OpenGL*. Implementiran je kroz sve aspekte poslova, kada je riječ o računalnoj grafici, i današnji način života bio bi nezamisliv bez njega.

OpenGL grafički sustav je softversko sučelje grafičkog hardvera. On omogućuje stvaranje interaktivnih programa koji proizvode slike u boji ili trodimenzionalne objekte u pokretu. Sa *OpenGL*-om je moguće kontrolirati računalnu grafiku za proizvodnju realnih slika, ili onih koje odstupaju od stvarnosti na maštovite načine. Ovaj rad objašnjava najvažnije dijelove *OpenGL* grafike za dobivanje željenih vizualnih efekata i prikaza teksturiranih 3D modela.

OpenGL je 2D i 3D grafički *API*, koji je najviše zastupljen u industriji, donoseći aplikacije na široku paletu računalnih platformi. To je sustav baziran na prozorima, neovisan o operacijskom sustavu. *OpenGL* omogućava programerima softvera za PC, radne stanice i super hardver stvaranje vizualno privlačnih grafičkih aplikacija, visokih performansi, kao što su *CAD*, kreiranje sadržaja, energija, zabava, razvoj igara, proizvodnja, medicina i virtualna stvarnost. *OpenGL* razotkriva sve mogućnosti najnovijeg grafičkog hardvera.

2. Korištene tehnologije

Prilikom izrade ovog projekta korištene su različite tehnologije. Tako je aplikacija pisana u programskom jeziku *C++*, koji je sastavni dio integriranog razvojnog okružja *Microsoft Visual Studio*-a. Kostur cijele aplikacije je *OpenGL API*, koji pruža trodimenzionalni prikaz modela. Model je spremljen u nekomercijalnoj datoteci *Wavefront OBJ*. Korištene tehnologije opisane su u nekim od sljedećih podpoglavlja.

2.1. OpenGL

OpenGL (*engl. Open Graphics Library*) je *API*[1] grafika koja se koristi za dobivanje visoko kvalitetne 2D i 3D računalne grafike i animacije. Podržava više programskih jezika i rad na različitim platformama. *OpenGL* je izvorno kreiran od strane *Silicon Graphics Inc*. (*SGI*), u nastojanju da se pojednostavi kod potreban za pisanje 3D aplikacija, kako bi se premostio jaz između softvera i hardvera. Sučelje se sastoji od više od 250 različitih funkcija koje se mogu pozvati i koristiti za crtanje složenih trodimenzionalnih prizora iz osnovnih geometrijskih oblika (URL 1). Naširoko se koristi u *CAD* (*engl. Computer-Aided Design*) sustavima, *GIS* (*engl. Geographic Information System*) sustavima, za stvaranje virtualne stvarnosti, znanstvene vizualizacije, simulacija leta, itd. Također se koristi u izradi video igara, gdje se natječe s *Direct3D*-om na Microsoft Windows platformama.

OpenGL koristi specijalizirane mogućnosti novih 3D video kartica za vrlo brzo renderiranje 3D grafike. To se često naziva *3D hardversko ubrzanje* (URL 2). Bez hardverskih ubrzivača mnoge mogućnosti *OpenGL*-a ne bi radile, ili bi radile vrlo sporo. *OpenGL* u kombinaciji sa grafičkom karticom će znatno ubrzati grafiku i omogućiti višu razinu vizualne kvalitete. Većina računala već sadrži *OpenGL driver*-e koji će omogućiti pokretanje programa koji to zahtijevaju. Ali, ako se

---

[1] **API** (*engl. Application Programming Interface*) - određeni skup pravila i specifikacija koje softverski program može pratiti kako bi mogao pristupiti i koristiti usluge i resurse koje pruža drugi poseban program koji implementira taj *API*.

određeni program ne može pokrenuti, potrebno je instalirati *driver*-e za grafičku karticu koja se koristi.

U 1980-im razvoj softvera, koji bi mogao funkcionirati sa širokim rasponom grafičkog hardvera, bio je pravi izazov. Softverski programeri pisali su prilagođena sučelja i *driver*-e za svaki komad hardvera. To je bilo skupo i rezultiralo je sa puno dupliciranja rada. Do ranih 90-ih, *SGI* je lider za 3D grafiku za radne stanice. Njihov *IrisGL*[2] *API* bio je vrijedan divljenja te je postao industrijski standard i tako zasjenio otvoreni standard *PHIGS*[3]. To je zato što se *IrisGL* smatrao lakšim za korištenje, i zato što podržava neposredan način renderiranja. Nasuprot tome, *PHIGS* se smatra teškim za korištenje i zastarjelim u smislu funkcionalnosti. *SGI* konkurenti (uključujući i *Sun Microsystems*, *Hewlett-Packard* i *IBM*) su također donijeli na tržište 3D hardver, podržan *PHIGS* standardom. Kako se pojavljivalo sve više proizvođača 3D grafičkog hardvera, to je uzrokovalo slabljenje *SGI*-a na tržištu. U nastojanju da utječe na tržište, *SGI* je odlučio pretvoriti *IrisGL API* u otvoreni standard. Osim toga, *SGI* je imao veliki broj korisnika softvera, a promjenom na *OpenGL API* planirali su svoje klijente zadržati na *SGI* hardveru na nekoliko godina, dok se tržište za *OpenGL* ne razvije. Sa tako mnogo različitih vrsta grafičkog hardvera, mogućnost da svi govore istim jezikom dovela bi programere na viši nivo razvijanja 3D softvera. 1992. godine *SGI* je vodio stvaranje *OpenGL* arhitektonskog odbora (*OpenGL ARB*), grupe tvrtki koje će održavati i proširiti *OpenGL* specifikaciju za godine koje dolaze.

OpenGL je evoluirao iz *IrisGL*-a. Jedno od ograničenja *IrisGL*-a je pristup njegovim mogućnostima pod uvjetom da te mogućnosti podržava hardver. Ako grafički hardver ne podržava određene mogućnosti, one se ne mogu koristiti. *OpenGL* je prevladao ovaj problem u pružanju softverske potpore za mogućnosti koje nisu

---

[2] **IrisGL** (*engl. Integrated Raster Imaging System Graphics Library*) - grafički *API* kreiran od strane *SGI*-a za proizvodnju 2D i 3D računalne grafike.

[3] **PHIGS** (*engl. Programmer's Hierarchical Interactive Graphics System*) – *API* standard za 3D računalnu grafiku koji se više ne koristi.

podržane od strane hardvera, omogućujući aplikacijama korištenje napredne grafike na sustavima relativno male snage.

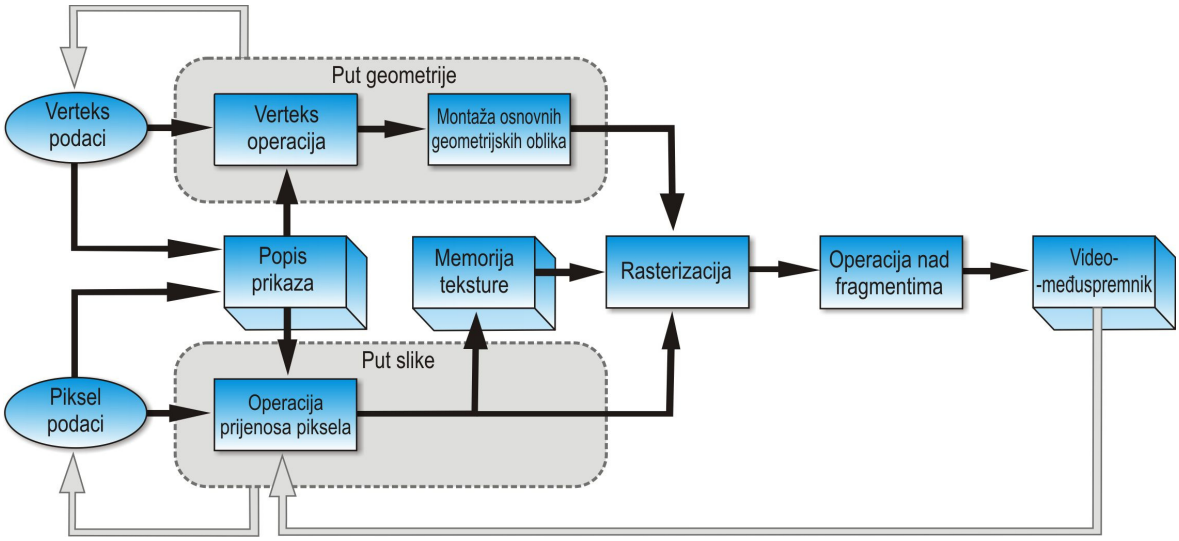
1995. godine *Microsoft* je objavio *Direct3D*, koji će postati glavni konkurent *OpenGL*-u. 1997. *Microsoft* i *SGI* pokrenuli su *Fahrenheit* projekt, da zajedničkim naporom objedine *OpenGL* i *Direct3D* sučelja. Projekt je u početku pokazivao neka obećanja da unese red u svijet interaktivne 3D računalne grafike, ali na račun financijskih ograničenja u *SGI*-u, i zbog strateških razloga *Microsoft*-a, te opći nedostatak potpore industrije, projekt je napušten 1999. godine. Zadnja verzija *OpenGL*-a je izdana 2010. godine i nosi naziv *OpenGL 4.1*.

2.1.1. OpenGL ″pipeline″ arhitektura

Mnoge *OpenGL* funkcije koriste se posebno za crtanje objekata kao što su točke, linije, poligoni i bitmape. Neke funkcije upravljaju načinom na koji će crtež biti prikazan, kao što su funkcije za antialiasing ili prikaz tekstura. Ostale funkcije se specifično odnose na manipulaciju videomeđuspremnika. Sve te funkcije rade zajedno kako bi stvorile *OpenGL* protočnu (*engl. pipeline*) arhitekturu.

OpenGL ne uključuje funkcije za upravljanje radnim prozorima, interakciju s korisnikom, ili *I/O* za datoteke. Svako radno okruženje (kao što je *Microsoft Windows*) ima vlastite funkcije za ovu svrhu, koje služe za provedbu nekih sredstava predaje kontrole prozora *OpenGL* crteža.

OpenGL aplikacije prolaze kroz dvije vrste podataka za iscrtavanje: podaci vrhova (verteksa) i piksela (URL 3). Podaci vrhova koriste se za prikaz 2D i 3D geometrijskih osnovnih oblika. Aplikacija definira podatke piksela kao nizove piksela i može usmjeriti *OpenGL* za prikaz piksela izravno iz videomeđuspremnika, ili ih kopirati u memoriju teksture za kasniju uporabu kao teksturirane mape.



Slika 1. OpenGL protočna arhitektura

Verteks operacija. *OpenGL* obavlja sljedeće operacije nad svakim vrhom kojeg prima iz aplikacije:

- *Transformacija*. *OpenGL* transformira svaki vrh iz koordinatnog sustava objekta u koordinatni sustav radnog prozora. To je ogromno pojednostavljenje procesa transformacije.
- *Rasvjeta*. Ako je aplikacijom omogućena rasvjeta, *OpenGL* izračunava vrijednost osvjetljenja za svaki vrh.
- *Izrezivanje*. Kada *OpenGL* utvrdi da cijeli osnovni geometrijski oblik nije vidljiv, jer je izvan područja prikaza, *OpenGL* odbacuje sve točke. Ako je osnovni oblik djelomično vidljiv, *OpenGL* ga isječke, tako da se samo vidljivi dio rasterizira.

Operacija prijenosa piksela. Dok geometrijski podaci prolaze kroz jedan put *OpenGL* renderiranja, piksel podaci prolaze drugačiji put. Nakon što se pikseli iz klijentove memorije otpakiraju, nad podacima se izvodi skaliranje, mapiranje i stezanje (*engl. biasing*). Ove operacije se nazivaju operacije prijenosa piksela. Preneseni podaci ili se pohranjuju u memoriju teksture ili se izravno rasteriziraju na fragmente. *OpenGL* obavlja operacije skladištenja piksela na svim blokovima podataka piksela koje aplikacija šalje i prima od *OpenGL*-a. Ove operacije kontroliraju zamjenu bajtova, punjenje i pomake unutar blokova podataka piksela kao podršku za slanje i primanje piksela u raznim formatima.

Popis prikaza (*engl. disply list*). Popis prikaza je skupina *OpenGL* naredbi koje su pohranjene za kasnije izvršavanje. Svi podaci, geometrija (verteks) i piksel podaci, mogu biti pohranjeni u popis prikaza. To može poboljšati performanse budući da su naredbe i podaci spremljeni u listu. Kada *OpenGL* program radi na mreži, može se smanjiti prijenos podataka preko mreže koristeći popis prikaza. Budući da je popis dio stanja poslužitelja (servera) i boravi na stroju poslužitelja, klijent stroj treba poslati naredbe i podatke samo jednom na poslužitelja, na kojem se nalazi popisa prikaza.

Montaža osnovnih geometrijskih oblika. Nakon verteks operacije, osnovni geometrijski oblici (točka, linija i poligon) ponovno se transformiraju pomoću projekcijske matrice i izrezuju se na veličinu radnog prozora. Nakon toga primjenjuje se perspektivna projekcija kako bi se 3D scena prikazala u sustavu koordinata prozora. Posljednja stvar koju treba učiniti je test uništavanja (*engl. culling test*), ako je omogućen. Time se uništavaju svi geometrijski oblici koji su zaklonjeni od strane objekata u prvom planu, ili ako je lice poligona okrenuto od kamere.

Memorija teksture. Slika teksture učita se u memoriju teksture, a zatim se primjenjuje na geometrijske objekte. *OpenGL* aplikacija može primijeniti teksturu na geometrijske objekte kako bi izgledali mnogo realističnije.

Rasterizacija. Rasterizacija je pretvorba geometrijskih podataka u fragmente. Fragmenti su pravokutni nizovi koji sadrže položaj, boju, dubinu, širinu linije, veličinu točke, antialiasing proračune i druge podatke koje *OpenGL* obrađuje prije spremanja u videomeđuspremnik. Svaki fragment odgovara jednom pikselu u videomemoriji.

Operacija nad fragmentima. Operacija nad fragmentima je posljednja radnja koja pretvara fragmenate u piksele, u videomeđuspremniku. *OpenGL* vrši obradu nad svakim fragmentom kako bi se utvrdile njegove konačne vrijednosti u videomemoriji. Za svaki fragment proizveden rasterizacijom, *OpenGL* obavlja sljedeće poslove, koji mogu biti omogućeni ili onemogućeni:

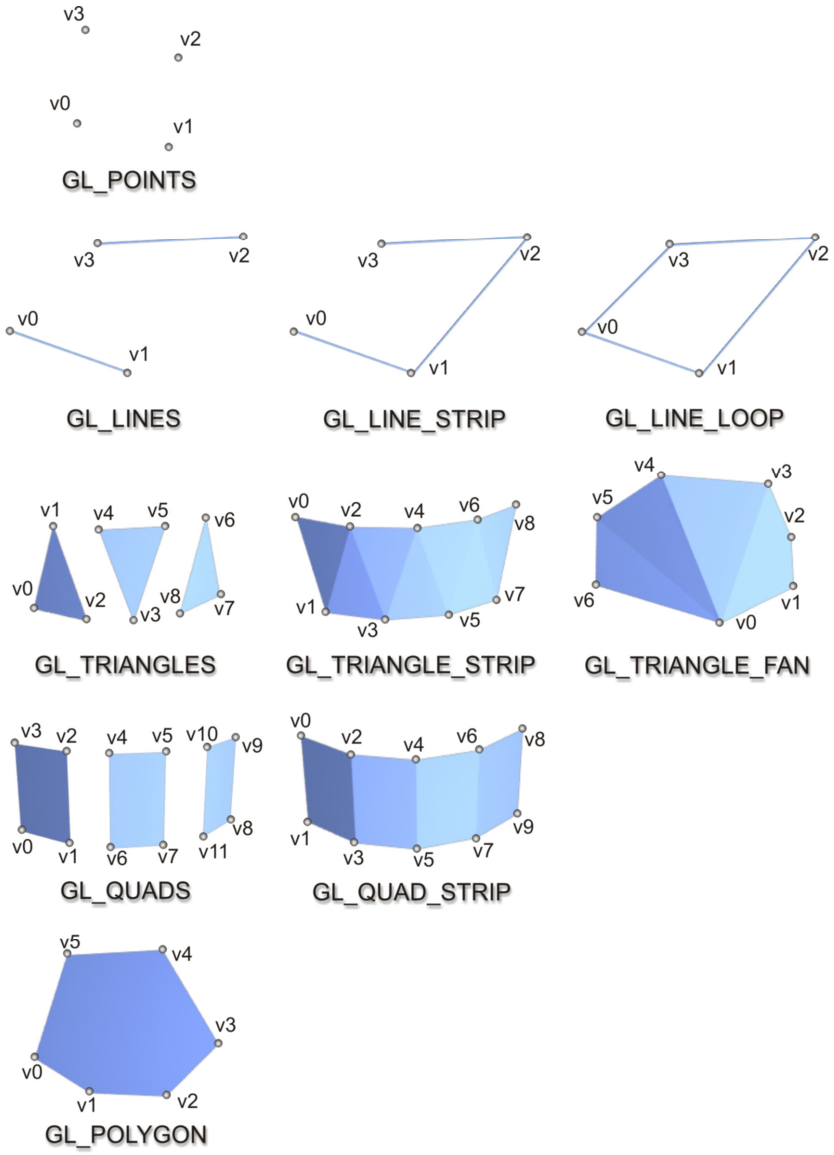
- *Testiranje vlasništva piksela*. Ovo testiranje omogućuje *OpenGL*-u ispravan prikaz u nazočnosti više preklapajućih prozora. Fragmenti koji su zaklonjeni

preklapanjem prozora mogu se odbaciti ili pohraniti kao podrška (*engl. backup*).

- *Scissor test*. Ako fragment leži izvan koordinatnog sustava pravokutnika definiranog unutar prozora aplikacije, *OpenGL* ga odbacuje.
- *Alpha test*. Ako alfa vrijednost fragmenta ne prođe navedene kriterije aplikacije, *OpenGL* ga odbacuje. Ovo je korisno za potpuno ili djelomično odlaganje transparentnih fragmenata.
- *Stencil test*. *OpenGL* uspoređuje spremljene vrijednosti matrica u videomemoriji i koristi trenutno stanje aplikacije kako bi se utvrdilo što učiniti s fragmentom i kako modificirati spremljenu vrijednost matrice. Ovaj test se koristi kod izrezivanja oblika koji nisu pravokutni, sjena i konstruktivne stereometrije (*engl. Constructive Solid Geometry - CSG*).
- *Depth test*. Ispitivanjem dubine odbacuje se fragment ako usporedba između vrijednosti dubine fragmenta i vrijednosti pohranjene u dubinskom međuspremniku ne uspije. *OpenGL* dubinski test je oblik koji se naziva *z-buffering*.
- *Occlusion query*. Kada je ovaj upit aktivan, *OpenGL* omogućava brzo testiranje vidljivosti u složenim scenama.
- *Blending*. Miješanje kombinira RGB boju i alfa vrijednosti fragmenta sa RGB i alfa vrijednostima spremljenim u videomeđuspremniku. Aplikacije obično koristite miješanje da simuliraju prozirnost objekata.
- *Dithering*. Kada videomeđuspremnik ima manju preciznost za spremanje boja od RGB i alfa vrijednosti fragmenta, *OpenGL* podrhtava boju i alfa vrijednost fragmenta, koristeći ponavljajući algoritam. Ova značajka se rijetko koristi.
- *Logičke operacije*. *OpenGL* piše konačnu vrijednost boje u videomemoriju prema navedenim logičkim operacijama aplikacije.

2.1.2. Crtanje osnovnih geometrijskih oblika

OpenGL se često naziva *API* niske razine zbog minimalne podrške za geometrijske oblike višeg reda, strukture podataka kao što su grafikoni scena, ili podrške za učitavanje 2D slikovnih datoteka ili datoteka 3D modela. Umjesto toga, *OpenGL* se fokusira na učinkovito renderiranje osnovnih geometrijskih oblika niske razine sa različitim osnovnim, ali fleksibilnim, postavkama za renderiranje. *OpenGL* aplikacije renderiraju osnovne geometrijske oblike tako da odrede tip osnovnog oblika i redoslijed vrhova s pripadajućim podacima. Tip osnovnog geometrijskog oblika određuje kako *OpenGL* tumači i renderira niz vrhova. *OpenGL* pruža deset različitih osnovnih tipova za crtanje točke, linije i poligona.



Slika 2. OpenGL tipovi osnovnih geometrijskih oblika

OpenGL tumači vrhove i renderira svaki osnovni geometrijski oblik korištenjem sljedećih pravila:

- *GL_POINTS*. Renderiranje matematičke točke. *OpenGL* renderira točku za svaki navedeni vrh.
- *GL_LINES*. Crtanje nepovezanih linijskih segmenata. *OpenGL* crta jednu liniju za svaku grupu od dva vrha. Ako aplikacija specificira *n* vrhova, *OpenGL* crta *n/2* linijska segmenta. Ako je *n* neparan, ignorira se završni vrh.
- *GL_LINE_STRIP*. Renderiranje niza povezanih linijskih segmenata. *OpenGL* renderira segment jedne linije između prvog i drugog vrha, između drugog i trećeg, između trećeg i četvrtog, i tako dalje. Ako aplikacija specificira *n* vrhova, *OpenGL* crta *n-1* linijskih segmenata.
- *GL_LINE_LOOP*. Crtanje zatvorene linijske trake. *OpenGL* renderira ovaj osnovni oblik kao i *GL_LINE_STRIP* s dodatkom zatvaranja linijskog segmenta između završnog i prvog vrha.
- *GL_TRIANGLES*. Crtanje pojedinačnih trokuta. *OpenGL* renderira trokut za svaku grupu od tri vrha. Ako aplikacija specificira *n* vrhova, *OpenGL* crta *n/3* trokuta. Ako je *n* veći od 3, preostali vrhovi se ignoriraju.
- *GL_TRIANGLE_STRIP*. Crtanje niza trokuta koji imaju zajedničke rubove. *OpenGL* renderira prvi trokut koristeći prvi, drugi i treći vrh, a zatim renderira drugi trokut koristeći drugi, treći i četvrti vrh, i tako dalje. Ako aplikacija specificira *n* vrhova, OpenGL crta *n-2* povezana trokuta. Ako je *n* manji od 3, *OpenGL* ne crta ništa.
- *GL_TRIANGLE_FAN*. Crtanje lepeze trokuta koji imaju zajedničke rubove i jedan vrh. Svakom trokutu je zajednički prvi navedeni vrh. Ako aplikacija specificira redoslijed vrhova *v*, *OpenGL* renderira prvi trokut pomoću *v0*, *v1* i *v2*, drugi trokut koristeći *v0*, *v2* i *v3*, treći trokut koristeći *v0*, *v3* i *v4*, i tako dalje (slika 2). Ako aplikacija specificira *n* vrhova, *OpenGL* renderira *n-2* povezanih trokuta. Ako je *n* manji od 3, *OpenGL* ne crta ništa.
- *GL_QUADS*. Crtanje pojedinog konveksnog četverokuta. *OpenGL* renderira četverokut za svaku grupu od četiri točke. Ako aplikacija specificira *n*

vrhova, *OpenGL* renderira *n/4* četverokuta. Ako je *n* veći od 4, *OpenGL* ignorira višak vrhova.

- *GL_QUAD_STRIP*. Crtanje povezanih četverokuta koji dijele rubove. Ako aplikacija specificira redoslijed vrhova *v*, *OpenGL* renderira prvi četverokut koristeći *v0*, *v1*, *v3* i *v2*, drugi četverokut koristeći *v2*, *v3*, *v5* i *v4*, i tako dalje. Ako aplikacija specificira *n* vrhova, *OpenGL* crta *(n-2)/2* četverokuta. Ako je *n* manji od 4, OpenGL ne crta ništa.
- *GL_POLYGON*. Crtanje jednog ispunjenog konveksnog poligona. *OpenGL* renderira poligon sa *n* vrhova definiranih aplikacijom. Ako je *n* manji od 3, OpenGL ne crta ništa.

Za korištenje *GL_QUADS*, *GL_QUAD_STRIP* i *GL_POLYGON*, svi osnovni geometrijski oblici moraju biti ravninski i konveksni. Inače, ponašanje *OpenGL*-a je nedefinirano. Tipovi crtanja *GL_LINE_STRIP*, *GL_LINE_LOOP*, *GL_TRIANGLE_STRIP*, *GL_TRIANGLE_FAN* i *GL_QUAD_STRIP* dijele vrhove između svojih linijskih segmenata. Ovi tipovi osnovnih geometrijskih oblika trebali bi se koristiti kada je to moguće i praktično za smanjenje proračuna za svaki vrh.

2.1.2.1 Crtanje osnovnih geometrijskih oblika korištenjem funkcija *glBegin()* i *glEnd()*

OpenGL aplikacije renderiraju osnovne geometrijske oblike po okolnim vrhovima sa parom funkcija *glBegin()* i *glEnd()*. Aplikacija određuje tip geometrijskog oblika tako da ga propušta kao parametar u funkciju *glBegin()*.

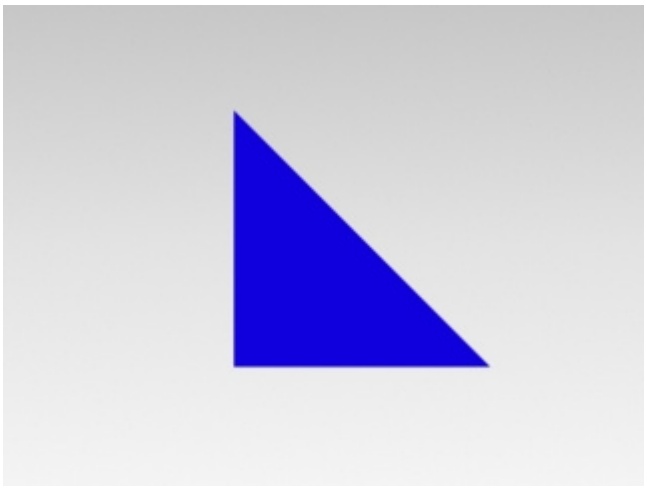
```
void glBegin( GLenum mode );
...
void glEnd( void );
```

Funkcije *glBegin()* i *glEnd()* su naredbe koje specificiraju vrhove i podatke vrhova. *mode* određuje tip osnovnog geometrijskog oblika koji će se crtati. Između *glBegin()* i *glEnd()*, aplikacija određuje vrhove i stanja vrhova kao što su trenutna primarna boja, tekuća normala te svojstva materijala za rasvjetu, te trenutne koordinate teksture za mapiranje tekstura.

glColor3f() naredba postavlja trenutno stanje primarne boje koristeći RGB vrijednosti. Kako *OpenGL* dobiva naredbe *glVertex3f()*, tako dodjeljuje trenutno

stanje primarne boje svakom vrhu. *OpenGL* ne ograničava broj vrhova koji mogu biti definirani aplikacijom, između *glBegin()* i *glEnd()* funkcija.

```
// Primjer OpenGL koda za crtanje plavog trokuta
glBegin( GL_TRIANGLES );
  glColor3f  ( 0.0f, 0.0f, 1.0f ); // Postavlja tekuću boju u plavu
  glVertex3f ( 0.0f, 0.0f, 0.0f ); // Definira tri vrha
  glVertex3f ( 1.0f, 0.0f, 0.0f );
  glVertex3f ( 0.0f, 1.0f, 0.0f );
glEnd();
```



Slika 3. Plavi OpenGL trokut definiran sa tri vrha.

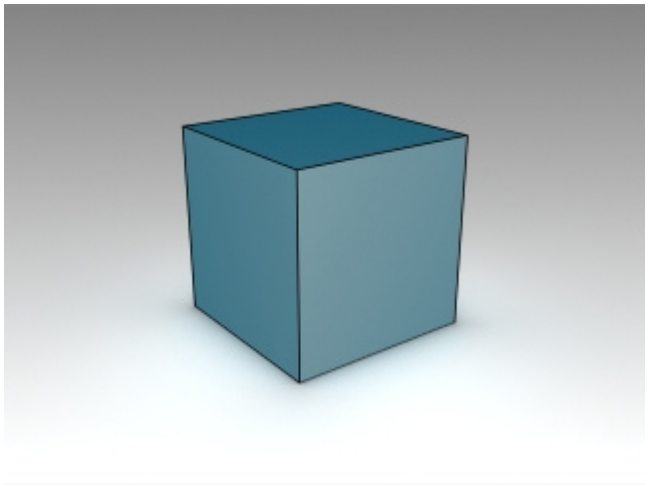
Pojedinačne funkcije za definiranje vrhova i stanja vrhova pružaju veliku fleksibilnost prilikom razvijanja aplikacija. Funkcije *glBegin()* i *glEnd()* upravo to dokazuju, međutim, dramatično ograničavaju performanse aplikacije. Iako popis prikaza dopušta implementaciju za optimiziranje podataka poslanih putem ovih funkcija, *OpenGL* distributeri očekuju da programeri koriste mehanizme koji su učinkovitiji i lakši za optimizaciju, kao što je pohranjivanje objekata u međuspremnik i polje vrhova (*engl. vertex array*). Iz tog razloga, najbolje je izbjegavati korištenje *glBegin()* i *glEnd()* funkcija.

2.1.2.2 Crtanje osnovnih geometrijskih oblika korištenjem polja vrhova

Može se primijetiti da *OpenGL* zahtijeva mnogo funkcija, koje dolaze između *glBegin()* i *glBegin()* funkcija, za prikaz složenijih objekata koji se sastoje od osnovnih geometrijskih oblika. Crtanje poligona sa 30 vrhova zahtijeva barem 32 funkcije: jedan poziv na *glBegin()* za početak crtanja, po jedan poziv na *glVertex3f()* za svaki vrh, i završni poziv na *glEnd()*. Dodatne informacije, kao što

su npr. normale površina, zahtijevaju poziv dodatnih funkcija za svaki vrh. Tako se vrlo brzo može udvostručiti ili utrostručiti broj pozvanih funkcija potrebnih za crtanje jednog geometrijskog objekta. Velik broj pozvanih funkcija može ometati rad složenijih aplikacija.

Dodatni problem koji predstavlja *glBegin()/glEnd()* metoda je suvišna obrada vrhova koji su zajednički između susjednih poligona. Na primjer, kocka na slici (slika 4) se sastoji od šest lica i osam zajedničkih točaka. Ako se koristi standardna metoda za crtanje ovog objekta, svaki vrh mora biti navedena tri puta i to jednom za svako lice koje ga koristi. Dakle, obrađeno je 24 vrhova, a bilo je dovoljno samo osam.



Slika 4. Kocka (šest lica sa osam zajedničkih točaka).

OpenGL ima rutine polja vrhova koje omogućuju određivanje mnogo povezanih podataka vrhova sa samo nekoliko polja, i pristup tim podacima s jednako malo funkcija. Korištenjem rutina polja vrhova, poligon sa 30 vrhova može se staviti u jedan niz i pozvati uz pomoć jedne funkcije. Ako svaki vrh ima definiranu normalu, svih 30 normala može se staviti u još jedan niz i također pozvati sa samo jednom funkcijom. Korištenjem polja vrhova smanjuje se broj pozvanih funkcija, što na kraju poboljšava performanse. Također, koristeći polja vrhova omogućuje se ponovna uporaba već obrađenih zajedničkih točaka

Polja vrhova omogućuju aplikacijama renderiranje osnovnih geometrijskih oblika iz podataka vrhova pohranjenih u blokovima memorije. Pod nekim okolnostima, *OpenGL* implementacija može obraditi vrhove i spremiti rezultat u predmemoriju

(*engl. cache*) za učinkovitu ponovnu upotrebu. Aplikacija određuje osnovne geometrijske oblike tako da ih indeksira kao podatke polja vrhova.

Postoje tri koraka za korištenje polja vrhova za prikaz geometrije:

Korak 1. Aktiviranje odgovarajućih nizova, i sa svakim nizom, pohranjivanje različitog tipa podataka: koordinata vrhova, normala površina, RGBA boja, sekundarnih boja, indeksa boja, koordinata magle, koordinata tekstura, ili rubova poligona. U ovom koraku poziva se funkcija *glEnableClientState()* uz navedeni parametar, koji aktivira izabrani niz.

```
void glEnableClientState ( GLenum array );
...
void glDisableClientState ( GLenum array );
```

Prihvatljivi parametri za aktiviranje određenog niza su: *GL_VERTEX_ARRAY*, *GL_COLOR_ARRAY*, *GL_SECONDARY_COLOR_ARRAY*, *GL_INDEX_ARRAY*, *GL_NORMAL_ARRAY*, *GL_EDGE_FLAG_ARRAY*, *GL_FOG_COORD_ARRAY* i *GL_TEXTURE_COORD_ARRAY*.

Korak 2. Stavljanje podataka u niz ili nizove. Nizovima se pristupa pomoću adrese koja upućuje na njihova mjesta u memoriji. Jednom naredbom definira se jedan niz. Postoji osam različitih rutina za definiranje nizova, po jedna funkcija za svaki niz.

```
void glVertexPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);
```

Gore navedena funkcija određuje gdje se može pristupiti podacima prostornih koordinata. *pointer* je adresa u memoriji prve koordinate prvoga vrha u nizu. *type* određuje tip podataka (*GL_SHORT*, *GL_INT*, *GL_FLOAT*, or *GL_DOUBLE*) svake koordinate u nizu. *size* je broj koordinata po svakom vrhu, a iznosi 2, 3 ili 4. *stride* je atribut za *gl*Pointer()* rutine koji govori *OpenGL*-u kako da pristupi podacima koji su mu pruženi pomoću niza. Njegova vrijednost trebala bi biti broj bajtova između dva *pointer* elementa u slijedu, ili nula, što je poseban slučaj.

Osam funkcija za pristup osam tipova nizova: *glVertexPointer()*, *glColorPointer()*, *glSecondaryColorPointer()*, *glNormalPointer()*, *glFogCoordPointer()*, *glIndexPointer()*, *glTexCoordPointer()*, *glEdgeFlagPointer()*.

```
// Primjer uspostavljanja i učitavanja polja vrhova
static GLint vertices[] = { 25, 25,
                            100, 325,
                            175, 25,
                            175, 325,
                            250, 25,
                            325, 325 };
glEnableClientState( GL_VERTEX_ARRAY );
glVertexPointer( 2, GL_INT, 0, vertices );
```

Korak 3. Crtanje geometrije s podacima spremljenim u nizove. *OpenGL* dobiva podatke iz svih aktivnih nizova tako da referencira naputke. Moguće je dobiti podatke iz jednog elementa polja (indeksirana lokacija), iz naloženog popisa polja elemenata (koji mogu biti ograničeni na podskup podataka polja vrhova), ili od niza elemenata polja.

```
void glDrawElements(GLenum mode, GLsizei count, GLenum type, const GLvoid
*indices);
```

Gore navedena funkcija definira niz osnovnih geometrijskih oblika pomoću određenog broja elemenata, čiji su indeksi pohranjeni u polju indeksa. *type* mora biti *GL_UNSIGNED_BYTE*, *GL_UNSIGNED_SHORT*, ili *GL_UNSIGNED_INT*, označavajući vrstu podataka polja indeksa. *mode* određuje kakvi će se osnovni geometrijski oblici konstruirati i prihvaća iste vrijednosti kao i funkcija *glBegin()* (*GL_POLYGON*, *GL_LINE_LOOP*, *GL_LINES, GL_POINTS*, i tako dalje).

```
void glMultiDrawElements(GLenum mode, GLsizei *count, GLenum type, const
GLvoid *indices, GLsizei primcount);
```

Funkcija poziva slijed *glDrawElements()* naredbi. *indices* je niz pokazivača na popis niza elemenata. *count* je niz koji ukazuje na to koliko je vrhova pronađeno u svakom pojedinačnom popisu niza elemenata. *mode* i *type* su isti kao i za funkciju *glDrawElements()*.

```
void glDrawRangeElements(GLenum mode, GLuint start, GLuint end, GLsizei
count, GLenum type, const GLvoid *indices);
```

Gore navedena funkcija stvara niz osnovnih geometrijskih oblika, slično kao i slijed kreiran sa funkcijom *glDrawElements()*. Nekoliko parametara *glDrawRangeElements()* naredbe isti su kao i u naredbi *glDrawElements()*, uključujući i *mode* (vrsta osnovnog geometrijskog oblika), *count* (broj elemenata), *type* (vrsta podataka) i *indices* (niz lokacija podataka vrhova). *glDrawRangeElements()* funkcija uvodi i dva nova parametra: *start* i *end*, koji

određuju raspon prihvatljivih vrijednosti za parametar *indices*. Da bi bile važeće, vrijednosti niza *indices* moraju biti između vrijednosti parametara *start* i *end*.

2.1.2.3 Objekti spremnika (*engl. buffer objects*)

Postoji mnogo operacija u *OpenGL*-u gdje se šalju golemi blokovi podataka u *OpenGL*, kao što je propuštanje podataka polja vrhova na obradu. Prijenos tih podataka može biti jednostavan, kao što je kopiranje iz sistemske memorije na grafičku karticu. Ali, zato što je *OpenGL* dizajniran kao klijent-server model, svaki puta kada *OpenGL* treba podatke, oni se moraju prenijeti iz klijent memorije. Ako se ti podaci ne mijenjaju, ili ako su klijent i server različita računala, prijenos tih podataka može biti spor ili suvišan. Objekti spremnika su dodani *OpenGL* verziji 1.5 kako bi se omogućilo aplikacijama da odrede koji će podaci biti spremljeni na grafički server.

2.1.2.4 Objekti polja vrhova (*engl. vertex-array objects*)

Kako aplikacija postaje sve veća i složenija, i koristi sve više modela, prilikom svakog iscrtavanja prozora izmjenjuju se setovi polja vrhova. U ovisnosti koliko se atributa vrhova koristi za svaki vrh, broj poziva funkcija (kao što je *glVertexPointer()*) može postati prevelik. Objekti polja vrhova su setovi poziva za postavke stanja polja vrhova. Nakon inicijalizacije, mogu se brzo izmjenjivati različiti setovi polja vrhova sa samo jednim pozivom. Za kreiranje objekata polja vrhova, prvo se zove funkcija *glGenVertexArrays()*, koja kreira željeni broj neinicijaliziranih objekata.

```
void glGenVertexArrays(GLsizei n, GLuint *arrays);
```

Funkcija vraća *n* trenutno neiskorištenih imena za korištenje objekata polja vrhova u nizu *arrays*.

Nakon kreiranja objekata polja vrhova, potrebno je inicijalizirati novonastale objekte, i povezati set podataka polja vrhova s pojedinačnim dodijeljenim objektima. To omogućuje *glBindVertexArray()* rutina. Nakon inicijalizacije svih objekata polja vrhova, navedena funkcija se koristi za prijelaz između različitih setova polja vrhova.

```
GLvoid gBindVertexArray(GLuint array);
```

Naredba *glBindVertexArray()* radi tri stvari. Kada se za *array* koristi vrijednost koja je različita od nule, i ako je ta vrijednost vraćena iz funkcije *glGenVertexArrays()*, tada je kreiran novi objekt polja vrhova. Kada se veže na prije kreirani objekt polja vrhova, taj objekt postaje aktivan. Kada se veže na *array* vrijednost ili na nulu, *OpenGL* prestaje koristiti objekt polja vrhova i vraća se na zadano stanje za polje vrhova.

Za brisanje objekata polja vrhova poziva se funkcija *glDeleteVertexArrays()*. Ako se koriste objekti spremnika za spremanje podataka, oni nisu obrisani kada je obrisan objekt polja vrhova koji upućuje na njih. Oni i dalje postoje (sve dok ne budu obrisani).

```
void glDeleteVertexArrays(GLsizei n, GLuint *arrays);
```

Gore navedena funkcija briše *n* objekata polja vrhova definiranih u *arrays*, omogućujući njihova imena za kasniju upotrebu kao polje vrhova.

2.1.3. Transformacija i prikaz

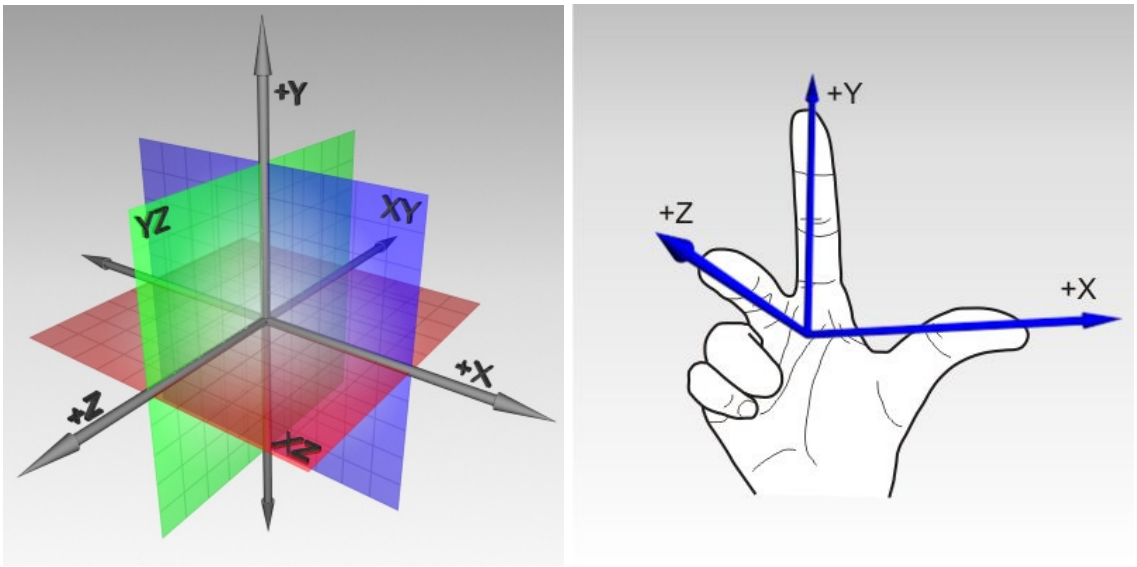
Možda je najčešći problem s kojima se susreću programeri, koji su novi u *OpenGL* programiranju, je prazan ekran (pokretanje *OpenGL* programa proizvodi prazan prozor umjesto renderirane slike). Iako nekoliko problema može uzrokovati ovaj rezultat, najčešći uzrok je pogrešno konfiguriranje *OpenGL* protočne transformacije. Da bi se izbjegao prazan ekran, potrebna je vještina kod kontroliranja stanja *OpenGL* transformacije.

OpenGL pruža moćan, fleksibilan skup rutina za kontrolu transformacije i prikaza. Ovo programsko sučelje nije uvijek jednostavno. Da bi se naveo prikaz lokacije i orijentacije, *OpenGL* zahtijeva primjenu izračuna inverzne matrice transformacije kamere.

Još jedan zajednički izvor konfuzije je razlika između transformacije prikaza, koja pozicionira i usmjerava kameru, i projekcijske transformacije, koja određuje vidno polje, oblik volumena prikaza, te tip projekcije (ortografska ili perspektivna). Te transformacije mogu uzrokovati probleme, ne samo sa prikazom, već i sa drugim *OpenGL* opcijama, kao što su rasvjeta i izrezivanje geometrije.

2.1.3.1 Koordinatni sustav

Grafički sustavi aplikacija definirani su koordinatnim sustavom prema pravilu desne ruke. Kod koordinatnih sustava desne ruke, gledano iz pozitivnog kraja osi, pozitivna rotacija oko osi izvodi se suprotno od kazaljke sata. *OpenGL* ne ograničava aplikacije na desno ili lijevo orijentirane koordinatne sustave, ali mnogi programeri se lakše snalaze ako koriste koordinatni sustav definiran pravilom desne ruke. Ako bi se koristio koordinatni sustav lijeve ruke, transformacijska matrica prikaza mora biti sa negativnim predznakom. Takva matrica se najčešće koristi da bi se dobio efekt zrcala. Kako bi neki objekt dobio zrcalnu sliku dovoljno ga je renderirati dva puta, sa pozitivnom i negativnom matricom transformacije.



Slika 5. Koordinatni sustav definiran pravilom desne ruke.

2.1.3.2 Transformacijske matrice

OpenGL transformira vrhove koristeći matrice dimenzija 4x4. Aplikacije napisane *C/C++* programskim jezikom obično definiraju *OpenGL* matrice kao 1D niz od 16 vrijednosti s pomičnim zarezom (*GLfloat* ili *GLdouble*).

$$\begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

Formula 1. OpenGL transformacijska matrica

OpenGL ima četiri različita tipa matrica; *GL_MODELVIEW*, *GL_PROJECTION*, *GL_TEXTURE* i *GL_COLOR*. Tip matrice može se promijeniti pomoću *glMatrixMode()* funkcije. Na primjer, kako bi se odabrala *GL_MODELVIEW* matrica, koristi se naredba *glMatrixMode(GL_MODELVIEW)*. *OpenGL* matricu treba promatrati kao osnovu za novi koordinatni sustav. Množenjem vektora sa matricom transformira vektor u novi koordinatni sustav definiran matricom.

Matrica model-prikaz (*GL_MODELVIEW*). *GL_MODELVIEW* matrica kombinira matricu prikaza i matricu modela u jednu matricu. Kako bi se transformirao prikaz (kamera), potrebno je premjestiti cijelu scenu s inverznom transformacijom. *gluLookAt()* naredba posebno se koristi za postavljanje transformacije prikaza.

$$\begin{matrix} \text{vektor X osi} & \text{vektor Y osi} & \text{vektor Z osi} & \text{translacija} \\ \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix} \end{matrix}$$

Formula 2. Matrica model-prikaz.

Tri elemenata matrice u krajnjem desnom stupcu [m_{12} m_{13} m_{14}] služe za translaciju transformacije pomoću funkcije *glTranslatef()*. Element m_{15} je homogena koordinata, koja se posebno koristi kod projekcijskih transformacija. Tri elementa skupa [m_0 m_1 m_2], [m_4 m_5 m_6] i [m_8 m_9 m_{10}] su za euklidske i afine transformacije, kao što su rotacija sa funkcijom *glRotatef()* ili skaliranje sa funkcijom *glScalef()*. Potrebno je imati na umu da ta tri vektora zapravo predstavljaju tri ortogonalne osi, tj. pozitivne smjerove X, Y i Z osi.

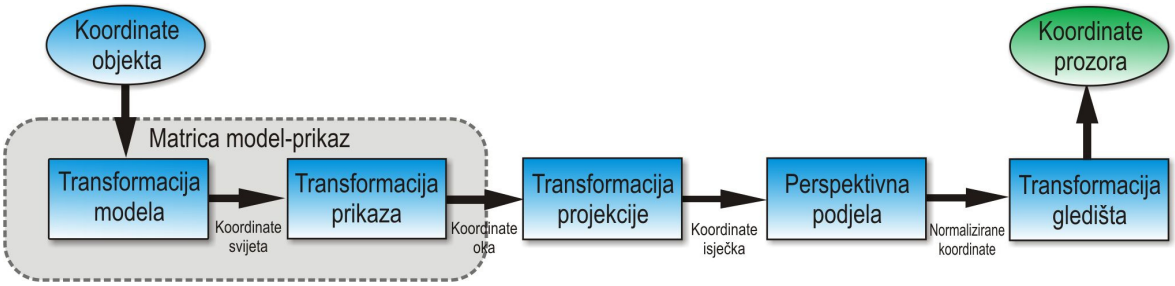
Projekcijska matrica (*GL_PROJECTION*). *GL_PROJECTION* matrica se koristi za definiranje krnje piramide (*engl. frustum*) volumena prikaza. Frustum određuje koji će objekti ili dijelovi objekata biti prikazani, a koji izrezani. Također određuje kako će se 3D scena projicirati na zaslonu. *OpenGL* nudi dvije funkcije za *GL_PROJECTION* transformaciju. *glFrustum()* definira perspektivnu projekciju, a *glOrtho()* definira ortografsku (paralelnu) projekciju. Obje funkcije zahtijevaju šest parametara za određivanje 6 ravnina izrezivanja geometrije.

Matrica teksture (*GL_TEXTURE*). Koordinate teksture se množe sa *GL_TEXTURE* matricom prije mapiranja teksture. Po zadanim postavkama, tekstura će se zalijepiti na objekte točno gdje su dodijeljene koordinate teksture. Modificiranjem *GL_TEXTURE* matrice, teksture se mogu pomicati, rotirati, rastezati i skupljati.

Matrica boje (*GL_COLOR*). Komponente boje (r, g, b, a) su pomnožene sa *GL_COLOR* matricom. Ova matrica se koristi za pretvorbu boja.

2.1.3.3 "Pipeline" transformacija

OpenGL protočna transformacija transformira koordinate vrhova u koordinate prozora, gdje mogu biti rasterizirani. Kao i svi 3D grafički sustavi, *OpenGL* koristi linearnu algebru i tretira vrhove kao vektore te ih transformira pomoću matrica. Proces transformacije naziva se protočnost, jer geometrija prolazi kroz nekoliko koordinatnih sustava na svojem putu do konačnog renderiranja. Svaki koordinatni sustav služi svrsi za jednu ili više *OpenGL* mogućnosti (URL 4).



Slika 6. OpenGL protočna transformacija.

Koordinate objekta. Aplikacije određuju vrhove u koordinatama objekta. Definicija koordinata objekta ovisi u cijelosti o aplikaciji. Neke aplikacije renderiraju scenu koja se sastoji od mnogo modela, svaki kreiran u svojem individualnim koordinatnom sustavu. Koordinate objekta smještene su u lokalni koordinatni sustav objekata i određuju početni položaj i orijentaciju objekata prije nego što se primjeni bilo koja transformacija.

Transformacija modela. Transformacijom modela transformira se geometrija iz koordinata objekta u koordinate svijeta. Aplikacije pohranjuju transformacije modela i prikaza u matricu model-prikaz kao jednu spojenu matricu. *OpenGL*

aplikacije gotovo uvijek koriste transformaciju modela za pomicanje geometrije iz lokalnog koordinatnog sustava objekta u globalni koordinatni sustav svijeta. Obično se koriste *glRotate*()*, *glScale*()* i *glTranslate*()* funkcije za pozicioniranje i orijentiranje geometrije i modela. Osim usmjeravanja geometrije statički, aplikacije koriste transformaciju modela za stvaranje animacije, tako da dinamički pomiču objekte od jednog okvira do drugog.

Koordinate svijeta. Koordinatni sustav svijeta je dio aplikacije koji postoji između *OpenGL* objekta i koordinatnog sustava oka. Matrica model-prikaz sastoji se od dvije transformacije, transformacija modela i transformacije prikaza. Koordinate objekta množe se sa transformacijom modela kako bi se dobile koordinate svijeta, te se koordinate svijeta množe sa transformacijom prikaza kako bi se proizvele koordinate oka. Budući da *OpenGL* ne obavlja operacije u koordinatnom sustavu svijeta, taj sustav nije dio formalne *OpenGL* protočne transformacije.

Transformacija prikaza. Transformacija prikaza je inverzna transformacija koja pozicionira i usmjerava kameru u koordinatnom sustavu svijeta aplikacije. Aplikacije pohranjuju transformaciju modela i transformaciju prikaza u matricu model-prikaz.

Koordinate oka. *OpenGL* koordinate oka definirane su na sljedeći način:

- točka gledišta je ishodišna točka,
- pravac gledanja je u smjeru negativne Z osi,
- pozitivan smjer Y osi usmjeren je prema gore,
- pozitivan smjer X osi usmjeren je prema desno.

Da bi geometrija bila vidljiva u standardnoj perspektivnoj projekciji, mora ležati negdje duž negativne Z osi u koordinatama oka. Geometrija koja se nalazi duž pozitivne Z osi je iza točke gledišta, zato, nije renderirana. Za prikaz geometrije s koordinatama u pozitivnom rasponu Z osi, aplikacija mora translatirati geometriju u negativan raspon Z osi koristeći matricu model-prikaz. Ovisno o trenutnom stanju, *OpenGL* izračunava rasvjetu, maglu, određene koordinate tekstura, izrezivanje definirano od strane programera, i veličine točaka iz vrijednosti koordinata oka.

Objekti se u *OpenGL*-u transformiraju iz koordinata objekata u koordinate oka koristeći *GL_MODELVIEW* matricu. *GL_MODELVIEW* matrica je kombinacija matrice modela i prikaza. Transformacija modela je pretvaranje koordinata iz prostora objekta u prostor svijeta, a transformacija prikaza je pretvaranje koordinata iz prostora svijeta u prostor oka.

$$\begin{pmatrix} x_{oko} \\ y_{oko} \\ z_{oko} \\ w_{oko} \end{pmatrix} = M_{model-prikaz} \cdot \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{pmatrix} = M_{model} \cdot M_{prikaz} \cdot \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{pmatrix}$$

Formula 3. Koordinate oka.

Kako bi se dobile homogene koordinate dodaje se dodatna dimenzija w. Treba imati na umu da ne postoji zasebna matrica kamere (prikaza) u *OpenGL*-u. Stoga, kako bi se simulirala transformacija kamere, scena (3D objekti i rasvjeta) mora biti transformirana inverznom transformacijom prikaza. Drugim riječima, *OpenGL* definira da se kamera uvijek nalazi na koordinati (0, 0, 0) i gleda u pravcu negativne Z osi u prostoru koordinata oka, i ne može se transformirati.

Transformacija projekcije. *OpenGL* množi koordinate oka sa matricom projekcije za proizvodnju koordinata isječka. Projekcijska matrica definira veličinu i oblik volumena prikaza, i tako određuje dio koordinata oka koje će u konačnici biti vidljive. Obično, aplikacija stvara ili perspektivnu ili ortografsku projekciju.

Koordinate isječka. *OpenGL* izreže osnovne geometrijske oblike koji se nalaze izvan volumena prikaza u prostoru koordinata isječka. Koordinate vrhova isječka su unutar volumena prikaza ako su x, y, i z vrijednosti unutar w do w raspona. Programeri se rijetko zamaraju time kako to zapravo radi. Ako aplikacija odredi dobro definiranu projekcijsku matricu pomoću *glOrtho()* ili *gluPerspective()* funkcija, isječak će se dogoditi kao što se očekuje (*OpenGL* izreže geometriju izvan volumena prikaza i ne izvodi nikakve daljnje obrade nad tom geometrijom). Koordinate isječka dobe se nakon primjene koordinata oka u *GL_PROJECTION* matrici.

$$\begin{pmatrix} x_{isječak} \\ y_{isječak} \\ z_{isječak} \\ w_{isječak} \end{pmatrix} = M_{projekcija} \cdot \begin{pmatrix} x_{oko} \\ y_{oko} \\ z_{oko} \\ w_{oko} \end{pmatrix}$$

Formula 4. Koordinate isječka.

Perspektivna podjela. *OpenGL* dijeli x, y i z vrijednosti koordinata isječka sa w vrijednostima kako bi se proizvele normalizirane koordinate. Ako je aplikacijom definirana perspektivna projekcija, perspektivna podjela se događa automatski. Kod perspektivne projekcije, perspektivna podjela učinkovito smanjuje daleku geometriju i širi bližu geometriju.

Normalizirane koordinate. Kod normaliziranih koordinata, x, y, i z vrijednosti vrhova leže u rasponu od -1.0 do 1.0. Dok prethodni koordinatni sustavi koriste homogene koordinate, s vrhovima koji se sastoje od x, y, z i w vrijednosti, normalizirane koordinate su 3D *Kartezijeve* koordinate. *OpenGL* ne obavlja nikakve radnje nad normaliziranim koordinatama. To je jednostavno koordinatni sustav koji postoji između perspektivne podjele i transformacije gledišta na koordinate prozora.

$$\begin{pmatrix} x_{normal.koor.} \\ y_{normal.koor.} \\ z_{normal.koor.} \end{pmatrix} = \begin{pmatrix} x_{isječak}/w_{isječak} \\ y_{isječak}/w_{isječak} \\ z_{isječak}/w_{isječak} \end{pmatrix}$$

Formula 5. Normalizirane koordinate.

Transformacija gledišta (*engl. viewport*). Transformacija gledišta je završna faza transformacije. Ova transformacija ne odnosi se samo na x i y vrijednosti, već i z vrijednosti. Ova transformacija normalizira z vrijednosti koordinata u rasponu prihvatljivom za vrijednosti dubinskog međuspremnika. Aplikacije obično postavljaju gledište kada dolazi do promjene veličine prozora. Gledište se postavlja sa pozivom na funkciju *glViewport()*.

Koordinate prozora. Koordinate prozora imaju svoje xy ishodište u donjem lijevom kutu prozora, a z vrijednosti koordinate prozora protežu se kroz dubinski međuspremnik. Programeri često čine pogrešku pretpostavljajući da su koordinate prozora cijeli brojevi. Te koordinate su zapravo vrijednosti točaka s pomičnim

zarezom (*engl. float*). Vrhovi koordinata prozora postoje u podpiksel lokacijama, a to je bitno za ispravnu rasterizaciju.

Da bi koncept koordinata prozora s pomičnim zarezom bio jasan, potrebno je zamisliti prozor kao prvi kvadrant *Kartezijevog* koordinatnog sustava s ishodištem u donjem lijevom kutu. Pozitivan smjer x osi usmjeren je prema desno, a pozitivan smjer y osi usmjeren je prema gore. Pikseli pokrivaju kvadratna područja, a cjelobrojne linije odgovaraju granicama piksela. Kao veliko pojednostavljenje procesa rasterizacije, *OpenGL* rasterizira piksele na temelju sjecišta područja piksela i xy koordinata prozora s pomičnim zarezom.

$$\begin{pmatrix} x_{prozor} \\ y_{prozor} \\ z_{prozor} \end{pmatrix} = \begin{pmatrix} \frac{w}{2} \cdot x_{normal.koor.} + \left(x + \frac{w}{2}\right) \\ \frac{h}{2} \cdot y_{normal.koor.} + \left(y + \frac{h}{2}\right) \\ \frac{f-n}{2} \cdot z_{normal.koor.} + \frac{f+n}{2} \end{pmatrix}$$

Formula 6. Normalizirane koordinate.

Koordinate prozora definirane su, na temelju zadanih parametara, uz pomoć dvije funkcije: *glViewport(x, y, w, h)* i *glDepthRange(n, f)*. Vrijednost *w* predstavlja širinu prozora, a vrijednost *h* je visina prozora. Vrijednost *n* predstavlja udaljenost bliže ravnine za izrezivanje geometrije, a vrijednost *f* je dalja ravnina izrezivanja.

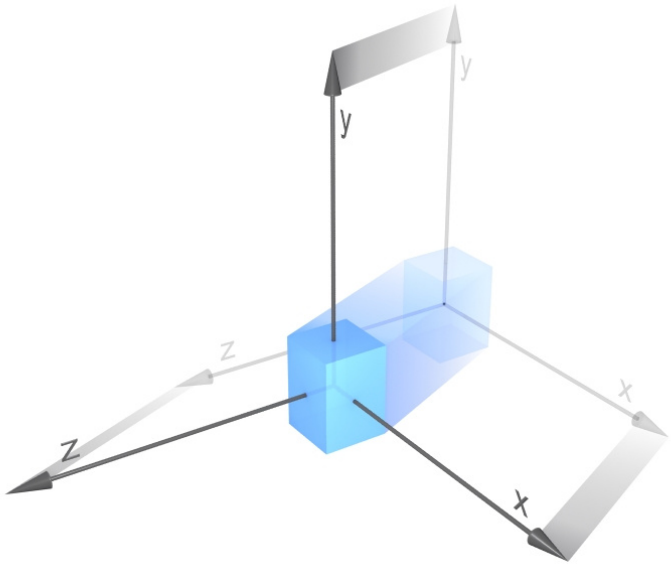
2.1.3.4 Transformacija modela

Tri *OpenGL* rutine za transformaciju modela su *glTranslate*()*, *glRotate*()* i *glScale*()*. Ove naredbe transformiraju objekt (ili koordinatni sustav) tako da ga pomiču, rotiraju, istežu, skupljaju, ili ga reflektiraju. Sve tri naredbe su ekvivalent za proizvodnju odgovarajuće matrice za translaciju, rotaciju i skaliranje. *OpenGL* automatski izračunava matrice.

Translacija

```
void glTranslate{fd}(TYPE x, TYPE y, TYPE z);
```

Množi trenutnu matricu sa matricom koja pomiče objekt sa zadanim *x*, *y* i *z* vrijednostima (ili pomiče lokalni koordinatni sustav za isti iznos).

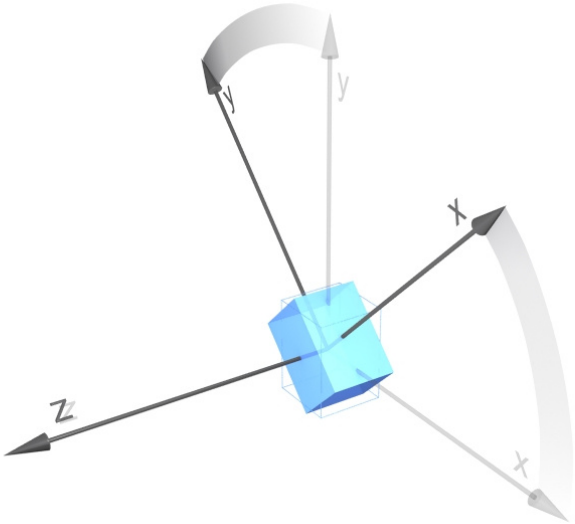


Slika 7. Translacija objekta.

Rotacija

```
void glRotate{fd}(TYPE angle, TYPE x, TYPE y, TYPE z);
```

Množi trenutnu matricu sa matricom koja rotira objekt (ili lokalni koordinatni sustav) oko pravca koji prolazi kroz ishodište i točku sa zadanim atributima (*x*, *y*, *z*), u smjeru suprotnom od smjera kazaljke sata. Parametar *angle* označava kut rotacije koji je izražen u stupnjevima.

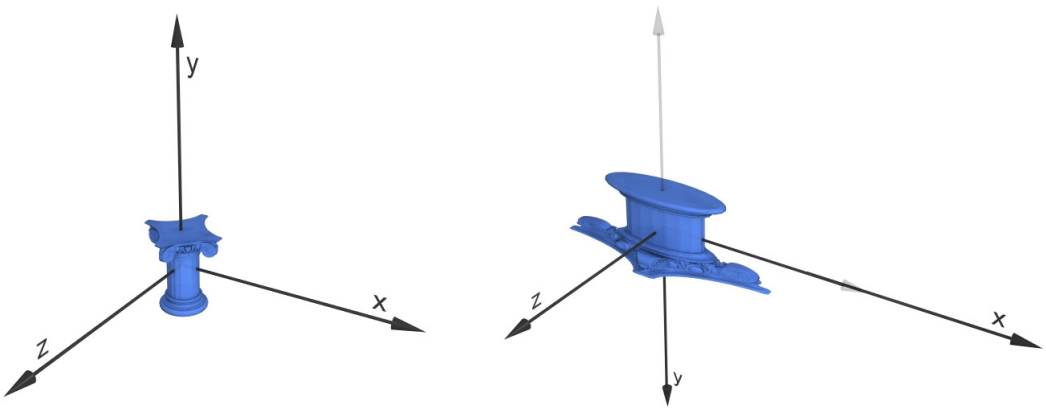


Slika 8. Rotacija objekta oko z osi.

Skaliranje

```
void glScale{fd}(TYPE x, TYPE y, TYPE z);
```

Množi trenutnu matricu sa matricom koja isteže, skuplja ili reflektira objekt duž osi. Svaka x, y i z koordinata svake točke objekta množi se sa odgovarajućim atributima *x*, *y*, *z*. Gledano od strane lokalnog koordinatnog sustava, lokalne koordinatne osi se istežu, skupljaju ili reflektiraju sa *x*, *y*, *z* atributima, a povezani objekt se transformira s njima.



Slika 9. Skaliranje i reflektiranje objekta duž x osi.

glScale()* je jedina, od tri funkcije transformacije modela, kojom se mijenja prividna veličina objekta. Skaliranjem s vrijednostima većim od 1.0 objekt se isteže, a vrijednostima manjim od 1.0 objekt se skuplja. Koristeći vrijednost -1.0, objekt se reflektira u odnosu na os. Ako su zadane vrijednosti (1.0, 1.0, 1.0), objekt ne mijenja svoju veličinu. Korištenje naredbe *glScale*()* trebalo bi biti ograničeno na slučajeve u kojima je to neophodno. *glScale*()* smanjuje učinkovitost izračuna rasvjete, jer se vektori normala moraju ponovno normalizirati nakon transformacije.

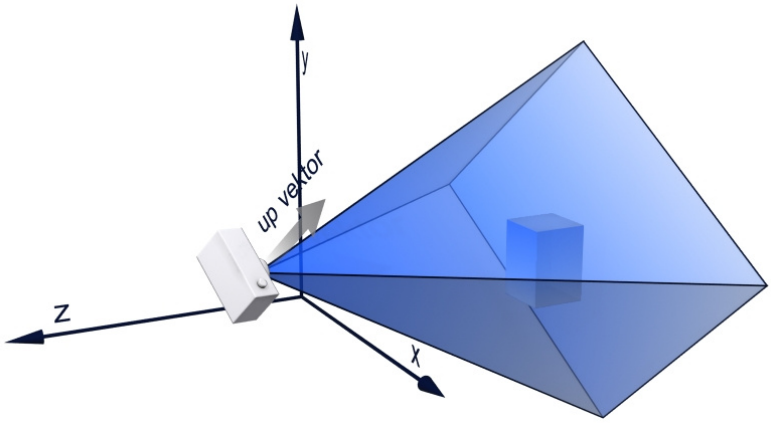
2.1.3.5 Transformacija prikaza

Transformacija prikaza mijenja položaj i orijentaciju gledište. Kao i kod stvarne kamere, transformacija prikaza pozicionira stativ kamere, usmjerujući kameru prema modelu. Da bi stvarna kamera bila usmjerena prema modelu, potrebno je kameru dovesti do željenog položaja i zarotirati sve dok ne gleda u željenom smjeru. Na istom principu se temelji transformacija prikaza i sastoji se od

određenih translacija i rotacija. Za postizanje završne slike, moguće je ili pomicati kameru ili pomicati sve objekte na sceni. Tako je, na primjer, transformacija modela koja rotira objekte u smjeru suprotnom od smjera kazaljke sata ekvivalentna transformaciji prikaza koja rotira kameru u smjeru kazaljke sata. Potrebno je imati na umu, da naredbe transformacije prikaza moraju biti pozvane prije izvođenja bilo kakve transformacije modela, tako da transformacije modela utječu samo na objekte.

Izvođenje transformacije prikaza moguće je na nekoliko načina:

- korištenjem jedne ili više naredbi transformacije modela (*glTranslate*()* i *glRotate*()*). Učinak ovih transformacija može se shvatiti na dva načina, kao pomicanje kamere ili pomicanje svih objekata u prostoru, u odnosu na stacionarnu kameru,
- korištenjem *OpenGL Utility Library* naredbe *gluLookAt()* za definiranje linije pogleda. Ova rutina se sastoji od niza naredbi za translaciju i rotaciju,
- izrada vlastitog algoritma koji će obuhvaćati naredbe za rotaciju i translaciju.



Slika 10. Pozicija kamere definirana naredbom gluLookAt().

```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, centerx,
GLdouble centery, GLdouble centerz, upx, GLdouble upy, GLdouble upz);
```

Definira matricu prikaza i množi je s desne strane trenutne matrice. Željeno gledište je određeno sa parametrima *eyex*, *eyey* i *eyez*. Parametri *centerx*, *centery* i *centerz* definiraju bilo koju točku duž željene linije pogleda, a obično se odredi

neka točka u centru scene prema kojoj se gleda. Parametri *upx*, *upy* i *upz* upućuju na to koji je smjer prema gore.

2.1.3.6 Perspektivna i ortografska projekcija

Uz navođenje položaja i orijentacije kamere, potrebno je navesti i druge atribute kamere, kao što je vidno polje, bliža i dalja udaljenost ravnina izrezivanja geometrije, i da li se koristi perspektivna ili ortografska projekcija. Aplikacija treba odrediti te parametre sa projekcijskom matricom.

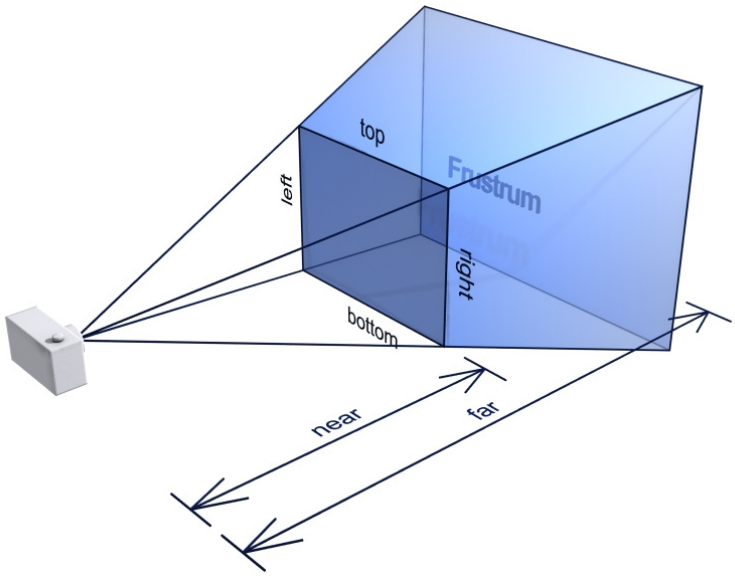
Nakon obavljanja operacija nad koordinatama oka, kao što je rasvjeta, *OpenGL* pretvara podatke vrhova u koordinate isječka pomoću projekcijske matrice. Ova matrica određuje oblik i veličinu volumena prikaza. Kada su podaci u prostoru koordinata isječka, *OpenGL* izreže svu geometriju izvan volumena prikaza i odbacuje tu geometriju.

Oblik projekcijske matrice određuje w vrijednost koordinate isječka. Ortografska matrica projekcije uvijek proizvode 1.0 za w vrijednost koordinate isječka, dok perspektivna matrica proizvodi niz w vrijednosti koordinata isječka. *OpenGL* pretvara koordinate isječka u normalizirane koordinate dijeljenjem koordinata isječka x, y, i z sa w vrijednostima koordinata isječka. U slučaju perspektivne projekcije, ova podjela uzrokuje da udaljena geometrija izgleda manja nego geometrija u blizini.

Perspektivna projekcija

Nedvojbeno najveća karakteristika perspektivne projekcije je ta da što je objekt dalje od kamere, čini se sve manji na konačnoj slici. To se događa zato što je volumen prikaza za perspektivnu projekciju krnja piramida (skraćena piramida čiji je vrh odsječen od ravnine paralelne sa vlastitom bazom). Objekti koji padaju unutar volumena prikaza projiciraju se prema vrhu piramide, gdje se nalazi kamera ili gledište. Objekti koji su bliže stajališta čine se većim, jer oni zauzimaju proporcionalno veći iznos volumena prikaza od onih koji su dalje. Ova metoda projekcije obično se koristi za animaciju, vizualne simulacije i kod bilo koje druge aplikacije koja teži višem stupnju realizma, jer na sličan način funkcioniraju naše oči i kamere.

Naredba za definiranje krnje piramide, *glFrustum()*, izračunava matricu kojom se ostvaruje perspektivna projekcija i množi se sa trenutnom projekcijskom matricom. Volumen prikaza se koristi za izrezivanje objekata koji se nalaze izvan njega. Četiri strane krnje piramide, njezin vrh i baza odgovaraju šest ravnina izrezivanja volumena prikaza, kao što je prikazano na slici (slika 11). Objekti ili dijelovi objekata izvan tih ravnina su izrezani iz konačne slike.



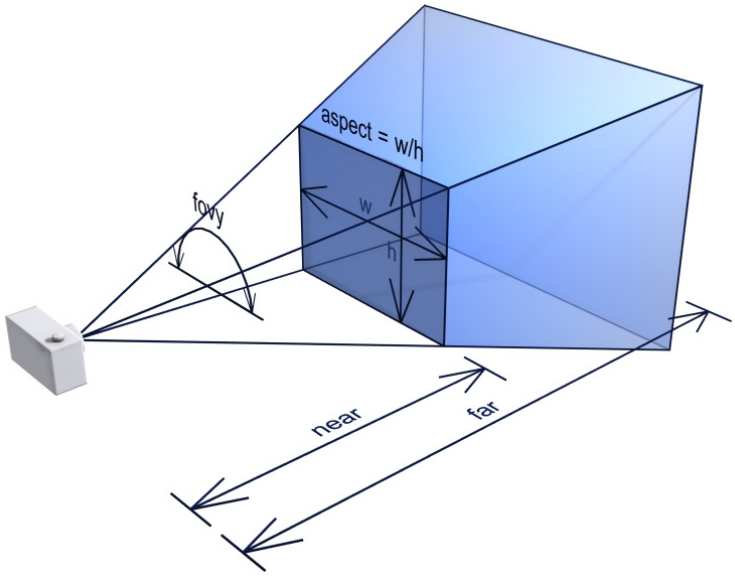
Slika 11. Perspektivni volumen prikaza definiran funkcijom glFrustum().

```
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
```

Stvara matricu za perspektivni prikaz i s njom množi trenutnu matricu. Krnja piramida je volumen prikaza definiran sa parametrima (*left*, *bottom*, *near*) i (*right*, *top*, *near*), koji određuju (x, y, z) koordinate donjeg lijevog i gornjeg desnog kuta. *near* i *far* atributi određuju udaljenost bliže i dalje ravnine izrezivanja od točke gledišta. Te vrijednosti moraju uvijek biti pozitivne.

Umjesto funkcije *glFrustum()* može se koristiti *gluPerspective()* funkcija, koja je dio *OpenGL Utility Library*-a. Ova rutina stvara volumen prikaza istog oblika kao i *glFrustum()*, ali na drugačiji način. Umjesto definiranja kutova bliske ravnine izrezivanja, određuje se kut vidnog polja Θ oko z osi i omjer širine i visine (w/h). Ova dva parametra su dovoljno kako bi se utvrdila krnja piramida duž linije gledanja. Može se također odrediti udaljenost između stajališta i bliže i dalje ravnine izrezivanja, čime se označava piramide. Treba imati na umu da

gluPerspective () funkcija stvara krnju piramidu koja je ograničena na simetriju po x i y osi duž linije gledanja, ali to je obično ono što se želi.



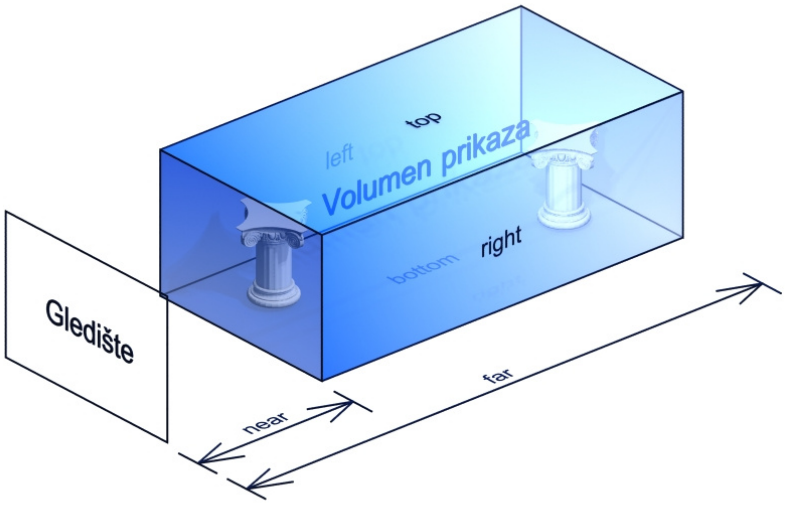
Slika 12. Perspektivni volumen prikaza definiran funkcijom gluPerspective().

```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);
```

Stvara matricu za simetrični perspektivni prikaz i množi je sa trenutnom matricom. *fovy* je kut vidnog polja Θ u yz ravnini i njegova vrijednost mora biti u rasponu od 0.0 do 180.0. *aspect* je omjer širine i visine. Vrijednosti *near* i *far* su udaljenosti između gledišta i ravnina izrezivanja, duž negativne z osi. One uvijek moraju biti pozitivne.

Ortografska projekcija

Kod ortografske projekcije volumen prikaza je pravokutni paralelopiped (kvadar). Za razliku od perspektivne projekcije, veličina volumena prikaza se ne mijenja od jednoga kraja do drugog, tako da udaljenost od kamere ne utječe na izgled veličine objekta. Ova vrsta projekcije koristi se kod aplikacija za arhitektonske nacrte i projektiranje uz pomoć računala, gdje je ključno održavati stvarnu veličinu objekata i kutove između njih, kao što su projicirani. Naredba *glOrtho()* stvara ortografski volumen prikaza. Kao i kod funkcije *glFrustum()*, definiraju se bliža i dalja ravnina izrezivanja. Ako ne postoji neka druga transformacija, smjer projekcije je paralelan sa z osi i pogled je usmjeren u smjeru negativne z osi.



Slika 13. Ortografski volumen prikaza

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
```

Stvara matricu za ortografski volumen prikaza i množi je sa trenutnom matricom. (*left*, *bottom*, *near*) i (*right*, *top*, *near*) su točke bliže ravnine izrezivanja, koje definiraju donji lijevi i gornji desni kut prozora gledišta. (*left*, *bottom*, *far*) i (*right*, *top*, *far*) su točke dalje ravnine izrezivanja, koje definiraju korespondentne kutove gledišta. *near* i *far* mogu biti pozitivni, negativni, pa čak i jednaki nuli. Međutim, *near* i *far* ne smiju imati jednake vrijednosti.

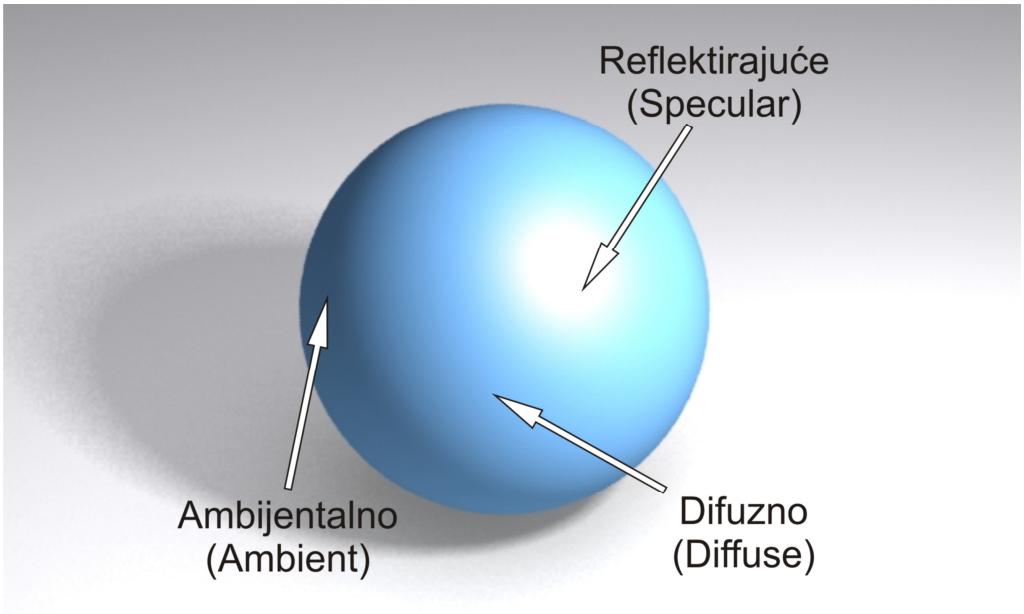
2.1.4. Rasvjeta

U stvarnosti, svjetlost je elektromagnetsko zračenje u vidljivom spektru. Prividna boja objekta je funkcija valne duljine, koja reflektira ili upija zrake svjetlosti koje emitira izvor svjetlosti prema objektu. Kao i većina 3D grafičkih *API*-a koji obavljaju radnje u realnom vremenu, *OpenGL* ima jednostavan model rasvjete koji je dovoljno snažan da na odgovarajući način aproksimira osvjetljenje stvarnog svijeta. *OpenGL* izračunava boju svakog piksela i prikazuje scenu koja je održana u videomeđuspremniku. Dio tog proračuna ovisi o tome kakva se rasvjeta koristi, te o tome kako objekti reflektiraju ili apsorbiraju svjetlo. *OpenGL* rasvjetom može se manipulirati i tako kreirati mnogo različitih vrsta efekata.

2.1.4.1 Ambijentalno, difuzno i reflektirajuće svjetlo

OpenGL ima tri vrste osvijetljena:

- *Ambijentalno svjetlo*. Ambijentalno svjetlo simulira indirektnu rasvjetu. Ona osvjetljava svu geometriju na sceni sa istim intenzitetom.
- *Difuzno svijetlo*. Difuzno svijetlo osvjetljava površinu na temelju njene orijentacije u odnosu na izvor svjetla. *OpenGL* difuzna rasvjeta pridržava se *Lambertovog* zakona, kojim je definirano da je iznos osvjetljenja proporcionalan kosinusu kuta između normale površine i vektora zrake svjetla (difuzno svjetlo reflektira jednako u svim smjerovima).
- *Reflektirajuće (engl. specular) svjetlo*. Reflektirajuće svjetlo aproksimira refleksiju izvora svjetlosti na sjajnoj površini. *OpenGL* reflektirajuće svjetlo pridržava se jednostavnog *Phongovog* modela osvjetljenja, u kojem reflektirajuće svjetlo naglašava vrhove duž vektora refleksije upadne svjetlosti.



Slika 14. Primjer ambijentalnog, difuznog i reflektirajućeg svjetla.

2.1.4.2 Kontrola rasvjete

OpenGL rasvjeta omogući se pozivom na naredbu *glEnable(GL_LIGHTING)*, a onemogućiti se naredbom *glDisable(GL_LIGHTING)*. Po zadanim *OpenGL* postavkama, rasvjeta je onemogućena i koristi se trenutna

primarna boja koja je postavljena naredbom *glColor3f()*. Kada je rasvjeta omogućena, *OpenGL* ne koristi trenutnu primarnu boju. Umjesto toga, koristi trenutnu boju materijala, koja je određena pozivom na naredbu *glMaterial*()*. Boja materijala određuje iznos ambijentalnog, difuznog, i reflektirajućeg svjetla, reflektiranog od objekata. *OpenGL* izračunava i difuzno i reflektirajuće svjetlo na temelju orijentacije geometrije s obzirom na izvor svjetla. Reflektirajuće svjetlo, uz položaj geometrije, ovisi i o položaju gledišta. Za orijentaciju koriste se normale vrhova.

```
// Minimalni kod rasvjete

// Omogućuje OpenGL rasvjetu
glEnable(GL_LIGHTING);

// Omogućuje jedan izvor svjetla
glEnable(GL_LIGHT0);

// Definirana geometrija zajedno sa normalama
//   ...
```

Kada je rasvjeta omogućena, *OpenGL* izračunava vrijednosti rasvjete kao dio obrade renderiranja vrhova. Za svaki vrh u svojoj geometriji, *OpenGL* zamjenjuje primarnu boju vrha s izračunatom vrijednosti osvjetljenja.

OpenGL izračunava vrijednosti rasvjete na sljedeći način:

- množi trenutnu boju ambijentalnog materijala sa ambijentalnim osvjetljenjem na sceni,
- množi trenutnu boju ambijentalnog materijala sa bojom ambijentalnog svjetla za svaki omogućeni izvor svjetla,
- za svako omogućeno svjetlo, *OpenGL* množi trenutnu boju difuznog materijala sa bojom difuznog svjetla i skalira rezultat sa skalarnim produktom trenutne normale i vektora svjetla,
- za svako omogućeno svjetlo, *OpenGL* množi trenutnu boju reflektirajućeg materijala sa bojom reflektirajućeg svjetla. Ako je skalarni produkt trenutne normale i vektora refleksije svjetla veći od nule, rezultat se skalira za taj skalarni produkt.

2.1.4.3 Normale

OpenGL koristi trenutne normale svakog vrha za određivanje orijentacije površine u odnosu na izvor svjetla i gledišta. Izračuni rasvjete događaju se u prostoru koordinata oka. *OpenGL* pretvara normale u koordinate oka pomoću matrice izvedene iz matrice model-prikaz. Trenutne normale utječu na rezultate difuznog i reflektirajućeg proračuna rasvjete. Jednadžbe *OpenGL* rasvjete proizvode realistične rezultate kada je trenutna normala jedinica duljine. Ako matrica model-prikaz ne sadrži transformaciju skaliranja, postići će se dobri rezultati osvjetljenja jednostavnim slanjem normale, koja odgovara jedinici duljine.

Matrica model-prikaz koje sadrži transformaciju skaliranja, proizvodi iskrivljene normale koordinata oka koje uzrokuju abnormalne rezultate rasvjete. To se obično događa kada aplikacija treba prikazati modele u jedinici prostora koja nije izvorna. *OpenGL* pruža dva načina za vraćanje iskrivljene normale na jedinicu duljine: reskaliranje normala (*engl. normal rescaling*) sa naredbom *glEnable(GL_RESCALE_NORMAL)* i normalizaciju sa naredbom *glEnable(GL_NORMALIZE)*.

2.1.4.4 Parametri materijala

Parametri materijala određuju kako površina reflektira svjetlo. Aplikacije mijenjaju *OpenGL* parametre materijala, kako bi se oponašali različiti obojeni materijali, sjajni i ravni materijali, materijali visokog sjaja ili materijali širokog sjaja kao što je mesing.

```
void glMaterial[fi]v(GLenum face, GLenum pname, const TYPE* param);
void glMaterial[fi](GLenum face, GLenum pname, TYPE param);
```

Naredba *glMaterial*v()* se koristi da se odredi ambijentalna, difuzna ili reflektirajuća boja materijala, a naredba *glMaterial*()* koristi se za određivanje parametra reflektirajućeg eksponenta. *face* određuje promjenu prednjeg ili leđima okrenutog parametra, ili oboje. *pname* je parametar materijala, a *param* je nova vrijednost za parametar.

2.1.5. Mapiranje tekstura

Mapiranje tekstura primjenjuje boje sa slike na svaki osnovni geometrijski oblik. Tijekom rasterizacije, *OpenGL* interpolira koordinate teksture te ih dodjeljuje svakom generiranom fragmentu. *OpenGL* koristi ove koordinate kao indekse za dobivanje teksela (elemenata teksture) sa slike teksture. Boje teksela izmjenjuju primarne boje fragmenata na temelju parametara stanja teksture.

OpenGL podržava četiri osnovne vrste tekstura:

- *1D teksture*. 1D teksturna mapa je 1D polje teksela. Aplikacije definiraju jednu *s* koordinatu teksture po vrhu.
- *2D teksture*. 2D teksturna mapa je slika koja se sastoji od 2D polja vrijednosti teksela. To je najčešći oblik mapiranja tekstura. Aplikacije definiraju i *s* i *t* koordinatu teksture po vrhu.
- *3D teksture*. 3D teksturne mape najčešće se koriste kod aplikacija za vizualizaciju 3D volumena teksela.
- *Mape tekstura kocke*. Mape tekstura kocke su setovi od šest 2D teksturnih mapa. Svaka od šest mapa predstavlja 90 stupnjeva vidnog polja svijeta sa aspekta geometrije koristeći teksture. Mape kocke se koriste za mapiranje okruženja i naglašavanja reflektirajućih vrhova.

Da bi se koristilo mapiranje tekstura, potrebni su sljedeći koraci:

- dobiti neiskorišten identifikator objekta teksture s *glGenTextures()* naredbom i stvoriti teksture objekta pomoću naredbe *glBindTexture()*,
- postaviti parametre stanja tekstura-objekt,
- odrediti sliku teksture pomoću *glTexImage2D()* ili *gluBuild2DMipmaps()* naredbe,
- prije renderiranja geometrije koja koristi objekte teksture, potrebno je vezati objekte teksture s naredbom *glBindTexture()*,
- prije renderiranja geometrije, omogućuje se mapiranje teksture,
- geometrija se šalje u *OpenGL* s odgovarajućim koordinatama teksture za svaki vrh.

2.1.5.1 Objekti tekstura

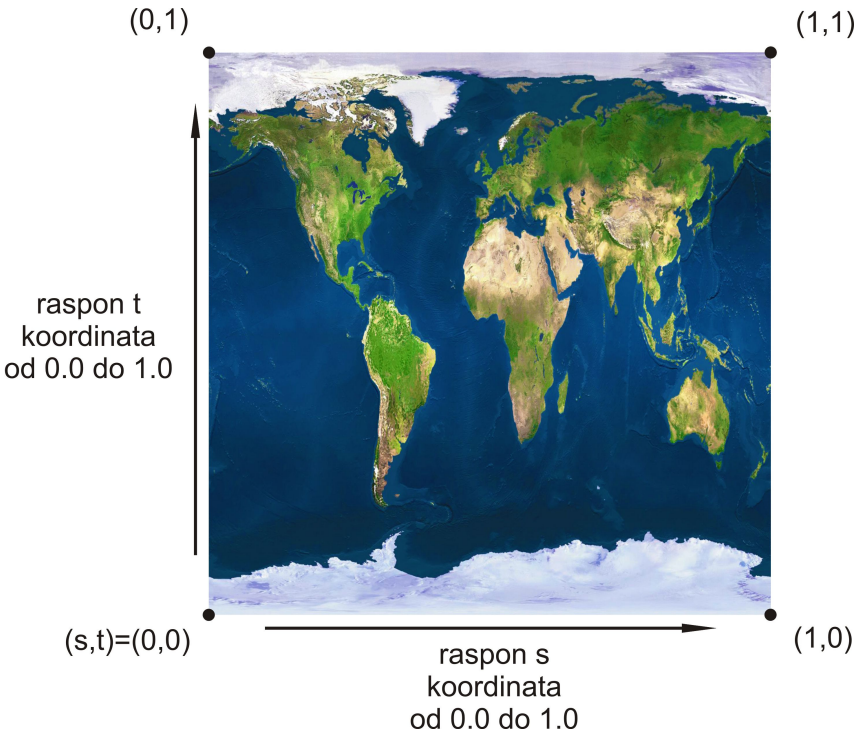
Objekti tekstura pohranjuju slike tekstura i njihova pridružena stanja. Većina *OpenGL* implementacija podržava ograničen radni skup objekata tekstura, kojeg obično implementira spremanjem slike teksture u posvećenu memoriju grafičkog hardvera. Aplikacije aktiviraju objekte tekstura izdavanjem *glBindTexture()* naredbe. Da bi se koristili objekti tekstura, aplikacija mora pribaviti, spremiti i na kraju korištenja, obrisati identifikatore tekstura-objekt. Za svaki objekt teksture koji aplikacija koristi, potrebno je spremiti sliku teksture u njega, zajedno s parametrima stanja teksture. Za dobivanje neiskorištenih tekstura-objekt identifikatora, koristi se naredba *glGenTextures()*. Za brisanje tekstura-objekt identifikatora, kada više nisu potrebni, poziva se *glDeleteTextures()* naredba.

```
void glBindTexture( GLenum target, GLuint texture );
```

Gornja naredba određuje aktivni objekt teksture. *target* mora biti *GL_TEXTURE_1D*, *GL_TEXTURE_2D*, *GL_TEXTURE_3D*, ili *GL_TEXTURE_CUBE_MAP*, ovisno o tome da li je stanje objekta teksture 1D, 2D, 3D, ili mape teksture kocke. *texture* navodi tekstura-objekt identifikator na koji se veže.

2.1.5.2 Koordinate teksture

Aplikacija dodijeli koordinate teksture svakom vrhu da poveže područja teksture s osnovnim geometrijskim oblicima. U jednostavnom 2D mapiranju tekstura, koordinate teksela promatraju se kao koordinate u *Kartezijevom* koordinatnom sustavu koji se sastoji od *s* i *t* osi (slika 15). Ishodište sustava je u donjem lijevom kutu slike teksture, tj. prvi piksel naveden u podacima parametara naredbe *glTexImage2D()*. Lokacija koordinate teksture (1, 1) nalazi se u gornjem desnom kutu slike, tj. zadnji piksel u bloku podataka. Moguće je pristupiti bilo kojem tekselu na slici teksture pomoću normaliziranih 2D koordinata teksture *s* i *t*, u rasponu od 0.0 do 1.0.



Slika 15. Slika teksture u normaliziranom koordinatnom sustavu.

Ako aplikacija određuje koordinate teksture izvan raspona 0.0 i 1.0, *OpenGL* koristi *GL_TEXTURE_WRAP_S* i *GL_TEXTURE_WRAP_T* parametre stanja za kontrolu pretraživanja teksela. Zadana vrijednost *GL_REPEAT* uzrokuje da se uzorak teksture ponavlja. Drugim riječima, *OpenGL* obavlja modalnu funkciju nad koordinatama teksture, tako da su koordinate teksture uvijek u rasponu od 0.0 do 1.0.

Slika 16. Sfera sa vezanom teksturom.

2.2. Wavefront OBJ format datoteke

Wavefront OBJ format je koristan standard za prikaz poligonalnih podataka u *ASCII* obliku. Koristi se za pohranu i razmjenu 3D podataka. *Wavefront OBJ* (objekt) datoteke koriste napredne vizualizacijske aplikacije za pohranu geometrijskih objekata sastavljenih od linija, poligona, i krivulja i površina slobodnog oblika. *Wavefront* je najpoznatiji po svojim zahtjevnim alatima za računalnu grafiku, koji uključuju modeliranje, animaciju i alate za izradu slika. Ti programi rade na snažnim radnim stanicama, kao što su one koje proizvodi *SGI*.

U *Wavefront* 3D aplikacijama, *OBJ* datoteke mogu biti pohranjene u *ASCII* formatu (koristeći **.obj* ekstenziju) ili u binarnom formatu (koristeći **.MORH* ekstenziju). *OBJ* datoteke češće su pohranjuju sa nastavkom **.obj*. Datoteke u binarnom formatu su vlasničke i nedokumentirane. Najnovija verzija *Wavefront OBJ* formata je v3.0, koja zamjenjuje prethodno v2.11 izdanje.

OBJ format datoteke podržava linije, poligone i krivulje i površine slobodnog oblika (URL 6). Linije i poligoni opisani su na temelju svojih točaka, dok su krivulje i površine definirane kontrolnim točkama i drugim podacima ovisno o vrsti krivulje. Format podržava razne vrste krivulja, uključujući i one utemeljene na *Bezijer*, *B-spline*, *Cardinal* (*Catmull-ROM spline*) i *Taylor* jednadžbama.

2.2.1. Struktura datoteke

OBJ datoteke ne zahtijevaju nikakvo zaglavlje (*engl. header*), iako je uobičajeno na početak datoteke postaviti liniju sa nekom vrstom komentara. Linija komentara počinje znakom ljestvi (#). Prazan prostor i prazni redci mogu slobodno biti dodani u datoteku, kao pomoć kod oblikovanja i čitljivosti. Svaki redak koji nije prazan, započinje s ključnim riječima. U produžetku linije slijede podaci vezani za tu ključnu riječ. Linije se čitaju i obrađuju do kraja datoteke. Linije mogu logički biti povezane s karakterom linije za nastavak (\), koji dolazi na kraj retka.

Sljedeće ključne riječi mogu biti uključene u *OBJ* datoteku. U ovom popisu, ključne riječi su raspoređene po vrsti podataka.

Podaci vrhova:

v – vrhovi geometrije
vt – vrhovi teksture
vn – vrhovi normala
vp – vrhovi prostornih parametara

Atributi krivulja i površina slobodnog oblika:

deg – stupanj
bmat – osnovna matrica
step – veličina koraka
cstype – tip krivulje ili površine

Elementi:

p – točka
l – linija
f – lice poligona
curv – krivulja
curv2 – 2D krivulja
surf – površina

Izvješća o tijelima sastavljenim od krivulja i površina slobodnog oblika:

param – vrijednosti parametara
trim – vanjska petlja podrezivanja
hole – unutarnja petlja podrezivanja
scrv – posebna krivulja
sp – posebna točka
end – kraj izvješća

Spajanje površina slobodnog oblika:

con – spajanje

Grupiranje:

g – naziv grupe
s – grupa zaglađivanja

mg – grupa spajanja

o – naziv objekta

Atributi za prikaz i renderiranje:

bevel – interpolacija nagiba

c_interp – interpolacija boje

d_interp – interpolacija razrješenja

lod – razina detalja

usemtl – ime materijala

mtllib – biblioteka materijala

shadow_obj – projiciranje sjena

trace_obj – traganje za zrakama svjetlosti (*engl. ray tracing*)

ctech – tehnika aproksimacije krivulje

stech – tehnika aproksimacije površine

2.2.2. Detalji datoteke

Wavefront OBJ datoteke najčešće sadrže samo lica poligona. Kako bi se opisao poligon, prvo se definira svaka točka s ključnom riječi *v*, a zatim opisuje lice s ključnom riječi *f*. Komandna linija lica sadrži brojeve točaka od kojih se poligon sastoji. Brojevi točaka služe kao indeksi u popisu točaka, i definirani su redom kako su točke zapisane u datoteci.

```
# Primjer jednostavne Wavefront OBJ datoteke koja opisuje trokut
v 0.0 0.0 0.0
v 0.0 1.0 0.0
v 1.0 0.0 0.0
f 1 2 3
```

OBJ datoteke ne sadrže definicije boja za lica poligona, iako mogu ukazivati na materijale koji su pohranjeni u posebnoj datoteci biblioteke materijala. Biblioteka materijala može se učitati pomoću ključne riječi *mtllib*. Biblioteka materijala sadrži RGB vrijednosti za difuzne, ambijentalne i reflektirajuće boje materijala, zajedno s drugim karakteristikama kao što su refrakcija, prozirnost, itd. *OBJ* datoteka ukazuje na ime materijala s ključnom riječi *usemtl*. Svim licima koja slijede dane su karakteristike ovog materijala, sve do sljedeće *usemtl* naredbe.

```
# Primjer kocke sa teksturom
# cube.obj

mtllib cube.mtl

# objekt Box01

v  -19.6850 0.0000 20.2391
v  -19.6850 0.0000 -19.1310
v  19.6850 0.0000 -19.1310
v  19.6850 0.0000 20.2391
v  -19.6850 39.3701 20.2391
v  19.6850 39.3701 20.2391
v  19.6850 39.3701 -19.1310
v  -19.6850 39.3701 -19.1310
# 8 vrhova

vn 0.0000 -1.0000 -0.0000
vn 0.0000 1.0000 -0.0000
vn 0.0000 0.0000 1.0000
vn 1.0000 0.0000 -0.0000
vn 0.0000 0.0000 -1.0000
vn -1.0000 0.0000 -0.0000
# 6 normala vrhova

vt 1.0000 0.0000 0.0000
vt 1.0000 1.0000 0.0000
vt 0.0000 1.0000 0.0000
vt 0.0000 0.0000 0.0000
# 4 koordinate tekstura

g Box01
usemtl matEarth
f 1/1/1 2/2/1 3/3/1 4/4/1
f 5/4/2 6/1/2 7/2/2 8/3/2
f 1/4/3 4/1/3 6/2/3 5/3/3
f 4/4/4 3/1/4 7/2/4 6/3/4
f 3/4/5 2/1/5 8/2/5 7/3/5
f 2/4/6 1/1/6 5/2/6 8/3/6
# 6 poligona
```

2.2.3. Biblioteka materijala

Datoteka biblioteke materijala sadrži jednu ili više definicija materijala, od kojih svaka uključuje boju, teksturu i mapu refleksije pojedinih materijala (URL 7). Materijali se primjenjuju na površine i vrhove objekata. Datoteke materijala su pohranjene u *ASCII* formatu i imaju **.mtl* ekstenziju. Komentari unutar datoteke počinju sa znakom ljestvi (#). Prazni redci mogu se umetnuti zbog jasnijeg prikaza. Datoteka se sastoji od niza *newmtl* naredbi, nakon čega slijedi definiranje različitih svojstava materijala.

Svaki opis materijala u datoteci sastoji se od *newmtl* naredbe, koja dodjeljuje naziv materijalu i označava početak opisa materijala. Nakon ove naredbe slijede naredbe za boju materijala, mapu teksture i mapu refleksije, koje opisuju materijal. Datoteke materijala sadrže mnogo različitih opisa materijala.

Sljedeće ključne riječi mogu biti uključene u datoteku biblioteke materijala. U ovom popisu, ključne riječi su raspoređene po vrsti podataka.

Ime materijala:

- *newmtl* – definira početak opisa materijala i dodjeljuje naziv materijalu.

Boja i osvjetljenje materijala:

- *Ka* – definira ambijentalnu boju materijala (r, g, b). Ambijentalne vrijednosti materijala množe se sa vrijednostima teksture. Zadani parametri su (0.2,0.2,0.2).
- *Kd* – definira difuznu boju materijala (r, g, b). Zadani parametri su (0.8,0.8,0.8).
- *Ks* – definira reflektirajuću boju materijala (r, g, b). Zadani parametri su (1.0,1.0,1.0).
- *Tf* – definira prozirnost materijala. Zadana vrijednost je 1.0 (bez prozirnosti). Neki formati koriste *d* umjesto *Tr.*
- *Illum* – označava osvjetljenje modela koje koristi materijal. *illum* = 1 označava ravan materijal bez reflektirajućih vrhova, tako da se vrijednost *Ks* ne koristi. *illum* = 2 označava prisustvo reflektirajućih vrhova, pa su atributi za *Ks* potrebni.
- *d* – definira prozirnost materijala. Zadana vrijednost je 1.0 (bez prozirnosti). Neki formati koriste *Tr* umjesto *d.*
- *Ns* – definira sjaj materijala. Zadana vrijednost je 0.0.
- *sharpness* – definira oštrinu refleksije iz lokalne mape refleksije.
- *Ni* – optička gustoća materijala (indeks loma). Vrijednost može biti u rasponu od 0.001 do 10.0. Vrijednost 1.0 znači da se svjetlost ne savija kod prolaza kroz objekt.

Mapa teksture:

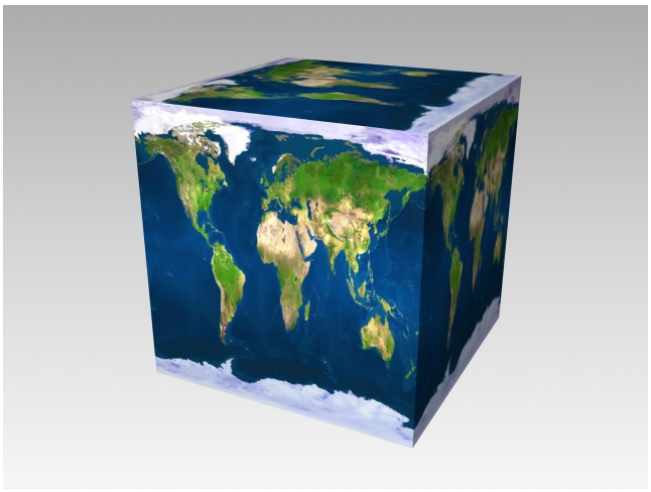
- *map_Ka* – definira ime datoteke koja sadrži ambijentalnu mapu teksture, koja se povezuje sa *Ka* vrijednostima. Kod renderiranja, *map_Ka* vrijednosti množe se sa *Ka* vrijednostima.
- *map_Kd* – definira ime datoteke koja sadrži difuznu mapu teksture.
- *map_Ks* – definira ime datoteke koja sadrži reflektirajuću mapu teksture.
- *map_Ns* – definira ime datoteke koja sadrži mapu teksture sa skalarnim vrijednostima za sjaj materijala.
- *map_d* – definira ime datoteke koja sadrži mapu teksture sa skalarnim vrijednostima za prozirnost materijala.
- *disp* – definira ime datoteke koja sadrži mapu teksture sa skalarnim vrijednostima za deformiranje površine.
- *decal* – definira ime datoteke koja sadrži mapu teksture sa skalarnim vrijednostima kako bi se zamijenila boja materijala sa bojom teksture.
- *bump* – definira ime datoteke koja sadrži mapu teksture sa skalarnim vrijednostima za prividno ispupčenje površine.

Mapa refleksije:

- *refl* - definira ime datoteke mape teksture, koja baca refleksiju na materijal sa beskonačno velike sfere.

```
# Primjer datoteke biblioteke materijala
# cube.mtl

newmtl matEarth
      Ns 10.0000
      Ni 1.5000
      d 1.0000
      Tr 1.0000
      Tf 1.0000 1.0000 1.0000
      illum 2
      Ka 0.0000 0.0000 0.0000
      Kd 0.5882 0.5882 0.5882
      Ks 0.0000 0.0000 0.0000
      Ke 0.0000 0.0000 0.0000
      map_Ka earth.jpg
      map_Kd earth.jpg
```



Slika 17. Model teksturirane kocke učitane iz OBJ datoteke

2.3. C++ programski jezik

C++ je programski jezik statičkog tipa, slobodnog oblika, više paradigme i opće namjene (URL 8). Smatra se kao jezik srednje razine, jer obuhvaća kombinaciju jezičnih mogućnosti visoke razine i niske razine. Razvijen je od strane programera *Bjarne Stroustrup*-a početkom 1979. godine u *Bell Labs*-u, kao pojašnjenje za *C* jezik. Izvorno je nazvan *C s klasama*, a kasnije je preimenovan u *C++*. *C++* je vrlo fleksibilan i prilagodljiv programski jezik. Ovaj jezik je poboljšao *C* programski jezik dodavanjem brojnih novih značajki, od kojih je najvažnija klasa (*engl. class*). Od svog osnivanja 1979. godine korišten je za različite programe, uključujući programiranje mikrokontrolera, operacijskih sustava, aplikacija i grafike.

C++ programski jezik je objektno-orijentirani aspekt jezika. Objektno-orijentirane tehnike su u središtu učinkovitosti svih alata koje pruža *Visual C++* programiranje za *Windows*-e. Objektno-orijentirano programiranje (*OOP*) je programiranje pomoću objektne strukture podataka, koja se sastoji od polja podataka i metoda, zajedno s njihovom interakcijom, za dizajn aplikacija i računalnih programa. Mnogi moderni jezici za programiranje imaju podršku za *OOP*.

Tri su osnovna alata koji se koriste za izgradnju *C++* aplikacija: kompajler, uređivač povezivanja (*engl. linker*) i arhivar (bibliotekar). Kompajler uzima *C++* izvorne datoteke kao ulazne i proizvodi datoteke objekata, koje sadrže mješavinu strojnog izvršnog koda i simboličke reference na funkcije i podatke. Arhivar uzima

zbirku datoteka objekta kao ulaznu i proizvodi statičku biblioteku, ili arhivu, što je jednostavna zbirka grupiranih datoteka objekata prikladnih za korištenje. Uređivač povezivanja uzima zbirku datoteka objekata i biblioteka, i rješava njihove simboličke reference za proizvodnju ili izvršne datoteke ili datoteke dinamičke biblioteke.

Microsoft Visual C++ (često skraćeno kao *MSVC* ili *VC++*) je komercijalni proizvod integriranog razvojnog okruženja (*engl. Integrated Development Environment - IDE*) *Microsoft*-a za *C*, *C++* i *C++/CLI* programske jezike. To je alat za razvoj i ispravljanje pogrešaka *C++* koda, pogotovo koda napisanog za *Microsoft Windows API*, *Direct3D API*, *OpenGL API* i *Microsoft.NET Framework*.

Microsoft.NET Framework je središnji koncept u *Visual C++*, kao i u svim ostalim *.NET* razvojnim proizvodima tvrtke *Microsoft*. *.NET Framework* se sastoji od dva elementa: vremena trajanja zajedničkog jezika (*engl. Common Language Runtime - CLR*[4]) u kojem se aplikacija izvršava, i skupa biblioteka pod nazivom *.NET Framework* biblioteke klasa. *.NET Framework* biblioteke klasa osiguravaju funkcionalnu podršku koja je potrebna kodu pri izvršenju sa *CLR*-om, bez obzira koji se programski jezik koristi, tako da svi *.NET* programi pisani u *C++*, *C#*, ili bilo kojim drugim jezikom koji podržava *.NET Framework*, koriste iste *.NET* biblioteke.

Postoje dvije bitno različite vrste *C++* aplikacija koje se mogu razviti sa *Visual C++*. To su aplikacije koje se izvorno izvršavaju na računalu. Ove aplikacije se nazivaju izvorni (*engl. native*) *C++* programi. Izvorni *C++* programi pišu se u verziji *C++* programskog jezika koji je definiran od strane *ISO*/*ANSI* (*engl. International Standards Organization / American National Standards Institute*) jezičnog standarda. Također, postoje aplikacije koje se izvršavaju pod kontrolom *CLR*-a, u proširenoj verziji *C++* programskog jezika pod nazivom *C++/CLI*. Ti programi imaju naziv *CLR* ili *C++/CLI* programi.

---

[4] **CLR** (*engl. Common Language Runtime*) – posebno okruženje vremena izvršavanja koje pruža temelje infrastrukture za *Microsoft. NET Framework*.

3. Koraci u izradi OpenGL OBJ preglednika

3.1. Kreiranje Win32 projekta

Windows API, neformalno *WinAPI*, je *Microsoft*-ov temeljni skup sučelja za programiranje aplikacija dostupnih u *Microsoft Windows* operacijskim sustavima (URL 9). Nekada se koristio naziv *Win32 API*, međutim, naziv *Windows API* točnije odražava svoje korijene u 16-bitnim *Windows*-ima i svoju potporu na 64-bitnim *Windows*-ima. Gotovo svi *Windows* programi imaju interakciju s *Windows API*-em.

Razvojna podrška dostupna je u obliku *Microsoft Windows SDK*-a (*engl. Software Development Kit*), pružajući potrebnu dokumentaciju i alate za izgradnju softvera koji se temelji na *Windows API*-u i pridruženoj *Windows* tehnologiji. *Windows API* izlaže veliki dio temeljne strukture *Windows* sustava programerima. To daje veliku prednost programerima kod fleksibilnosti i kontrole vlastitih aplikacijama. Međutim, također postoji velika odgovornost u rukovanju raznim operacijama niske razine, koje su vezane uz grafičko korisničko sučelje.

Verzije API-a:

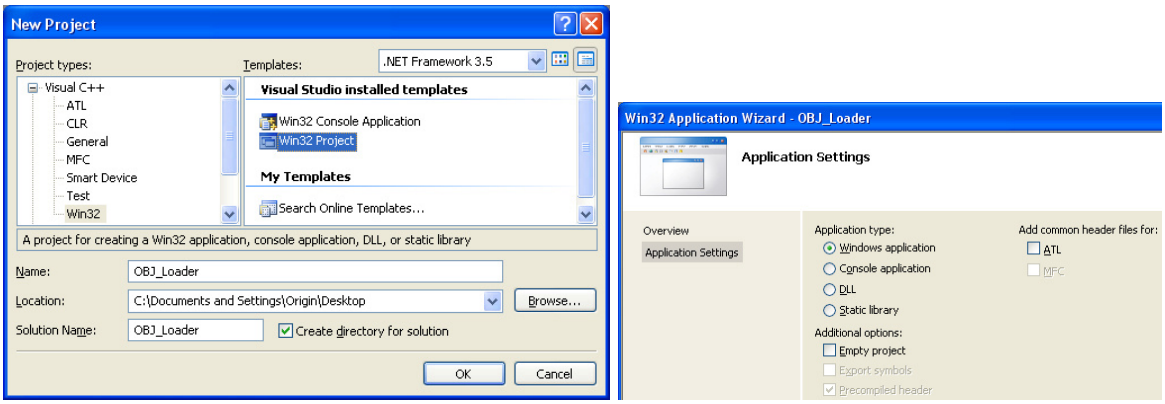
- *Win16* je *API* za prvu, 16-bitnu verziju *Microsoft Windows* operacijskog sustava. U početku je jednostavno imao naziv *Windows API*, ali je kasnije preimenovan u *Win16* u nastojanju da se razlikuje od novije, 32-bitne verzije sustava *Windows API*-a. *Win16 API* funkcije uglavnom žive u jezgri datoteka operacijskog sustava: kernel.exe (ili krnl286.exe ili krnl386.exe), user.exe i gdi.exe. Unatoč ekstenziji **.exe*, to su zapravo dinamički povezane biblioteke.
- *Win32* je 32-bitni *API* za moderne verzije operacijskog sustava *Windows*. *API* se sastoji od funkcija implementiranih u *DLL*[5] sustav. Glavni *DLL*-ovi *Win32 API*-a su Kernel32.dll, User32.dll i gdi32.dll. Win32 je predstavljen sa

[5] **DLL** (*engl. Dynamic Link Library*) – je *Microsoft*-ova implementacija koncepta zajedničkih biblioteka u *Microsoft Windows* i *OS/2* operativnim sustavima.

Windows NT operacijskim sustavom. Verzija *Win32 API*-a, koji je bio isporučen s *Windows 95* operacijskim sustavom, u početku ima naziv *Win32c* (*c* znači kompatibilnost). Taj termin je kasnije napušten od strane *Microsoft*-a u korist *Win32*.

- *Win32s* je ekstenzija za *Windows 3.1x* obitelj *Microsoft Windows* sustava, koja je implementirala podskup od *Win32 API* za ove sustave (*s* znači podskup).
- *Win32* za 64-bitne *Windows*-e, ranije poznat kao *Win64*, je varijanta *API*-a implementiranog na 64-bitnim platformama *Windows* arhitekture. 32-bitne i 64-bitne verzije aplikacije mogu još uvijek biti sastavljene iz jedne kodne baze, iako su neki stariji *API*-i zastarjeli, a neki *API*-i koji su zastarjeli u *Win32* potpuno su uklonjeni.

Za potrebe ovog projekta korišten je *Microsoft Visual Studio 2008*. *Visual Studio* je integrirano razvojno okružje (*engl. Integrated Development Environment - IDE*), proizvedeno od strane *Microsoft*-a (URL 10). Može se koristiti za razvoj konzola i grafičkih korisničkih sučelja aplikacija, *Windows Form* aplikacija, web stranica, web aplikacija, i dr. *Visual Studio* podržava različite programske jezike pomoću jezičnih usluga, koje omogućavaju uređivaču koda podršku (u različitim stupnjevima) gotovo bilo kojeg programskog jezika. Jedan od tih programskih jezika je i *Visual C++* koji je korišten za izradu ovog projekta.



Slika 18. Kreiranje Win32 projekta.

U cilju djelotvornog renderiranja teksturiranog 3D objekta pomoću *OpenGL*-a, potrebno je kreirati prozor u kojem je smješten prozor gledišta *OpenGL*-a. Unutar prozora gledišta biti će prikazana sva željena *OpenGL* geometrija. Novi *Microsoft*

Visual C++ projekt kreira se pomoću padajućeg izbornika *File* (*File>New>Project…*). Unutar okvira *Visual C++* programskog jezika, odabere se *Win32* projekt. Dodijeli mu se ime i lokacija gdje će biti spremljen. Za tip aplikacije odabere se *Windows application*.

Svakom *Windows* programiranju potrebna je glavna polazna točka. Glavna polazna točka za bilo koji *Windows* program zove se *WinMain*. Baš kao što *C++* program uvijek ima *main()* funkciju, *Win32* program treba poziv glavne funkcije *WinMain*. Svi 32-bitni *Windows* operativni sustavi koriste poziv konvencije *WINAPI*. Ovaj poziv konvencije mora se koristiti za razlikovanje funkcije kao polazne točke. Kada program počinje s radom, *WinMain* koristi četiri parametra.

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd);
```

hInstance upravlja instancom aplikacije, gdje se kao instanca može smatrati jedno pokretanje aplikacije. Instancu koriste *Windows*-i kao referencu za upravljanje događajima aplikacije, obradu poruka i raznih drugih zadaća. *hPrevInstance* koristi se ako je aplikacija imala bilo koju prethodnu instancu. Ako nema, ovaj argument se može zanemariti, što će uvijek biti slučaj. *lpCmdLine* je pokazivač niza i koristi se za držanje bilo kojeg argumenta komandne linije, koji su mogli biti navedeni kad je program počeo. *nShowCmd* je parametar koji određuje kako će prozor aplikacije biti prikazan nakon što počne izvršavanje.

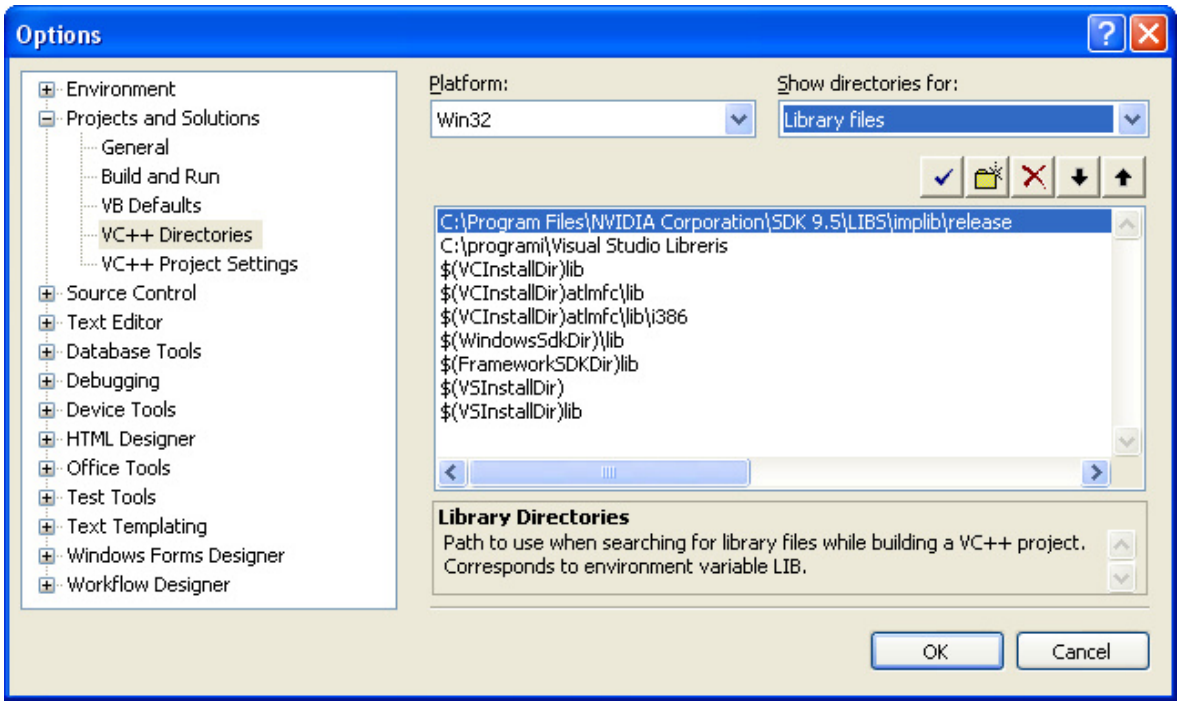
WndProc je funkcija za primanje svih ulaznih poziva usmjerenih na prozor *Windows*-a. Kada je klasa prozora registrirana, usmjerava se na ovaj dio koda za rješavanje poruka prozora.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
```

hWnd je jedinstvena drška (*engl. handle*) prozora. Mnogi prozori mogu biti kreirani iz iste definicije klase prozora, koja je prethodno registrirana. Dakle, dva kreirana prozora iste klase imati će drugačiju dršku, ali svaki će pozvati ovu funkciju s porukama. *uMsg* je poruka koju prozor prima. *wParam* i *lParam* su dodatne informacije o porukama. Svaka poruka poslana na prozor može imati broj dodatnih vrijednosti, npr. ako je pokazivač miš pomaknut u odnosu na prozor, onda dodatni parametri uključuju x i y položaj miša.

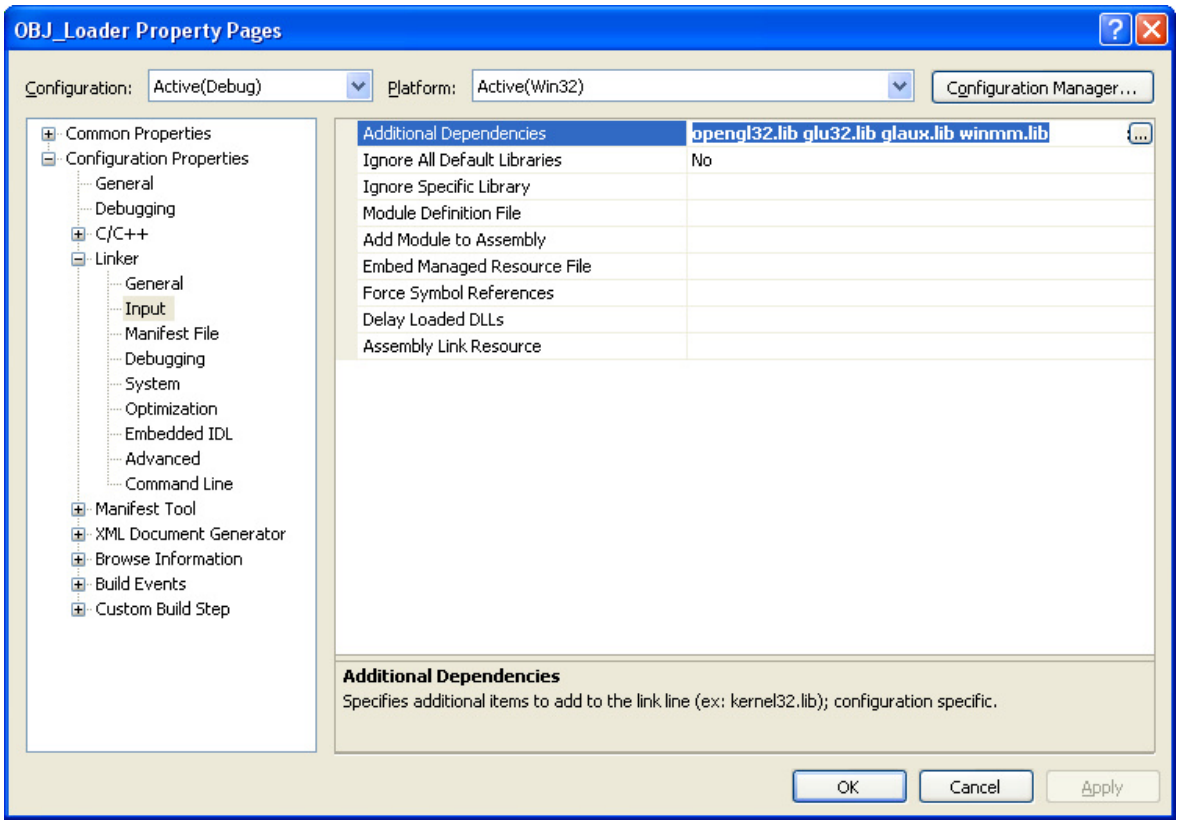
3.2. OpenGL implementacija

U cilju razvijanja *OpenGL* aplikacija, programerima je potreban *nVidia SDK*. *nVidia SDK* je zbirka tutorijala, dokumentacije, primjera i biblioteka uslužnih programa, dizajnirana za lakše učenje uzbudljivog svijeta 3D programiranja s *OpenGL* i *Direkt3D* bibliotekama. Nakon instalacije, svi se potrebni moduli i biblioteke implementiraju u *Visual Studio*. Ako je potrebno dodati ili ukloniti određene module i biblioteke, to se može učiniti pomoću opcija padajućeg izbornika *Tools* (*Tools>Options…*).



Slika 19. Dodavanje mapa sa ključnim modulima i bibliotekama.

Kada se izvorni kod pretvara u kompilirani, kompajler generira datoteke objekata (biblioteka) koje sadrže kompilirani kod. Ove datoteke objekata koriste se od strane uređivača povezivanja za stvaranje konačnog izvršnog programa. Datoteke objekata obično sadrže samo kod koji se koristi u programu. Zato je potrebno povezati datoteke biblioteka *OpenGL*-a sa *OpenGL* projektom, pomoću padajućeg izbornika *Project* (*Project>Proporties…*). Datoteke biblioteka služe za importiranje *DLL* datoteka.



Slika 20. Definiranje datoteka biblioteka za uređivač povezivanja.

Glavne *DLL* datoteke *OpenGL*-a (URL 11):

- *opengl32.dll* je modul koji sadrži jezgru *OpenGL* funkcija. Kako bi operativni sustav *Microsoft Windows* radio potrebna je ova datoteka biblioteke. Ovaj *DLL Windows*-i koriste kako bi se omogućilo korištenje *OpenGL* 3D grafike. Bilo koji softver koji koristi *OpenGL* za prikaz grafike, neće raditi, ako ova datoteka nije dostupna.
- *glu32.dll*, pod nazivom *Microsoft OpenGL Utility Library*, je datoteka biblioteke potrebna za pokretanje *Microsoft OpenGL* softvera. Biblioteka *glu32.dll* sadrži kodove i rutine koje su odgovorne za omogućavanje korištenja *OpenGL* formata. Ona sadrži industrijske standarde za 2D i 3D računalnu grafiku. *glu32.dll* sadrži do 250 različitih funkcija koje se mogu koristiti za izradu različitih trodimenzionalnih scena. Pruža podršku za operacije višeg reda koje nisu izravno dostupne u *OpenGL*-u, kao što su nadzor nad položajem i orijentacijom kamere, krivulje i površine višeg reda, geometrijski oblici kao što su valjci i sfere, projekcija u i iz prostora koordinata prozora, i tesalacija poligona. Također uključuje i funkcije za

automatsko skaliranje i međumapiranje kako bi se pojednostavila svojstva teksture.

- *glaux.dll* datoteka je modul *OpenGL*-a, koji omogućava programerima izradu 2D i 3D mapa teksture, filtra teksture, osvjetljenja i kontrole tipkovnice. Biblioteka se koristi i za određivanje načina miješanja boja, pomicanja bitmapa u 3D prostoru i za učitavanje i kretanje objekata kroz 3D svijet.

3.3. Postavljanje OpenGL prozora

U ovom podpoglavlju je opisano kreiranje *OpenGL* prozora i inicijalizacija njegovih postavki, te iscrtavanje geometrije unutar prozora. Glavna datoteka koja sadrži kodove za kreiranje prozora, inicijalizaciju postavki, iscrtavanje scene, te pozive na ostale module, je *main.cpp*. *main.cpp* je datoteka izvornog koda C++ programskog jezika, koja sadrži kodove za kreiranje *Windows* prozora i *OpenGL* prozora unutar njega. Na početku *main.cpp* datoteke potrebno je obuhvatiti datoteke zaglavlja (*engl. header*) za svaku biblioteku koja se koristi.

```
#include <windows.h>       // Datoteka zaglavlja za Windows-e
#include <gl\gl.h>         // Datoteka zaglavlja za OpenGL32 biblioteku
#include <gl\glu.h>        // Datoteka zaglavlja za GLu32 biblioteku
#include <gl\glaux.h>      // Datoteka zaglavlja za GLaux biblioteku
```

Zadaća *ReSizeGLScene()* rutine je promjena veličine *OpenGL* scene, kada se mijenja veličina prozora. Ako nije moguće promijenili veličinu prozora (na primjer, prikaz na cijelom zaslonu (*engl. fullscreen mode*)), ova rutina će i dalje biti pozvana (najmanje jednom prilikom pokretanja aplikacije). *OpenGL* scena će se mijenjati na temelju širine i visine prozora na kojem se prikazuje. Rutina sadrži i funkcije i matrice za postavljanje zaslona na perspektivni ili ortografski prikaz.

```
GLvoid ReSizeGLScene(GLsizei width, GLsizei height);
```

Ovom naredbom inicijalizira se i mijenja veličina *OpenGL* prozora. *width* i *height* predstavljaju širinu i visinu *Windows* prozora.

Rutina za inicijalizaciju postavaka *OpenGL*-a je *InitGL()*. *InitGL()* sadrži naredbe za boju zaslona, uključivanje dubinskog međuspremnika, glatko sjenčanje, postavke

osvjetljenja itd. Ova rutina neće biti pozvana sve dok *OpenGL* prozor ne bude kreiran.

```
int InitGL(GLvoid);
```

Funkcija za podešavanje *OpenGL* postavki. *GLvoid* znači da naredba ne prima ulazne parametre.

Sljedeća funkcija sadrži kod koji je potreban za crtanje 3D scene. *DrawGLScene()* rutina služi za prikaz sadržaja *OpenGL* zaslona. Između ostaloga, ova naredba čisti sadržaj ekrana i dubinski međuspremnik, resetira trenutnu matricu model-prikaz, te crta učitani 3D model iz *Wavefront OBJ* datoteke, sa funkcijom *DrawOBJModel()*. Ova rutina sadrži i kodove za translaciju, rotaciju i skaliranje objekata i kamere. Ako nije bilo nikakvih problema prilikom izvršavanja funkcije *DrawGLScene()*, povratna vrijednost biti će *TRUE*. Ako korisnik želi izaći iz aplikacije ili ako je došlo do pogrešaka unutar rutine *DrawGLScene()*, povratna vrijednost je *FALSE* i program će se zatvoriti.

```
int DrawGLScene(GLvoid) ;
```

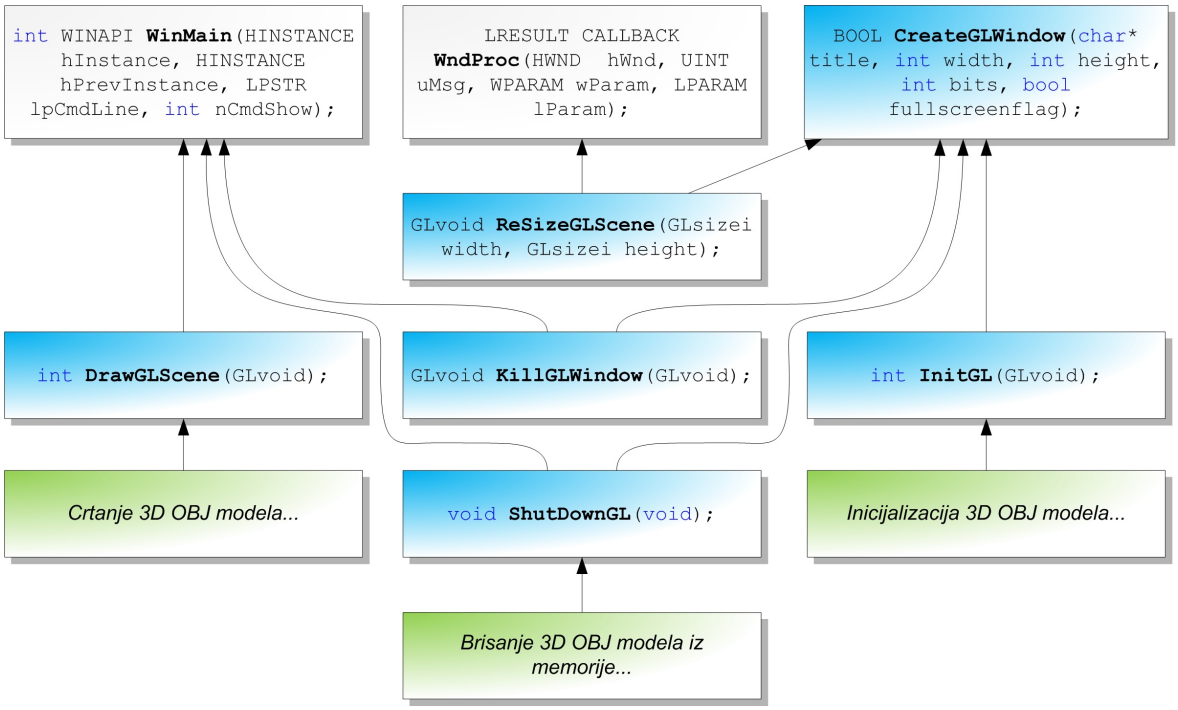
Navedena funkcija obavlja crtanje *OpenGL* scene. *GLvoid* znači da naredba ne prima ulazne parametre.

KillGLWindow() i *ShutDownGL()* su rutine pozvane neposredno prije zatvaranja programa. Posao *KillGLWindow()* naredbe je otpuštanje sadržaja renderiranja iz memorije, te uništavanje *OpenGL* i *Window*-s prozora. Naredba *ShutDownGL()* služi za brisanje sadžaja *OBJ* modela i tekstura iz memorije.

Procedurom *CreateGLWindow()* kreira se *OpenGL* prozor. Naredbe *ReSizeGLScene()*, *InitGL()* i *KillGLWindow()* pozivaju se od strane *CreateGLWindow()* procedure.

```
BOOL CreateGLWindow(char* title, int width, int height, int bits, bool
fullscreenflag);
```

Kao što se može vidjeti, procedura vraća *boolean* vrijednost (*TRUE* ili *FALSE*). Također uzima 5 parametara: *title* predstavlja naslov prozora, *width* širinu prozora, *height* visinu prozora, *bits* parametar bita (16/24/32), i na kraju *fullscreenflag* za prikaz na cijelom zaslonu ili u prozoru. Vraćena je *boolean* vrijednost koja ukazuje na to da li je prozor bio uspješno kreiran.



Slika 21. Dijagram kreiranja Windows i OpenGL prozora.

3.4. Inicijalizacija, crtanje i brisanje 3D OBJ modela

Pokretanjem aplikacije učitaju se podaci o modelu i teksturama, iz *Wavefront OBJ* datoteke i biblioteke materijala. Za to je potrebno nekoliko funkcija. Naredba *ReadOBJModel()* čita podatke iz *OBJ* datoteke, na temelju naziva datoteke i njenog mjesta na disku. Prvo se otvara datoteka i dodjeljuje se novi model. Zatim se napravi prvi prolaz kroz datoteku da bi se dobio broj vrhova, normala, koordinata tekstura i trokuta. U drugom prolazu kroz datoteku čitaju se njihovi podaci i dodjeljuje im se mjesto u memoriji.

```
OBJmodel* ReadOBJModel(char* filename);
```

ReadOBJModel () čita opis modela iz *Wavefront OBJ* datoteke. *filename* je naziv i put do datoteke koja sadrži podatke *Wavefront OBJ* formata. *OBJmodel* je struktura koja definira model.

Funkcija *OBJUnitize()* prilagođava položaj i veličinu modela sceni. Prvo se računa širina, visina i dubina modela. Sljedeći korak je definiranje središta modela. Nakon izračunatog faktora skaliranja, model se translatira u ishodište, te se skalira.

```
GLfloat OBJUnitize(OBJmodel* model);
```

Navedena funkcija unificira model tako da ga translatira u ishodište *OpenGL* koordinatnog sustava i skalira na jedinice unaprijed definirane kocke. model je pravilno inicijalizirana *OBJmodel* struktura.

OBJFacetNormals() funkcija služi za definiranje normala faceta unutar modela. Ako postoje, stare se normale brišu i dodjeljuje se memorija za nove normale faceta.

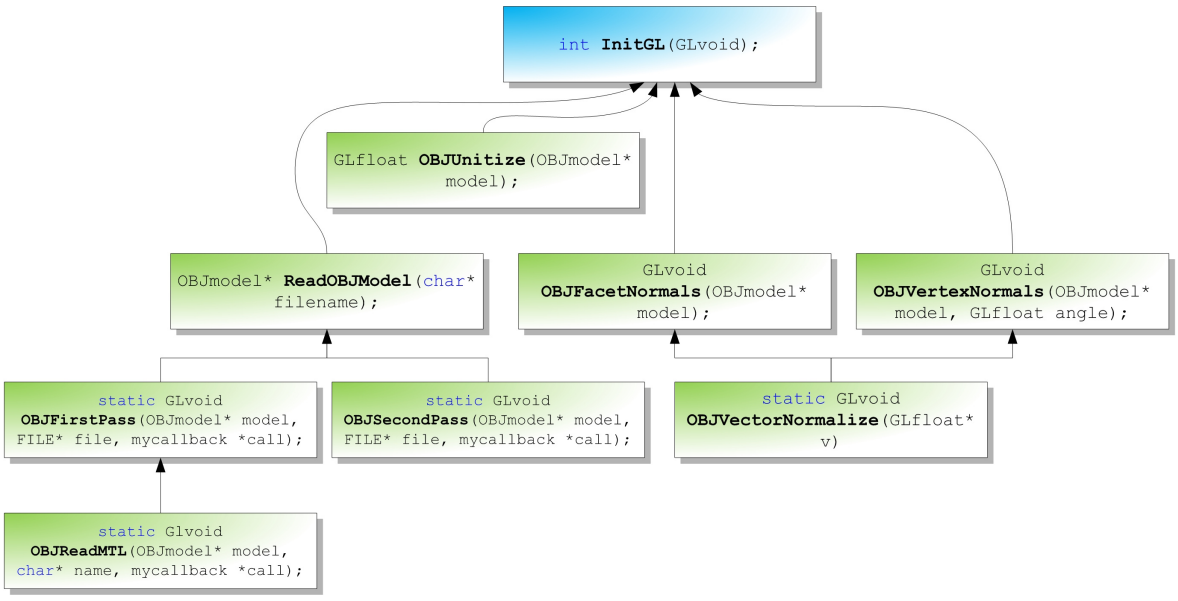
```
GLvoid OBJFacetNormals(OBJmodel* model);
```

Naredba *OBJFacetNormals()* generira normale trokuta modela (uzima vektorski produkt dvaju vektora koji proizlaze iz strana svakog trokuta). *model* je pravilno inicijalizirana *OBJmodel* struktura.

Naredba *OBJVertexNormals()* kreira normale vrhova kako bi se model mogao izglačati. Prvo se napravi popis svih trokuta u kojima je svaki vrh. Zatim petlja prolazi kroz svaki vrh u popisu osrednjavajući sve normale trokuta u kojima se nalazi svaki vrh.

```
GLvoid OBJVertexNormals(OBJmodel* model, GLfloat angle);
```

OBJVertexNormals() stvara normale vrhova za glačanje modela. *model* je pravilno inicijalizirana *OBJmodel* struktura. *angle* je maksimalni kut (u stupnjevima) glačanja u odnosu na kut između dva vektora normala. Kut glačanja ovisi o modelu, ali 90 stupnjeva je obično dovoljno.

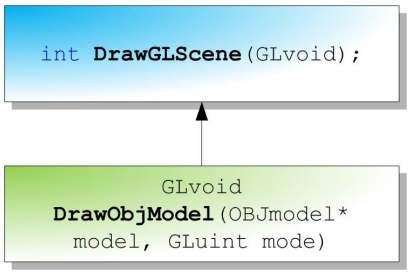


Slika 22. Dijagram inicijalizacije OBJ modela.

Kako bi *OBJ* model bio prikazan potrebno je pozvati funkciju *DrawObjModel()* koja ga crta. Ova naredba sadrži petlju koja prolazi kroz svaku grupu objekata, te uz pomoć prije sakupljenih podataka crta model sa teksturama, koji se sastoji od trokuta dobivenih sa funkcijama *glBegin()/glEnd()*.

```
GLvoid DrawObjModel(OBJmodel* model, GLuint mode);
```

DrawObjModel() renderira model pomoću trenutnog *OpenGL* sadržaja. *model* je pravilno inicijalizirana *OBJmodel* struktura. *mode* je vrijednost koja opisuje način renderiranja. Vrijednosti za *mode* argument mogu biti: *OBJ_NONE* (renderira samo vrhove), *OBJ _FLAT* (renderira s normalama faceta), *OBJ _SMOOTH* (renderira s normalama vrhova), *OBJ_TEXTURE* (renderira s koordinatama tekstura), *OBJ_COLOR* (renderira s bojama), *OBJ_MATERIAL* (renderira s materijalima).

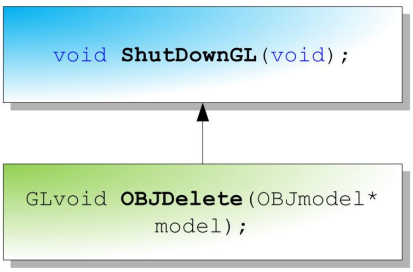


Slika 23. Dijagram crtanja OBJ modela.

Prije prestanka rada aplikacije potrebno je očistiti memoriju računala. Naredba *OBJDelete()* otpušta sve podatke modela, koji su učitani tijekom inicijalizacije, iz memorije. Bez ove funkcije svi bi podaci ostali u memoriji (nakon gašenja aplikacije) i gušili bi daljnji rad računala.

```
GLvoid OBJDelete(OBJmodel* model);
```

OBJDelete() briše *OBJmodel* strukturu. *model* je pravilno inicijalizirana *OBJmodel* struktura.



Slika 24. Dijagram brisanja OBJ modela iz memorije.

Slika 25. Završni izgled aplikacije sa učitanim modelom.

Slika 26. Prikaz aplikacije sa izbornikom, modelom "Zdenac života" i nebeskom sferom koja prikazuje okolinu.

4. Zaključak

Ovim je radom prikazan jedan, od više načina izrade *OpenGL* preglednika *Wavefront OBJ* datoteka. Opisane su neke od osnovnih značajki *OpenGL*-a i njegovog rada na *Windows* operativnim sustavima. Implementacijom *OpenGL*-a u sustav *Windows* prozora omogućuje se korištenje svih mogućnosti *Windows API*-a. Takvom implementacijom, aplikacija je vezana samo za *Windovs* operativne sustave. U ovom radu prikazane su i osobine *Wavefront OBJ* datoteka i njima priključenih datoteka biblioteka materijala. *OBJ* datoteke su moćne, ali jednostavne za shvaćanje, datoteke za prikaz 3D objekata i njima priloženih tekstura.

OpenGL je podržan na svim vodećim operativnim sustavima. Njegove naredbe mogu biti pozvane sa većinom programskih jezika. Sve *OpenGL* aplikacije proizvode dosljedan vizualni prikaz rezultata na bilo kojem kompatibilnom *OpenGL API* hardveru, bez obzira na operativni sustav. Činjenica da je *OpenGL* otvoreni standard, sigurno je jedan od razloga za njegovu popularnost.

Ovaj *OBJ* preglednik ne zahtjeva instalaciju i ne ovisi o drugim aplikacijama. Preglednik ovisi samo o *OpenGL* datotekama biblioteke, koje se smjeste na računalo prilikom instaliranja *drivera* za dotičnu grafičku karticu, koja podržava *OpenGL*. Relativno malim izmjenama koda, moguće je dodati nove module i tako izmijeniti izgled i mogućnosti aplikacije.

5. Literatura

Rost, Randi J.: OpenGL® Shading Language, Second Edition, Addison Wesley Professional, Boston, Massachusetts, 2006.

Shreiner, D.: OpenGL® Programming Guide, Seventh Edition, Pearson Education, Inc., Boston, 2009.

Wright, R.; Haemel, N.; Sellers, G.; Lipchak, B.: OpenGL® Superbible, Fifth Edition, Wesley Professional, Boston, Massachusetts, 2010.

Horton, I.: Beginning Visual C++ ® 2008, Wiley Publishing, Inc., Indianapolis, 2008

URL 1. OpenGL, http://en.wikipedia.org/wiki/OpenGL, 23.05.2011.

URL 2. What is OpenGL?, http://www.gldomain.com/about/, 24. 05.2011.

URL 3. OpenGL Rendering Pipeline, http://www.songho.ca/opengl/gl_pipeline.html, 02.06.2011.

URL 4. OpenGL Transformation, http://www.songho.ca/opengl/gl_transform.html, 05.06.2011.

URL 5. nVidia, http://www.nvidia.com/, 13.06.2011.

URL 6. Object Files (.obj), http://paulbourke.net/dataformats/obj/, 18.06.2011.

URL 7. MTL material format, http://paulbourke.net/dataformats/mtl/, 18.06.2011.

URL 8. C++, http://en.wikipedia.org/wiki/C%2B%2B, 20.06.2011.

URL 9. Windows API, http://en.wikipedia.org/wiki/Windows_API, 20.06.2011.

URL 10. Microsoft Visual Studio, http://en.wikipedia.org/wiki/Microsoft_Visual_Studio, 20.06.2011.

URL 11. OpenGL DLL-s, http://dll.paretologic.com/, 22.06.2011.

6. Popis slika

7. Prilozi

7.1. Sadržaj priloženog optičkog medija

Br.	Datoteka	Opis sadržaja
1.	DiplomskiRad.pdf	Tekst diplomskog rada
2.	OBJ_Viewer.exe	*OpenGL OBJ* aplikacija

8. Životopis

European curriculum vitae format

Osobne informacije

Ime	**Marić, Ranko**
Adresa	**Matenačka 4, 49240 D. Stubica, Hrvatska**
Telefon	**098/9214-258**
E-pošta	**ranmaric@gmail.com**
Državljanstvo	Hrvatsko
Datum rođenja	18.12.1982.

Radno iskustvo

• Datum (od – do)	07. travanj 2011. – 05. Svibanj 2011.
• Naziv i sjedište tvrtke zaposlenja	Ispostava za katastar D. Stubica, Trg Matije Gupca 20, D. Stubica
• Vrsta posla ili područje	Obavljao praksu
• Datum (od – do)	Rujan 2009.
• Naziv i sjedište tvrtke zaposlenja	JVP-Krapina, stacioniranje na Dugom Otoku
• Vrsta posla ili područje	Prevencija i gašenje požara, dobrovoljni vatrogasac
• Datum (od – do)	Lipanj 2007. – Kolovoz 2007.
• Naziv i sjedište tvrtke zaposlenja	Geodetska poslovnica, vl. Ankica Katanić, Toplička cesta 25, Donja Stubica
• Vrsta posla ili područje	Rad na terenu i izrada elaborata
• Datum (od – do)	Lipanj 2006 – Kolovoz 2006.
• Naziv i sjedište tvrtke zaposlenja	Geoprojekt d.o.o, V Ravnice 5, Zagreb
• Vrsta posla ili područje	Vektorizacija
• Datum (od – do)	20. studeni 2001. – 31. kolovoz 2002.
• Naziv i sjedište tvrtke zaposlenja	Sokol Marić i dr., Trg M. Tita 8/II, Zagreb
• Vrsta posla ili područje	Zaštitar

Obrazovanje

• Datum (od – do)	Srpanj 2002. – Srpanj 2011.
• Naziv i vrsta obrazovne	Geodetski fakultet u Zagrebu

ustanove	
• Osnovni predmet /zanimanje	
• Naslov postignut obrazovanjem	Diplomirani inženjer geodezije
• Stupanj nacionalne kvalifikacije (ako postoji)	-
• Datum (od – do)	Siječanj 2004. – Ožujak 2004.
• Naziv i vrsta obrazovne ustanove	Vatrogasna zajednica Krapinsko – Zagorske zajednice
• Osnovni predmet /zanimanje	
• Naslov postignut obrazovanjem	Vatrogasac
• Stupanj nacionalne kvalifikacije (ako postoji)	
• Datum (od – do)	1998. – 2001.
• Naziv i vrsta obrazovne ustanove	Geodetska tehnička škola, Zagreb
• Osnovni predmet /zanimanje	Geodezija, matematika
• Naslov postignut obrazovanjem	Geodetski tehničar
• Stupanj nacionalne kvalifikacije (ako postoji)	-

Ostale vještine

Stečene radom/životom, karijerom, a koje nisu potkrijepljene potvrdama i diplomama.

Strani jezik	**Engleski**
• sposobnost čitanja	Da.
• sposobnost pisanja	Da.
• sposobnost usmenog izražavanja	Da.
Organizacijske vještine i sposobnosti *Npr. koordinacija i upravljanje osobljem, projektima, financijama; na poslu, u dragovoljnom radu (npr. u kulturi i športu) i kod kuće, itd.*	Dobrovoljan rad u DVD-u D. Stubica (sudjelovanje na vježbama, natjecanjima i izlasci na požar)
Tehničke vještine i sposobnosti *S računalima, posebnim vrstama opreme, strojeva, itd.*	Programiranje u C++, Visual Basic, Java i ActionScript programskim jezicima. Programiranje u OpenGL-u. Poznavanje rada na slijedećem softveru: Microsoft Office, Autodesk Autocad, Autodesk 3D Studio Max, Adobe Photoshop, Adobe After Effect, Adobe Flash, Corel Draw, Microsoft Visual Studio, Eclipse.

Umjetničke vještine i sposobnosti *Glazba, pisanje, dizajn, itd*.	
Vozačka dozvola	Da, B kategorije.