Univerza v Mariboru
Fakulteta za elektrotehniko,
računalništvo in informatiko

Grega Žlahtič

PREPROSTI GEOMETRIJSKI MODELIRNIK

Diplomsko delo

Maribor, september 2011

Diplomsko delo univerzitetnega študijskega programa

PREPROSTI GEOMETRIJSKI MODELIRNIK

Študent:	Grega Žlahtič
Študijski program:	UNI ŠP Računalništvo in informacijske tehnologije
Mentor:	doc. dr. Simon Kolmanič
Somentor:	red. prof. dr. Nikola Guid

Maribor, september 2011

Univerza v Mariboru
Fakulteta za elektrotehniko,
računalništvo in informatiko

Številka: BRIT-74
Datum in kraj: 03. 06. 2011, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 1/2010)

SKLEP O DIPLOMSKEM DELU

1. **Gregi Žlahtiču,** študentu univerzitetnega študijskega programa RAČUNALNIŠTVO IN INFORMACIJSKE TEHNOLOGIJE, se dovoljuje izdelati diplomsko delo pri predmetu Računalniška grafika.

2. **MENTOR: doc. dr. Simon Kolmanič**
SOMENTOR: red. prof. dr. Nikola Guid

3. **Naslov diplomskega dela:**
PREPROSTI GEOMETRIJSKI MODELIRNIK

4. **Naslov diplomskega dela v angleškem jeziku:**
SIMPLE GEOMETRIC MODELLER

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije do 03. 06. 2012 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO RAČUNALNIŠTVO IN INFORMATIKO

Obvestiti:
- kandidata,
- mentorja,
- somentorja,
- odložiti v arhiv.

ZAHVALA

Zahvaljujem se mentorju doc. dr. Simonu Kolmaniču za pomoč in vodenje pri opravljanju diplomskega dela. Prav tako se zahvaljujem somentorju red. prof. dr. Nikoli Guidu.

Zahvalil bi se še staršem in bratu, za vso podporo, ki so mi jo nudili v času študiju. Hvala tudi punci Miji za vzpodbujanje in moralno podporo.

PREPROSTI GEOMETRIJSKI MODELIRNIK

Ključne besede: C++, OpenGL, GLUT, modeliranje, modelirnik, 3D vsebnostni test, računalniška grafika.

UDK: 004.43:004.94(043.2)

Povzetek

Pri izdelavi računalniških iger ali računalniških animacij se slej kot prej potrebuje kakšen 3D model. Prav zaradi tega smo izdelali modelirnik, program za modeliranje 3D modelov. Najprej smo opisali knjižnici OpenGL in GLUT, s pomočjo katerih smo program za modeliranje izdelali, nato pa predstavili še njegovo strukturo in delovanje. Program se osredotoča na metodo modeliranja s kvadrom, drugih tehnik pa nismo implementirali. Omogočena je uporaba osnovnih geometrijskih transformacij, in sicer rotacije, skaliranja in translacije, ki jih lahko na izbranih delih objekta uporabimo s pomočjo vsebnostnega testa in miške.

SIMPLE GEOMETRIC MODELLER

Key words: C++, OpenGL, GLUT, modelling, modeller, 3D content test, computer graphics.

UDK: 004.43:004.94(043.2)

Abstract

During the creation of computer game or a computer animation sooner or later a 3D model is needed. This was our reason for creating the modeller, a programme for modelling of 3D models. First we describe the OpenGL and GLUT libraries, which help us to create the modelling programme and then we demonstrate its structure and how iz works. The application focuses on the modelling method of "Box modelling", other techniques were not implemented. The modeller enables the use of the basic geometric transformations such as rotation, scaling and translation, which can be used on the chosen parts of the object with the help of a content test and a computer mouse.

KAZALO VSEBINE

KAZALO SLIK

1 UVOD

Pri izdelavi računalniških iger ali računalniških animacij se slej kot prej potrebuje kakšen 3D model. Model je matematična predstava 3D površine ali objekta, ki ga izdelamo s pomočjo specializirane programske opreme. Eden najpreprostejših načinov za izdelavo 3D modela je metoda, ki so jo znotraj animacijskega paketa 3DS Max poimenovali modeliranje s kvadrom (ang. *Box modelling*), pod drugačnimi imeni pa ga najdemo tudi v drugih animacijskih paketih. Za prikaz oziroma izdelavo 3D modela potrebujemo okolje, v katerega lahko model umestimo in ki nam omogoča delo z njim, zato bomo uporabili vmesnik OpenGL, ki predstavlja knjižnico za podporo 3D grafiki. V modelirniku modele izdelujemo s pomočjo geometrijskih transformacij in dodatnih funkcij, ki jih omogočajo posamezni modelirniki. Med bolj znanimi modelirniki sta Maya in 3DS Max.

Namen diplomske naloge je spoznati osnovna orodja in algoritme, ki se uporabljajo pri modeliranju 3D objektov. Prav tako je namen diplomske naloge izdelava programa za 3D modeliranje z metodo modeliranja s kvadrom, ki je napisan v jeziku C++, z uporabo knjižnic OpenGL in GLUT.

Diplomska naloga je razdeljena v šest poglavij. V drugem in tretjem poglavju je predstavljeno okolje, v katerem smo izdelali program za modeliranje. Tema poglavjema sledi poglavje, ki se nanaša na samo strukturo, izdelavo in delovanje modelirnika. Zajema več podpoglavij, med njimi so pomembnejša: Metoda modeliranja s kvadrom, Krmiljenje in Vsebnostni test. Peto poglavje predstavlja rezultate našega dela. Diplomska naloga se zaključi s sklepom, ki povzame napisano in podaja predloge za nadgradnjo.

2 VMESNIK OPENGL

OpenGL (ang. *Open Graphics Library*) je specifikacija standarda, ki določa programski vmesnik za pisanje 2D in 3D računalniških aplikacij. Pri tem se ne omejuje na programski jezik in tudi ne na platformo, zato ga lahko srečamo povsod, tako na mobilnih kot na stacionarnih napravah. Vmesnik OpenGL sestavlja več kot 250 funkcij, ki omogočajo izrisovanje kompleksnih 3D scen, sestavljenih iz preprostih geometrijskih oblik.

OpenGL v osnovi deluje tako, da sprejema osnovne objekte, kot so točke, daljice in mnogokotniki, ter jih pretvarja v slikovne pike. To delo opravlja grafični cevovod, ki je znan kot avtomat OpenGL (ang. *OpenGL state machine*). Večina ukazov OpenGL v grafični cevovod pošilja osnovne objekte, ali pa določa, kako naj jih cevovod obdeluje.

OpenGL je nizkonivojski proceduralni programski vmesnik, ki programerju omogoča, da do potankosti nadzoruje vsak korak, ki je potreben za izris scene. Nizkonivojska zasnova programerju daje večjo svobodo pri implementaciji novih algoritmov za izrisovanje, a zahteva dobro poznavanje grafičnega cevovoda.

OpenGL je v svoji zgodovini v precejšnji meri vplival na razvoj grafičnih pospeševalnikov. Promoviral je osnovno raven funkcionalnosti, ki je danes na voljo v strojni opremi:

- osnovni objekti so rasterizirane točke, daljice in mnogokotniki;
- cevovod za preoblikovanje in osvetljevanje;
- medpomnilnik Z za globino;
- preslikovanje tekstur;
- mešanje alfa. [11]

Svojo razširjenost med uporabniki lahko OpenGL pripiše svoji izvrstni kvaliteti dokumentacije. OpenGL ARB so izdali serijo priročnikov s specifikacijami, ki so bile posodobljene, da so se lahko zasledovale spremembe v uporabniškem vmesniku. Lahko bi dejali, da so priročniki skorajda najbolj prepoznavni po barvi naslovnic [10]:

- rdeča: programski vodič za vmesnik OpenGL,
- modra: priročnik za vmesnik OpenGL,
- zelena: predstavlja povezovanje med vmesnikom OpenGL in okenskim sistemom X ter knjižnico GLUT,
- bela: predstavlja povezovanje med vmesnikom OpenGL in Microsoft Windows,
- oranžna: priročnik za GLSL (ang. OpenGL Shading Language).

Seveda obstajajo še drugi pripomočki, ki jih lahko uporabimo za prikaz 3D scen. Sem spadata tudi knjižnica DirectX in programsko okolje Microsoft XNA, ki pa sta omejena samo na Microsoftove naprave. Microsoft XNA je namenjen predvsem preprosti izdelavi iger, zaradi tega ima funkcije za vstavljanje modelov že vključene, lastnoročno izrisovanje objektov pa je zelo zahtevno, zato se ne uporablja pogosto. DirectX je sicer enakovreden vmesniku OpenGL, od njega pa se razlikuje po zgoraj omenjeni omejenosti na Microsoftove naprave, kar je tudi razlog za omejenost na določene programske jezike (Delphi, C++, C#, .NET) [9].

Prav vsa ta dejstva so pripomogla k naši odločitvi za vmesnik OpenGL. Daje namreč veliko svobode in nadzora nad programi, ki jih izdelujemo. Seveda je k odločitvi pripomogla tudi dokumentacija, ki je bila zelo v pomoč pri delu, in kompatibilnost s prejšnjimi različicami vmesnika OpenGL.

3 KNJIŽNICA GLUT

GLUT oziroma *OpenGL Utility Toolkit* je koristna orodjarna za pisanje programov, temelječih na vmesniku OpenGL, ki je neodvisna od okenskega sistema. Knjižnica GLUT implementira preprost okenski programski vmesnik za delo z OpenGL, predvsem pa olajša učenje in raziskovanje programiranja v njem. Prav tako je neodvisen od platforme, zaradi česar lahko napišemo en sam program, ki ga lahko nato uporabljamo na bolj ali manj vseh napravah.

Knjižnica GLUT je kljub svoji preprostosti in prenosljivosti namenjena za izdelavo manjših do srednje velikih aplikacij OpenGL. Ker GLUT ni izpopolnjena orodjarna, ni primerna za večje projekte, predvsem kadar gre za zahtevnejše uporabniške vmesnike.

Knjižnica GLUT podpira:

- večokenski način za senčenje z vmesnikom OpenGL,
- procesiranje dogodkov,
- rastrske (ang. *bitmap*) in črtne (ang. *stroke*) pisave,
- preprost spustni meni,
- rutine za odsotno stanje (ang. *idle rutine*) in časovnik,
- komunikacijo z vhodno-izhodnimi napravami,
- rutine za generiranje ploskovnih (ang. *solid*) in žičnih (ang. *wireframe*) objektov. [5]

Za knjižnico GLUT smo se odločili zaradi njegove preprostosti, ki nam je omogočala, da smo se lažje osredotočili na samo programiranje, kot pa na uporabniški vmesnik in komunikacijo z vhodno-izhodnimi napravami. Na odločitev pa je pomembno vplivalo tudi dejstvo, da je neodvisen od operacijskega sistema.

4 MODELIRANJE 3D OBJEKTOV

V računalniški grafiki je modeliranje proces, pri katerem izdelamo matematično predstavo 3D površine ali objekta s pomočjo specializirane programske opreme. Tako nastal izdelek imenujemo model. Modele oblikujemo s pomočjo modelirnika, ki bo ne glede na to, katerega bomo izbrali, imel isto nalogo, se pravi izdelavo modelov. V to je vključeno spreminjanje, dodajanje in odstranjevanje delov modela.

Modelirniki se razlikujejo predvsem v količini možnosti, ki jih nudijo za modeliranje. V določenih modelirnikih lahko zato obdelujemo in uporabljamo le osnovna geometrijska telesa in osnovne geometrijske transformacije (ang. *geometric(al) transformations*). Modelirniki se razlikujejo tudi v dimenzijah (2D ali 3D).

Z modelirnikom lahko izdelamo veliko oziroma neskončno mnogo modelov, ki se predvsem razlikujejo po kvaliteti, odvisni od sposobnosti programske opreme, modelirnika in samega uporabnika. Na kvaliteto lahko prav tako vpliva postopek, s katerim bomo izdelovali model. Ena izmed pogosto uporabljenih metod je tako imenovano modeliranje s kvadrom.

Med najpogosteje uporabljenimi modelirniki so:

- Maya,
- 3DS Max, prvotno 3D Studio Max,
- Blender,
- LigthWave 3D.

Večina modelirnikov, tudi zgoraj omenjeni, imajo modelirno površino razdeljeno na štiri ločena okna, ki nam kažejo model iz različnih zornih kotov, se pravi skozi osnovne tri vzporedne projekcije in perspektivni pogled, kar nam pri modeliranju sicer olajša delo, hkrati pa nam zmanjša modelirno površino in s tem vidljivost.

S končanim modelom se delo z modelirnikom nikakor ne konča. Večina modelirnikov omogoča, da se modelom lahko dodajo teksture, kosti in animacije. Delo uporabnika je

zaradi tega lažje, saj mu ni treba vsega programsko izračunati, lahko namreč uporabi že vnaprej izračunane podatke.

Modelirnik, ki smo ga izdelali, je namenjen predvsem za metodo modeliranja s kvadrom in zato kot osnovni objekt vsebuje kocko, iz katere z osnovnimi geometrijskimi transformacijami izdelujemo model. Pri tem si pomagamo z ostalimi funkcijami modelirnika, kot so kopiranje objekta, razpolavljanje ploskev in možnost, da izvlečemo ploskev (ang. *extrude*). Imamo pa še možnost barvanja objektov, ploskev, in točk. Na tak način lahko izdelamo kvalitetne modele.

4.1 Metoda modeliranja s kvadrom

Med vsemi metodami, s katerimi lahko začnemo z modeliranjem, se nam je zdela najprimernejša metoda modeliranja s kvadrom. Ideja te metode je, da iz kocke izdelamo, kateri koli model želimo. Pri tej metodi si pomagamo predvsem z izvlečenjem ploskev ter dodajanjem robov in oglišč, ki jih z geometrijskimi transformacijami preoblikujemo v želeno obliko.

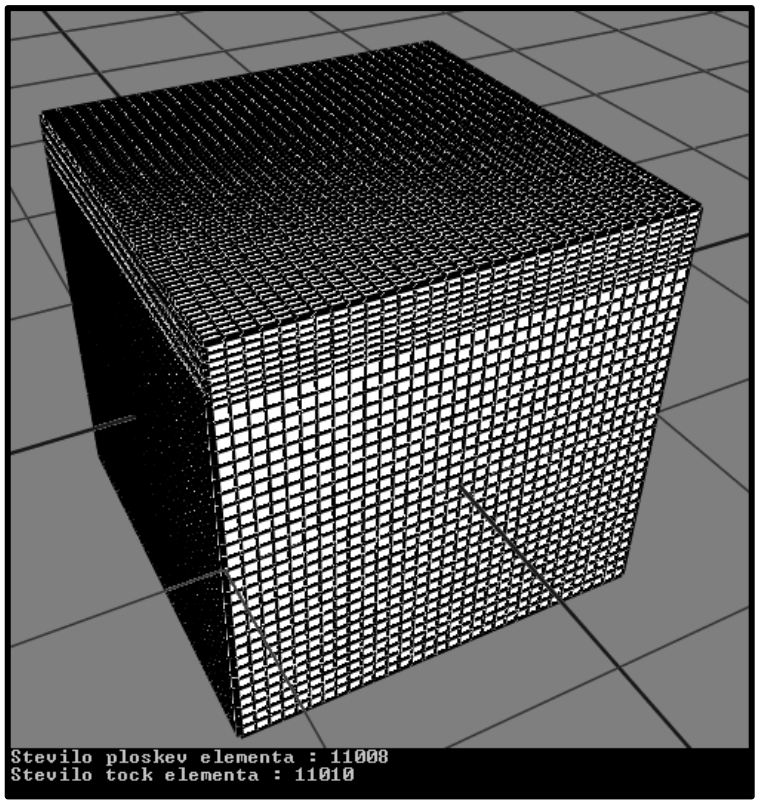
Glavna slabost te tehnike je izdelava okroglih modelov. Z geometrijskimi transformacijami namreč ne moremo oglišč objekta preoblikovati v okroglo obliko, lahko pa s pomočjo dodajanja robov in oglišč dobimo obliko, ki je na videz okrogla. V komercialnih modelirnikih je opisana težava veliko manjša, saj imajo podprte ploskve B-zlepkov.

Pri oblikovanju modela pa si lahko pomagamo tudi s sliko objekta, ki ga želimo modelirati in ga zato postavimo v ozadje modelirne površine. Slika nam predstavlja skico, po kateri izdelamo grob model, ki ga nato še fino obdelamo.

4.2 Struktura in delovanje modelirnika

Pri izdelavi programa je bil velik poudarek na sami strukturi, ki hrani podatke o objektih, da bi bil program čim manj potraten in bi pri tem izvedli čim manj operacij. Prav zaradi tega je že sama zasnova programa narejena tako, da vsebuje:

- razred *Objekti:* hrani osnovne podatke objekta, to pa so središče, barva, očrtani kvader za vsebnostni test ter razreda *Tocke* in *Ploskve*;
- razred *Tocke*: hrani vse točke, vsebovane v objektu, in barvo posamezne točke;
- razred *Ploskve*: hrani barvo ploskve in polje, ki hrani indekse točk, s čimer dosežemo to, da se točke ne podvajajo in je število ploskev ter točk skoraj enako, glej sliko 1.



Slika 1: Razmerje med ploskvami in točkami

Krmiljenje modelirnika poteka s pomočjo vnašanja ukazov v konzolo in s pomočjo kombinacije miške in tipkovnice. Ukazi konzole pri tem prevzemajo vlogo vnašanja podatkov, miška s pomočjo tipkovnice pa izvaja geometrijske transformacije in izbiranje predmetov.

4.2.1 Izbira objekta

Prav vsak program, ki ima možnost izbiranja omogočeno z miško, potrebuje nekakšen vsebnostni test. Ne glede na kompleksnost programa moramo preveriti ali se klik miške nahaja v objektu ali ne, kar v 2D programih ni zahtevno. 2D vrednost klika pa nam v 3D prostoru ne koristi kaj dosti, saj mu moramo izračunati še tretjo vrednost, to je globina. Čim pa kliku dodamo globino, dobimo 3D vektor, zaradi česar se tudi vse računske operacije pretvorijo v 3D vektorske in matrične izračune. Pretvorbo iz 2D klika v 3D lahko dosežemo s funkcijo *gluUnProject* [4], ponujeno v vmesniku OpenGL, ki nam klik spremeni v 3D točko. Za izvedbo vsebnostnega testa nam ena sama točka ne zadošča, saj zanj potrebujemo vektor, imenovan žarek (ang. *ray*). Dobimo ga na način, kot kaže izpis 1.

```
GLint viewport[4];
    GLdouble modelview[16],projection[16];
    GLfloat wx=mX,wy;

    glGetIntegerv(GL_VIEWPORT,viewport);
      //mY je y koordinata klika miške, ki pa ima točko (0,0) v zgornjem
levem kotu okna
      //OpenGL pa ima točko (0,0) v spodnjem levem kotu, zato moramo mY
odšteti od višine,
      //ki je enakovredno viewport[3]
    wy=viewport[3]-mY;
    glGetDoublev(GL_MODELVIEW_MATRIX,modelview);
    glGetDoublev(GL_PROJECTION_MATRIX,projection);

      //z koordinato nastavimo na 0, da dobimo točko, ki je najbližja
zaslonu
    gluUnProject(wx,wy,0.0f,modelview,projection,viewport,&ox,&oy,&oz);
      //z koordinato nastavimo na 1, da dobimo točko, ki je najbolj
oddaljena od zaslona

gluUnProject(wx,wy,1.0f,modelview,projection,viewport,&ox1,&oy1,&oz1);

      //začetna točka 3D vektorja
      r0 = vektor3D(ox, oy, oz);
      //končna točka 3D vektorja
      vektor3D r1 = vektor3D(ox1, oy1, oz1);
      //r predstavlja žarek, ki ga dobimo s klikom
      vektor3D r = r0 - r1;
      rs = r.smer();
```

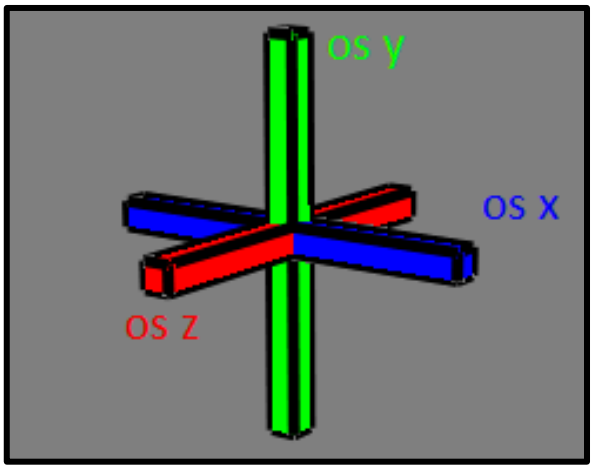
Izpis 1: Konstrukcija žarka

Pridobljen žarek nato uporabimo v vsebnostnih testih. V tem modelirniku sta za večjo učinkovitost vključena dva vsebnostna testa. Prvi vsebnostni test z metodo očrtanega

kvadra (ang. *Bounding box*) na hiter in nepotraten način pove, katere objekte žarek zadane. Nadaljnji vsebnostni test pa izračuna točko, v kateri se žarek in objekt sekata. Pridobljena točka se uporabi za nadaljnje preverjanje vsebnosti na delu objekta, ki ga uporabnik vnaprej določi. Ta način je najučinkovitejši, kadar se model izdeluje iz več objektov, saj tako zmanjšamo količino podatkov, ki jih je potrebno preveriti.

4.2.2 Obdelava izbranega objekta

Modeliranje se s časom oteži, saj se število točk, robov, ploskev in objektov, nad katerimi moramo imeti nadzor, povečuje. Uporaba geometrijskih transformacij se oteži tudi z večanjem števila generiranih objektov. Prav iz tega razloga naš modelirnik vsebuje poseben izris za izbrane dele objekta. Prav tako izbranemu objektu dodamo izris koordinatnih osi z izhodiščem v njegovem središču (slika 2).



Slika 2: Prikaz koordinatnih osi

Poseben izris za izbran del objekta je narejen z namenom, da ga lažje prepoznamo. Izris točke je na primer narejen tako, da lahko v trenutku vidimo vse robove, ki se bodo skupaj s točko spreminjali. Čeprav imamo s tem večjo preglednost, pa lahko kljub temu pride do neugodnih situacij pri izvajanju geometrijskih transformacij objekta po določeni osi. Pretvarjanje podatkov gibanja miške je lahko pri tem nenadzorovano zaradi manjših tresljajev, ki lahko pomenijo premik objekta ali dela v napačno smer. Zaradi tega ima modelirnik v središču objekta izrisane koordinatne osi, ki nam prav to situacijo preprečijo.

Izbira ene od koordinatnih osi objekta nam omogoča geometrijsko transformacijo izvesti vzdolž te osi.

4.2.3 3D osnovne geometrijske transformacije

Namen modelirnika je manipulacija že obstoječih objektov. Z njim spreminjamo njihov položaj, orientacijo in velikost. Modelirnik si pri manipulaciji že obstoječih objektov pomaga z geometrijskimi transformacijami [2].

V računalniški grafiki so osnovne geometrijske transformacije naslednje:

- **Translacija ali premestitev** (ang. *translation*).

  Translacija ali premestitev nam predstavlja premikanje objekta po prostoru. Translacijo objekta pa dosežemo s pomočjo množenja točke s translacijsko matriko.

$$\mathbf{T}_h = \mathbf{T}_h(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

- **Skaliranje ali razteg oziroma krčenje** (ang. *scaling*, *zooming*),

  Skaliranje nam omogoča razteg oziroma krčenje objektov po vseh oseh, kar lahko naredimo z enakomernim skaliranjem (ang. *uniform scaling*) ali neenakomernim skaliranjem (ang. *differential scaling*). Točko skaliramo tako, da jo pomnožimo s skalirno matriko.

$$\mathbf{S}_h = \mathbf{S}_h(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

- **Rotacija ali zasuk** (ang. *rotation*).

  Kadar imamo opravka z rotacijo v 3D prostoru in zbiramo informacije o rotaciji, imamo opravka z rotacijsko matriko. Rotacijske matrike so matrike, s katerimi želen vektor ali točko pomnožimo in za rezultat dobimo njeno rotirano vrednost. Poznamo več rotacijskih matrik, in sicer glede na os rotacije (*x*, *y* ali *z*):

$$\mathbf{R}_{z,h}(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

$$\mathbf{R}_{y,h}(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

$$\mathbf{R}_{x,h}(\gamma) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma & 0 \\ 0 & \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

Rotacija objekta v 3D prostoru pa lahko poteka tudi v vseh smereh hkrati, kar dosežemo s tem, da pomnožimo matrike posameznih osi skupaj.

$$\mathbf{R}_h(\gamma,\beta,\alpha) = \mathbf{R}_{z,h}(\alpha)\mathbf{R}_{y,h}(\beta)\mathbf{R}_{x,h}(\gamma) = \begin{bmatrix} A & B & C & 0 \\ D & E & F & 0 \\ G & H & I & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

Pri tem so A do I izrazi za [2]:

- $A = \cos\alpha \cos\beta$
- $B = -\sin\alpha \cos\gamma + \cos\alpha \sin\beta \sin\gamma$
- $C = \sin\alpha \sin\gamma + \cos\alpha \sin\beta \cos\gamma$
- $D = \sin\alpha \cos\beta$
- $E = \cos\alpha \cos\gamma + \sin\alpha \sin\beta \sin\gamma$
- $F = -\cos\alpha \sin\gamma + \sin\alpha \sin\beta \cos\gamma$
- $G = -\sin\beta$
- $H = \cos\beta \sin\gamma$
- $I = \cos\beta \cos\gamma$

V modelirniku se nismo odločili za uporabo matričnega množenja, ampak smo uporabili vektorsko množenje. Po prvih poskusih smo namreč ugotovili, da so bili matrični računi počasnejši.

4.2.4 Ostale funkcije

Poleg osnovnih geometrijskih transformacij so za metodo modeliranja s kvadrom pomembne še naslednje funkcije:

- **Kopiranje objekta**.

  Pogosto se pojavi situacija, v kateri je potrebno imeti dva enaka ali pa le enako oblikovana objekta. V takšnih primerih bi bila izdelava enakega objekta nesmiselna. To je razlog, da modelirnik vsebuje možnost kopiranja, s katero določen objekt v celoti preslikamo v nov objekt.

- **Izvlečenje ploskev.**

  Izvlečenje ploskev nam daje možnost, da iz objekta izvlečemo ploskev, ne da bi pri tem vplivali na sosednje ploskve. Pri tem postopku pa izvlečeni ploskvi izdelamo nove sosednje ploskve.

- **Razpolavljanje ploskev.**

  Modeliranje, predvsem metoda modeliranja s kvadrom, potrebuje možnost za razpolavljanje ploskev. Dodatne ploskve in točke, nastale pri tem postopku, dajejo možnost, da se iz preprostega geometrijskega telesa naredi kompleksen model. V tem modelirniku lahko izberemo rob in razpolovimo ploskve, na katerih obstaja nasprotni rob izbranemu robu in tako nadaljujemo, dokler ne razpolovimo celotnega objekta. Funkcija za razpolavljanje najprej izračuna središče roba, kar naredi izpis 2.

```
//p0 in p1 sta točki roba
vektor3D v = p0-p1;
vektor3D vs = v.smer();
vektor3D vd = v.dolzina();
//izračunamo točko p, ki je na sredini roba med točko p0 in p1
vektor3D p = p0 + vs*(vd/2);
```

Izpis 2: Izračun središča roba

4.3 Krmiljenje z miško in tipkovnico

Krmiljenje je pomemben del v modelirniku, ki lahko uporabniku olajša delo. Modelirniki imajo velik nabor operacij, ki pa se med verzijami zelo spreminjajo. Prav iz tega razloga večina modelirnikov uporablja bližnjice s pomočjo tipkovnice. S kombinacijo tipk in miške pridobimo na učinkovitosti pri delu.

V našem modelirniku smo s pomočjo upravljavca dogodkov GLUT za detektiranje pritiska tipke uporabljali funkcijo *glutKeyboardFunc* in funkcijo *glutKeyboardUpFunc* za izpust tipke. Prav tako nam GLUT omogoča spremljanje klika miške z *glutMouseFunc* in premik miške z *glutMotionFunc*. S kombinacijo teh funkcij lahko hitro prehajamo med geometrijskimi transformacijami in ostalimi funkcijami. Pri implementiranem modelirniku imamo kar nekaj takšnih bližnjic in te so:

- tipka *a* in desni klik omogočata premikanje modelirne površine navzgor in navzdol ter levo in desno;
- tipka *s* in levi klik omogočata uporabo translacije nad izbranim delom objekta;
- tipka *d* in levi klik omogočata uporabo skaliranja nad izbranim delom objekta;

- tipka *f* in levi klik omogočata aktiviranje rotacije nad izbranim delom objekta;
- tipka *q* ponastavi nastavitve kamere;
- tipko *w* uporabimo za izvlečenje ploskve;
- tipka *e* razpolovi objekt glede na rob, ki ga izberemo;
- tipka *esc* omogoča izhod iz programa;
- tipka *del* izbriše izbran objekt;
- tipka *tab* odpre/zapre konzolo;
- levi klik miške izbere objekt;
- desni klik miške rotira globalni koordinatni sistem;
- sredinski klik miške približuje in oddaljuje kamero.

4.4 Konzola

Pri delu je najbolje, če je kar največji del okna na razpolago delu z objekti in ni v napoto nepotrebnih menijev, zato smo se odločili, da bomo za nadzor dela z modelirnikom in vnos ukazov uporabljali tipke in konzolo, glej sliko 3. Konzola je v tem primeru vnosno polje za dodajanje in barvanje objektov. Z njo določamo, kateri del objekta bomo izbrali (točka, rob, ploskev, celoten objekt). Da pa konzola ne bi bila moteča, jo lahko s pritiskom na tipko *tab* zapiramo in odpiramo, omogočena pa je tudi prosojnost.

Slika 3: Prikaz znakov v konzoli

Vgrajene funkcije za izdelavo konzole ni ne v knjižnici GLUT in ne v vmesniku OpenGL, zato jo je bilo potrebno v celoti lastnoročno izdelati. Izdelali smo svoj razred, imenovan Konzola, ki je prevzel vso delo s konzolo. Tukaj so se tudi pojavila vprašanja, kako delati v vmesniku OpenGL v ravnini, kako prikazati besedilo in seveda, kako narediti, da ne bo konzola preveč moteča.

4.4.1 OpenGL v ravnini

Kot že omenjeno, je OpenGL narejen za izdelavo 2D in 3D scen, ki jih lahko izdelamo vsako zase ali pa jih v določenih primerih združimo. Za konzolo vsekakor želimo, da se izriše v ravnini, saj je predstavljena z ravno ploskvijo, brez globine. Prav tako je nezaželeno, da bi jo z ostalimi 3D objekti premikali po prostoru. Zaradi tega v času izrisa konzole spremenimo projekcijo iz 3D v 2D s pomočjo funkcije *gluOrtho2D* **[**4**].** Pri izrisu takšnih scen je potrebno biti pazljiv, kdaj jo izrišemo, saj se lahko drugače želen 2D izris popači. To pomeni, da če izrišemo konzolo prehitro, bo prišlo do prekrivanja 2D in 3D scene, zato je najbolje, da se scene, ki so v ospredju, izrisujejo kasneje.

4.4.2 Prikaz znakov

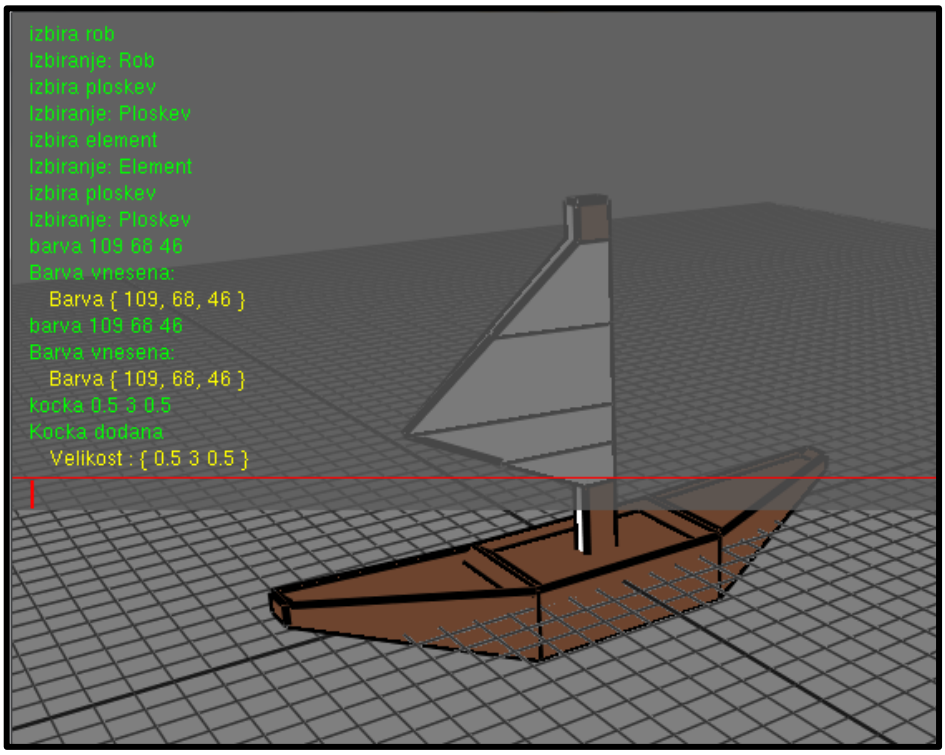
Prikaz znakov na zaslonu nam omogoča funkcija knjižnice GLUT *glutBitmapCharacter*, ki kot vhodni parameter sprejme font, s katerim bodo znaki prikazani, ter znak, ki ga želimo prikazati. Funkciji predhodno s pomočjo translacije določimo mesto, kjer se začne izpis, nato se pa nadaljnji razmiki med znaki samostojno izračunajo. Pri tem moramo omeniti, da je nabor pisav, ki jih ima knjižnica GLUT, zelo omejen, saj jih nima veliko.

Fonti, omogočeni v knjižnici GLUT (ang. *GLUT fonts*), so [3]:

- *fixed8by13* pisava fiksne širine, pri kateri vsak znak ustreza pravokotniku velikosti 8 pikslov širine in 13 višine;
- *fixed9by15* pisava fiksne širine, pri kateri vsak znak ustreza pravokotniku velikosti 9 pikslov širine in 15 višine;
- *Times New Roman* (velikosti 10 in 24);
- *Helvetica* (velikosti 10, 12 in 18).

4.4.3 Prikaz konzole

Prikaz konzole je, kljub prej omenjeni možnosti, da jo lahko skrijemo, lahko moteč, saj zaseda del modelirnega površja. Modelirna površina bi se sicer lahko ob vklopu spremenila in bi bila konzola ter ostali del programa prikazana v ločenih oknih, vendar bi bilo to prav tako moteče, saj je pri modeliranju najbolje, da je modelirna površina čim večja, s tem pa je večja tudi preglednost nad modelom. Zato je naša rešitev bila, da konzolo preprosto naredimo prosojno in lahko tudi v času, ko je konzola vklopljena in vanjo vpisujemo ukaze, vidimo v ozadju vso modelirno površino. Prosojnost v vmesniku OpenGL dobimo tako, da jo omogočimo z ukazom *glEnable(GL_BLEND)* [4]. Ko vključimo možnost za prosojnost, moramo določiti še, s katerim načinom želimo prosojnost doseči. To pa storimo z uporabo funkcije *glBlendFunc* [4], ki za vhod sprejme dva parametra. Le-ta povesta, kako se izračuna izvorni (ang. *source*) in ciljni (ang. *destination*) faktor prosojnosti barv (rdeča, zelena, modra in alfa), glej sliko 4.



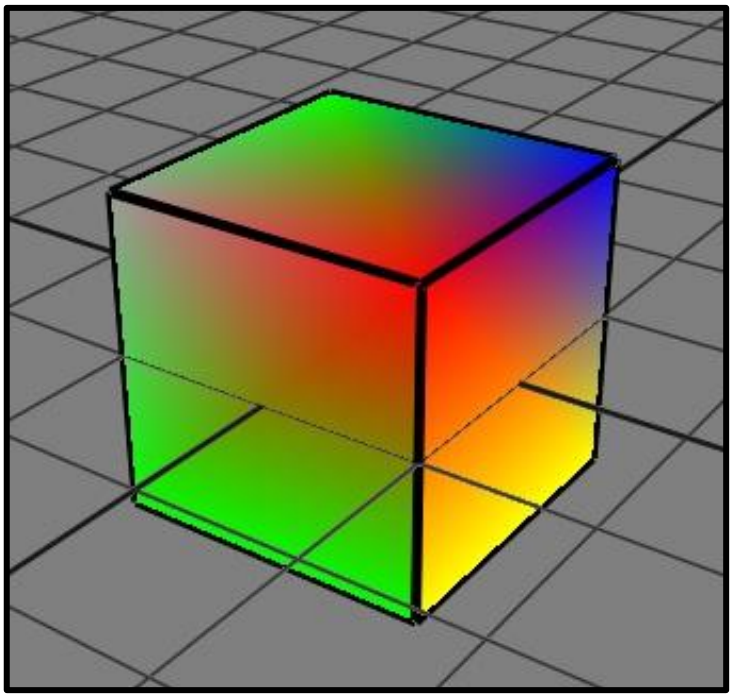
Slika 4: Prikaz konzole

4.4.4 Delovanje konzole

Konzole delujejo bolj ali manj vse po istem principu in to tako, da uporabnik vnese ukaz, konzola ga preveri in, če je pravilen, izvrši. V nasprotnem primeru vrne sporočilo in izpiše napako. Tako deluje tudi konzola tega modelirnika. V njej je prav tako omogočena možnost spreminjanja še ne vnesenega ukaza ter sklicevanja na zgodovino vnesenih ukazov. Vsebuje razčlenjevalnik ukazov, ki ukaz sprejme in izvrši želeno funkcijo. V kolikor ukaz vsebuje dodatne parametre, te prebere in jih sporoči ustrezni funkciji. Pri tem ni pomembno, ali so ukazi napisani z velikimi ali malimi črkami, saj jih bo razčlenjevalnik ukazov vedno zaznal kot pravilne. V kolikor je ukaz vnesen napačno, konzola uporabnika o napaki opozori, in z bolj izstopajočo barvo izpiše sporočilo z obrazložitvijo napake, glej sliko 3. Konzola izpisuje tudi pravilne ukaze in sporočila, ki so namenjena za pomoč ali pojasnilo uporabniku:

Ukazi konzole:

- *celozaslonski nacin*: program raztegne čez celoten zaslon;
- *okenski nacin*: program spremeni v okenski način;
- *izhod*: izklop programa;
- *kocka*: doda kocko s privzetimi vrednosti (velikost 1, središče {0,0,0}, barva {255, 255, 255}):
    - *parametri*: doda kocko glede na število parametrov (parametri so ločeni z presledkom): najprej lahko napišemo številko za velikost kocke, to številko pa lahko zamenjamo s tremi, ki predstavljajo smeri *x*, *y* in *z* (velikost ne sme biti enaka nič), nadaljnje tri številke predstavljajo položaj središča kocke, zadnje tri pa predstavljajo barvo kocke;
- *izbira*: izpiše, kateri del objekta trenutno izbiramo (točka, rob, ploskev, objekt):
    - *oglisce*: nastavi izbiranje na izbiro točk;
    - *rob*: nastavi izbiranje na izbiro robov;
    - *ploskev*: nastavi izbiranje na izbiro ploskev;
    - *objekt*: nastavi izbiranje na izbiro posameznega objekta;
- *barva r g b*: trenutno izbran del objekta prebarva z barvo določeno z *r g b*, pri tem modelirnik omogoča tudi mešanje barv, glej sliko 5.



Slika 5: Obarvan objekt

4.5 Knjižnica Vektor3D

Pri modelirnikih in na splošno v 3D grafiki se večina matematičnih izračunov vrti okoli 3D točk in 3D vektorjev. Za računanje le teh pa nimamo nobene prave podpore in je za to potrebno veliko redundantne kode. Da bi se to zmanjšalo, smo izdelali knjižnico Vektor3D. Knjižnica nam omogoča uporabo novega tipa vektor3D, sestavljenega iz polja dolžine 3 (*x*, *y*, *z*) tipa »float«. Knjižnica prav tako vsebuje prekrite operatorje (izpis 3) za vse računske operacije, ki jih lahko izvršimo nad vektorji.

Knjižnica Vektor3D pa ne omogoča le računanja med vektorji, temveč tudi izračun smeri, inverzne smeri in dolžine vektorjev. Ena izmed pomembnejših funkcij, vsebovanih v knjižnici, je tudi rotacija vektorja, saj rotacija v 3D prostoru predstavlja kar veliko težav, še posebej pa rotacija okoli poljubne osi.

```
      // seštevanje vektorjev
      vektor3D operator+ (vektor3D &v) const { return
vektor3D(w[0]+v.x(),w[1]+v.y(),w[2]+v.z());}
      //odštevanje vektoprjev
      vektor3D operator- (vektor3D &v) const { return vektor3D(w[0]-
v.x(),w[1]-v.y(),w[2]-v.z());}
      //množenje vektorja s skalarjem
      vektor3D operator* (float x) const {return vektor3D(w[0]*x, w[1]*x,
w[2]*x);}
      //skalarni produkt vektorjev
      float operator* (const vektor3D &v) const {return
(w[0]*v.x()+w[1]*v.y()+w[2]*v.z());}
      //vektorski produkt
      vektor3D operator ^ (vektor3D &v) const {return vektor3D(y()*v.z()-
z()*v.y(), z()*v.x()-x()*v.z(), x()*v.y()-y()*v.x());}
      //preverjanje enakosti vektorjev
      bool operator==(const vektor3D &op2) const {return (w[0] ==
op2.w[0] && w[1] == op2.w[1] && w[2] == op2.w[2]);}
```

Izpis 3: Prekrivanje operatorjev

4.5.1 3D Rotacija vektorja

Kot je bilo omenjeno že zgoraj, se za rotacijo uporabljajo rotacijske matrike. Osnovne matrike za rotacijo objekta okoli osi so najbolj učinkovite, kadar je objekt rotacije v koordinatnem izhodišču. To pa bi pomenilo, da moramo vsak objekt, ki se ne nahaja v koordinatnem izhodišču, pred rotacijo premestiti v izhodišče, rotirati in nato premestiti nazaj na prvotno mesto. Prav zaradi tega smo se odločili za rotacijsko matriko (4.7), ki ni vezana na izhodišče, temveč ji lahko določimo poljuben smerni vektor, okoli katerega se bo rotacija izvedla.

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \tag{4.7}$$

Pri tem so A do I izrazi za naslednje izračune (x, y, z so vrednosti poljubnega smernega vektorja):

- $A = (1-\cos\alpha)x^2 + \cos\alpha$
- $B = (1-\cos\alpha)xy + \cos\alpha$
- $C = (1-\cos\alpha)xz - \sin\alpha x$

- $D = (1-\cos\alpha)xy - \sin\alpha z$
- $E = (1-\cos\alpha)y^2 + \cos\alpha$
- $F = (1-\cos\alpha)yz + \sin\alpha x$
- $G = (1-\cos\alpha)xz + \sin\alpha y$
- $H = (1-\cos\alpha) - \sin\alpha x$
- $I = (1-\cos\alpha) - z^2 + \cos\alpha$

Pri vnosu kota smo morali biti pazljivi, saj vmesnik OpenGL ne sprejema stopinjskih podatkov, temveč podatke v radianih. Tako smo morali vsak stopinjski podatek s pomočjo enačbe 4.8 pretvoriti v radiane.

$$\alpha' = \frac{\pi \cdot \alpha}{180} \tag{4.8}$$

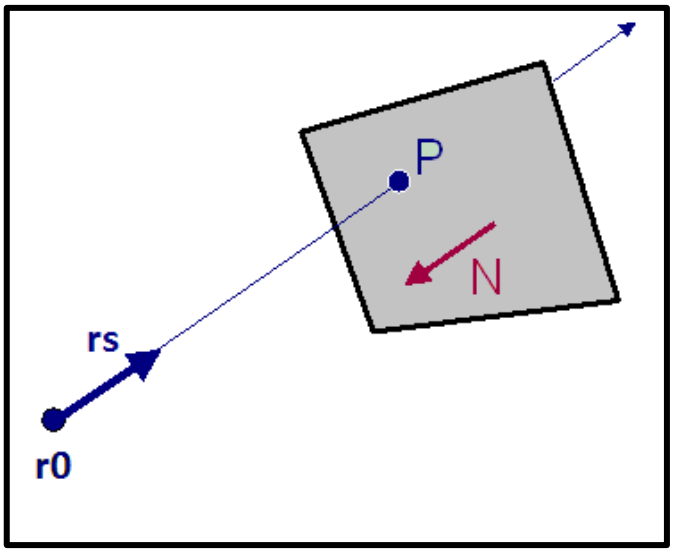
4.6 Vsebnostni test

Kot že omenjeno, ima vsebnostni test pomembno vlogo v mnogih programih. Že samo za klik gumba je potreben vsebnostni test, čeprav zelo preprost. Pri grafičnih programih pa se lahko pojavi veliko problemov.

Vsebnosti testi se uporabljajo pri geometrijskih problemih, kjer želimo vedeti, ali je točka v mnogokotniku, objektu ali izven njega. Obstajajo vsebnostni testi, ki so namenjeni ravnini in tisti, ki so namenjeni za 3D prostor. K vsebnostnim testom ravnine spadajo:

- metoda z vsoto notranjih kotov,
- metoda enakih predznakov,
- metoda baricentričnih koordinat.

V 3D prostoru se večina vsebnostnih testov izvaja s sledenjem žarkom (ang. *ray tracing*), glej sliko 6. Žarek je sestavljen iz začetne točke, ki jo določimo s klikom miške in iz nje pošljemo žarek. Nato lahko preverjamo, kolikokrat se objekt in žarek sekata, ali pa s pomočjo matematičnih izračunov ugotovimo, če žarek objekt sploh seka in kje ga seka.



Slika 6: Vsebnostni test, [1]

4.6.1 3D vsebnostni test s pomočjo očrtanega kvadra

Vsebnostni test s pomočjo očrtanega kvadra je preverjanje, ali se nek predmet nahaja v očrtanem kvadru. Ta očrtani kvader zajema ves objekt in ga lahko opišemo z diagonalo kvadra. Na tak način lahko z malo primerjave med koordinatami ugotovimo, ali se zgodi interakcija med očrtanim kvadrom in poljubnim objektom.

Metoda očrtanega kvadra je zelo priljubljena pri računalniških igrah za detekcijo trkov (ang. *colision detection*), saj je hitra in učinkovita. V računalniških igrah se sicer večinoma preverja interakcija med več očrtanimi kvadri, kar pa v našem modelirniku ne pride v poštev. V modelirniku moramo preveriti, ali žarek, ki ga določimo s klikom miške, zadane objekt ali ne. Čeprav to ne poda točke, v kateri se zgodi interakcija med objektom in žarkom, pa nam omeji število objektov, ki jih moramo preveriti.

Očrtani kvader, ki smo ga v ta modelirnik implementirali, je narejen po Brian Smitsovem algoritmu, ki deluje na principu »plošč« (ang. *slabs*). Plošča v tem primeru predstavlja prostor med dvema paralelnima ravninama, interakcija večjega števila takšnih plošč pa definira očrtani kvader. Gre torej za metodo, ki preverja interakcije med pari plošč z žarkom [6], [7].

V naši aplikaciji je vsebnosti test še dodatno optimiziran, in sicer smo zmanjšali število preverjan in zamenjali operacije deljenja z množenjem. Vsebnostnemu testu kot vhod

podamo začetno točko žarka in njegovo inverzno smer. Prva od optimizacij je preverjanje predznakov smeri žarka, saj s tem odpravimo potrebo po pogojnih stavkih. Druga optimizacija je nadomestitev deljenja s smerjo žarka, z množenjem inverzne smeri, kar pa še dodatno odpravi možnost, da bi prišlo do deljenja z 0, (izpis 4).

Vsebnostni test, kot je omenjeno zgoraj, vsako od osi preveri z očrtanim kvadrom in žarkom. To naredimo tako, da za vsako os izračunamo dolžino najbližje in najbolj oddaljene točke, ki jo seka žarek. Nato te vrednosti primerjamo med seboj in poiščemo, katera od točk je najbližja, najbližjo izmed teh pa glede na podan interval (v našem primeru je to interval [-1000, 1000] in predstavlja celoten prostor) označimo kot interakcijo z očrtanim kvadrom.

```
bool Objekti::preveri(vektor3D r0, vektor3D ris)
{
	bool sign[] = {ris.x()<0,ris.y()<0,ris.z()<0};
	float tmin, tmax, tymin, tymax, tzmin, tzmax;

	tmin = (minmax[sign[0]][0] - r0.x()) * ris.x();
	tmax = (minmax[1-sign[0]][0] - r0.x()) * ris.x();
	tymin = (minmax[sign[1]][1] - r0.y()) * ris.y();
	tymax = (minmax[1-sign[1]][1] - r0.y()) * ris.y();

	if ( (tmin > tymax) || (tymin > tmax) )
		return false;

	if (tymin > tmin)
		tmin = tymin;

	if (tymax < tmax)
		tmax = tymax;

	tzmin = (minmax[sign[2]][2] - r0.z()) * ris.z();
	tzmax = (minmax[1-sign[2]][2] - r0.z()) * ris.z();

	if ( (tmin > tzmax) || (tzmin > tmax) )
		return false;

	if (tzmin > tmin)
		tmin = tzmin;

	if (tzmax < tmax)
		tmax = tzmax;

	return ( (tmin < 1000) && (tmax > -1000) );
}
```

Izpis 4: Optimiziran Smitsov algoritem

4.6.2 Vsebnostni test za detekcijo posameznih delov objekta

V modelirnikih pa potrebujemo še vsebnostni test, ki je zmožen zaznati objekt, ploskve, robove in točke. Zato smo v tej nalogi poleg zgoraj omenjenega optimiziranega Smitsovega algoritma, s katerim smo zaznavali objekte, uporabili še hierarhično sestavljen vsebnostni test, ki izračuna interakcijsko točko med zaznanim objektom in žarkom. Interakcijska točka pa nam pomaga pri zaznavanju ploskev, robov in oglišč.

Hierarhija vsebnostnega testa:

- **Interakcija med žarkom in ploskvijo.**

  Iskanje interakcijske točke med žarkom in ravnino ploskve poteka čez vse ploskve objekta. Najprej preverimo, če obstaja interakcija med žarkom in ravnino, na kateri leži ploskev. Pri tem si pomagamo z enačbo 4.9, ki jo dobimo tako, da vstavimo enačbo vektorja v enačbo ravnine in izpostavimo faktor, ki nam predstavlja razdaljo med izhodiščem žarka in interakcijsko točko. Dobljen rezultat pomnožimo z izhodiščem žarka, pri čemer dobimo interakcijsko točko [8].

$$t = \frac{-d - \mathbf{r}_0 \cdot \mathbf{n}}{\mathbf{r}_s \cdot \mathbf{n}} \tag{4.9}$$

  Interakcijska točka se nahaja na ravnini, določeni s ploskvijo objekta, preveriti pa še moramo, če se nahaja na sami ploskvi objekta. To preverimo z vsebnostnim testom interakcija točka-ploskev, glej izpis 5.

```
//pk - točka interakcije
//p1, p2, p4 točke ploskve
vektor3D t0 = pk-p1;
vektor3D t1 = p2-p1;
vektor3D t2 = p4-p1;

float t01 = t0*t1;
float t02 = t0*t2;

if(0<=t01 && t01<=t1*t1 && 0<=t02 && t02<=t2*t2)
{

      if((r0-pk).dolzina()<dol)
    {
            //shranimo podatke o interakciji (indeks ploskve in
točko interakcije)
            p = pk;
            dol = (r0-pk).dolzina();
            index = i;
      }
}
```

Izpis 5: Vsebnostni test točka-ploskev

- **Izbiranje oglišča.**

  Preverjanje, ali je bilo izbrano oglišče, izračunamo s pomočjo interakcijske točke in ploskve, ki nam pove, katere točke so lahko oglišče. Naredimo ga tako, da izračunamo vektorje med interakcijsko točko in točkami ploskve. Vektorjem izračunamo dolžino in med njimi poiščemo najkrajšega in vrednost njegove dolžine primerjamo s faktorjem odstopanja. Če se nahaja znotraj tega faktorja, lahko oglišče izberemo.

- **Izbiranje roba.**

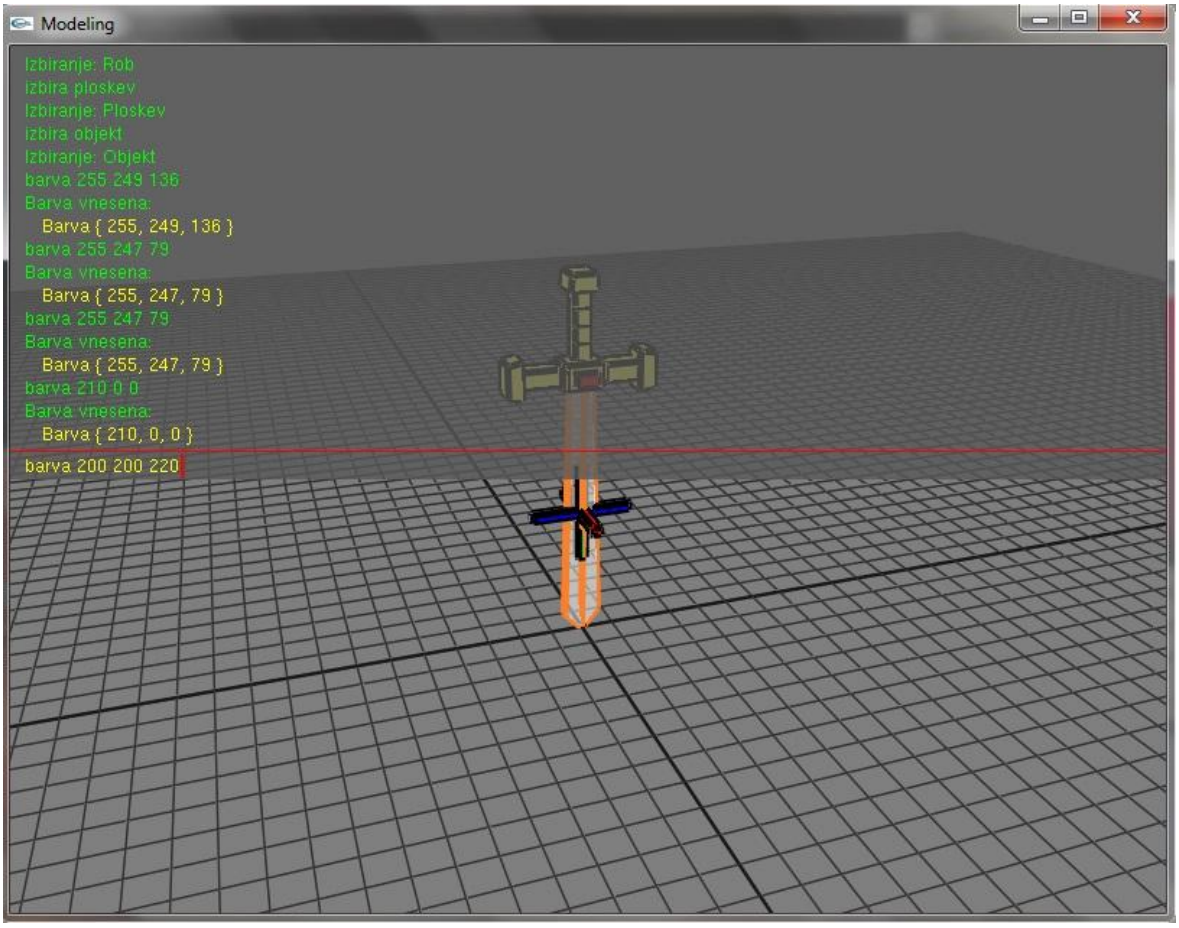
  Preverjanje, ali je bil izbran rob, se izvede na koncu vsebnostnega testa. Takrat imamo že znano oglišče, ki je najbližje interakcijski točki, kar nam omeji število robov za nadaljnje izbiranje. Izračunamo razdalje med interakcijsko točko in robovi ter primerjamo najkrajšo razdaljo med njimi s faktorjem odstopanja. Če se razdalja nahaja znotraj tega faktorja, ga izberemo.

5 REZULTATI

Danes obstaja veliko programov, ki uporabnikom pomagajo pri modeliranju različnih modelov. Z nekaj izmed teh smo se imeli priložnost srečati tudi mi in prav srečanje z njimi nam je dalo zamisel o izdelavi lastnega modelirnika.

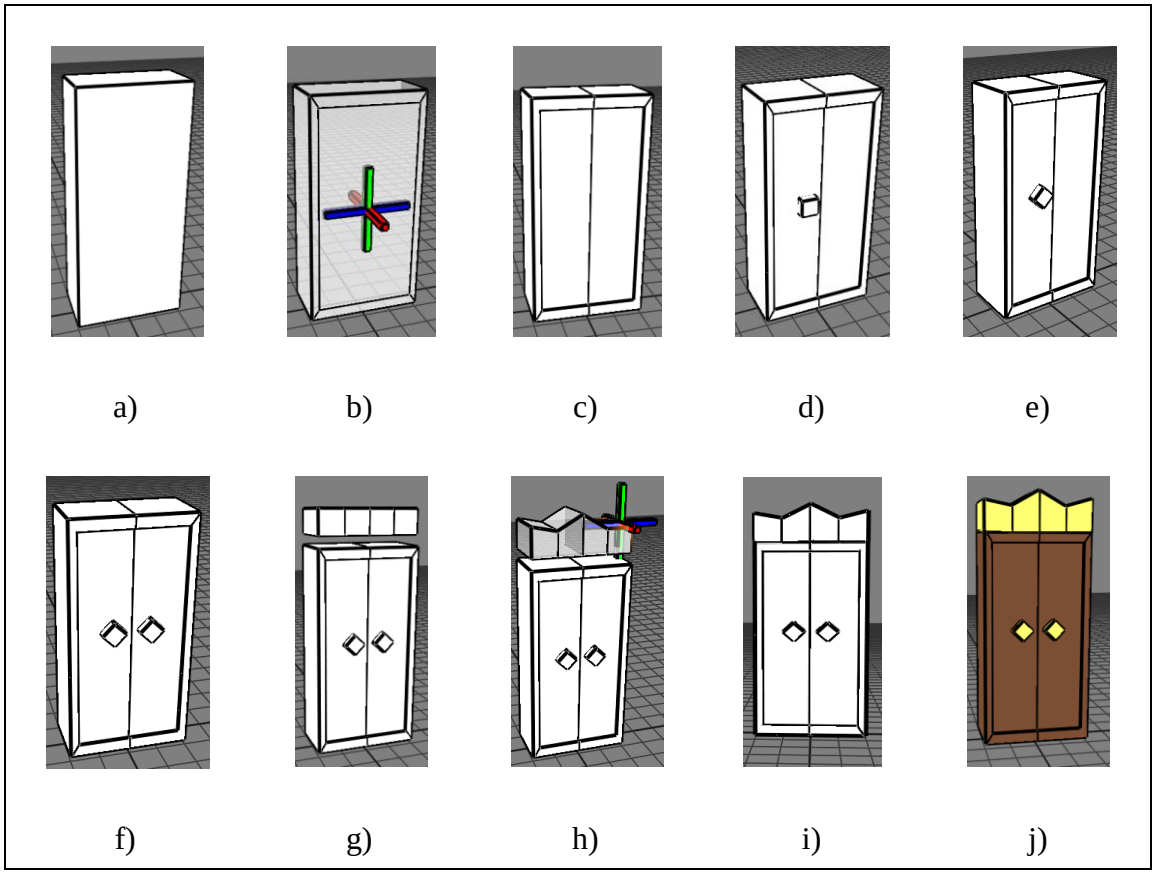
Cilj, ki smo si ga zadali, je bil izdelava modelirnika, ki bi bil vsaj približno tako kakovosten kot kateri izmed že obstoječih. Želeli smo implementirati čim več funkcij, ki bi uporabniku omogočale hitro in preprosto izdelavo zahtevnejšega modela.

Zastavljene cilje smo dosegli. Uspelo nam je vključiti metodo modeliranja s kvadrom, osnovne geometrijske transformacije, kopiranje objektov, deljenje in izvlečenje ploskev ter barvanje posameznih delov objekta. Kombinacija teh funkcij nam omogoča izdelavo najrazličnejših modelov, od preprostih do zahtevnejših.



Slika 7: Glavno okno z modelom in konzolo

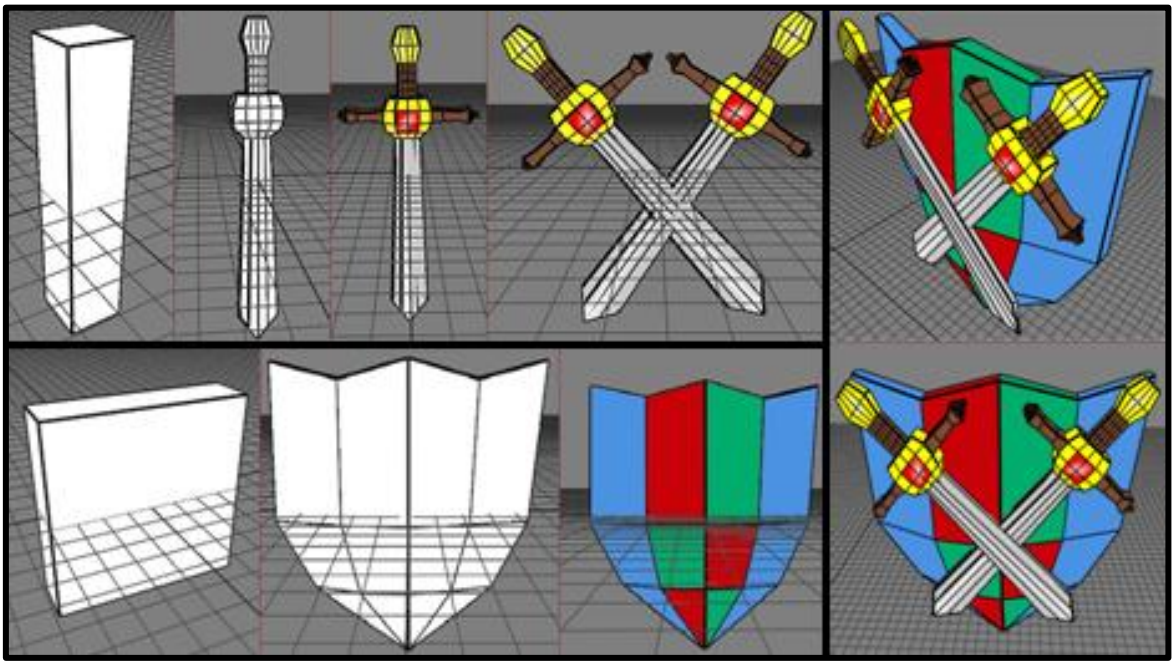
Modelirnik se ob zagonu odpre v enem oknu, na katerem se izriše mreža, ki nam je v pomoč pri izdelavi modela. Okno se privzeto zažene v okenskem načinu, lahko pa ga spremenimo na celozaslonski način. Da lahko začnemo z modeliranjem, najprej pritisnemo tipko *tab*, s katero odpremo konzolo, saj je, kot že omenjeno, potrebna za krmiljenje modelirnika, glej sliko 7. Z uporabo ukaza *kocka* na modelirno površino vstavimo poljuben kvader, ki bo predstavljal celoten model ali pa le sestavni del modela, ki ga želimo izdelati. S ponovnim pritiskom na tipko *tab* lahko konzolo skrijemo, da nas ne moti pri modeliranju. V kolikor velikost kvadra ne ustreza našim potrebam, ga s klikom miške označimo in skaliramo na želeno velikost. Če želimo, lahko z ukazom *izbira rob*, držanjem tipke *t* in klikom miške razdelimo ploskve vstavljenega kvadra. Z uporabo nabora funkcij, ki jih modelirnik omogoča, lahko nadaljujemo z izdelavo poljubnega modela. Za boljšo predstavo modeliranja z našim modelirnikom smo se odločili, da bomo podrobneje predstavili izdelavo modela omare (slika 8).



Slika 8: Izdelava modela omare

Tako kot že zgoraj omenjeno, najprej s pomočjo ukaza *kocka* dodamo kvader, ki nam bo predstavljal osnovno obliko modela omare (slika 8.a). Za izdelavo roba omare označimo sprednjo ploskev in s pomočjo uporabe funkcije izvlečenje ploskve ter skaliranje, sprednjo ploskev pomanjšamo ter premaknemo v model (slika 8.b). Pri izdelavi vrat omare smo uporabili razpolavljanje ploskev (slika 8.c). S ponovno uporabo ukaza *kocka* smo izdelali kvader, ki smo ga skalirali in s translacijo postavili na vratno krilo omare, kjer predstavlja kljuko (slika 8.d). Kljuko smo s funkcijo za rotacijo rotirali za 90° (slika 8.e). Kljuko smo prenesli na drugo vratno krilo, kar smo naredili tako, da smo s klikom miške izbrali kljuko, jo kopirali in premaknili na ustrezno mesto (slika 8.f). Modeliranje omare bi lahko tukaj zaključili, vendar bi bila preveč pusta, zato smo ji dodali še okrasek. Z ukazom *kocka* smo dodali nov objekt, ki predstavlja okrasek, in ga z razpolavljanjem ploskev razdelili na štiri enake dele (slika 8.g). Okrasek omare smo s pomočjo izbire robov in translacije oblikovali tako, da smo dobili obliko krone (slika 8.h). Okrasek smo premaknili na vrh omare (slika 8.i). Na koncu smo naš model še pobarvali (slika 8.j). Tako smo hitro in preprosto izdelali model omare.

Zgoraj smo prikazali, kako se z našim modelirnikom izdeluje preproste modele. Lahko pa z istim naborom funkcij izdelamo tudi zahtevnejše modele (slika 9).



Slika 9: Primer izdelave mečev in ščita

6 SKLEP

V nalogi se je izdelal modelirnik, ki je zmožen izdelovanja najrazličnejših modelov s kombiniranjem osnovnih geometrijskih transformacij in ostalih funkcij modelirnika.

Pri tej nalogi se je pridobilo predvsem veliko izkušenj tako s področja računalniške grafike, kot s področja vektorske algebre in analitične geometrije, sploh pri izdelavi vsebnostnih testov in pri njihovih optimizacijah. Optimizacije vsebnostnih testov in strukturirana hranjenja podatkov objektov pa so pripomogla k temu, da program ves čas deluje in se odziva realnočasovno, tudi pri večji količini podatkov.

Ker je program za modeliranje rezultat diplomskega dela, je težko pričakovati, da bi se lahko kosal z že obstoječimi modelirniki. Kljub temu, da naš modelirnik ni na povsem enakem nivoju z že obstoječimi modelirniki, pa ga imamo vseeno možnost nadgraditi do takšne mere, da bi jim lahko konkuriral. Nadgradimo ga lahko s posebnimi geometrijskimi transformacijami (zrcaljenje in strig), dodatnimi geometrijskimi telesi in z dodajanjem tekstur. Prav tako bi lahko razširili območje uporabe modelirnika z možnostjo izdelave animacij in okostij modela. Naš modelirnik je trenutno namenjen uporabnikom z manjšim predhodnim znanjem, saj je pregleden in preprost za uporabo, vseeno pa uporabnikom omogoča izdelati tudi zahtevnejši izdelek.

7 VIRI IN LITERATURA

[1] R. Funkhouser, *Ray casting, Princeton University*, COS 426, 2000, http://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/raycast/sld017.htm, (17. 8. 2011).

[2] N. Guid, *Računalniška grafika*, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, 2001.

[3] M. J. Kilgard, *The OpenGL Utility Toolkit (GLUT), Programming Interface*, Silicon Graphics, (1996), http://www.opengl.org/resources/libraries/glut/glut-3.spec.pdf, (11. 5. 2011).

[4] *OpenGL*, http://www.opengl.org/documentation/, (11. 5. 2011).

[5] *OpenGL*, http://www.opengl.org/resources/libraries/glut/, (11. 5. 2011).

[6] B. Smits, Efficiency issues for ray tracing, *Journal of Graphics Tools*, 3, (1998), 2, str. 1–14.

[7] B. Smits, Efficient bounding box intersection, *Ray tracing news*, 15, (2002), 1.

[8] *The Intersection Test*, http://www.vhml.org/theses/usher/node27.html, (17. 8. 2011)

[9] *Wikipedia: The Free Encyclopedia, Comparicon of OpenGL and Direct3D,* http://en.wikipedia.org/wiki/Comparison_of_OpenGL_and_Direct3D, (1. 9. 2011).

[10] *Wikipedia: The Free Encyclopedia, OpenGL*, http://en.wikipedia.org/wiki/OpenGL, (1. 9. 2011).

[11] *Wikipedia: The Free Encyclopedia*, *OpenGL*, http://sl.wikipedia.org/wiki/OpenGL, (1. 9. 2011).

8 PRILOGE

8.1 Naslov študenta

Grega Žlahtič

Spodnja korena 13

2241 Spodnji Duplek

8.2 Kratek življenjepis

Rojen: 10. 6. 1989

Šolanje: 1996─2004 Osnovna šola Duplek

2004─2008 Srednja elektro-računalniška šola Maribor,

smer elektrotehnik računalništva

2008─2011 Fakulteta za elektrotehniko računalništvo in informatiko,

smer Računalništvo in informacijske tehnologije

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Grega Žlahtič,

z vpisno številko E1012905,

sem avtor/-ica diplomskega dela z naslovom:

Preprosti geometrijski modelirnik

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)
  doc. dr. Simon Kolmanič
  in somentorstvom (naziv, ime in priimek)
  red. prof. dr. Nikola Guid
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 15. 9. 2011 Podpis avtorja/-ice: Grega Žlahtič

Univerza v Mariboru
Fakulteta za elektrotehniko,
računalništvo in informatiko

IZJAVA O USTREZNOSTI DIPLOMSKEGA DELA

Podpisani mentor doc. dr. Simon Kolmanič izjavljam, da je
(ime in priimek mentorja)

študent Grega Zahtila izdelal diplomsko
(ime in priimek študenta-tke)

delo z naslovom: Preprosti geometrijski modelirnik

(naslov diplomskega dela)

v skladu z odobreno temo diplomskega dela, Navodili o pripravi diplomskega dela in mojimi navodili.

Datum in kraj: 15. 9. 2011, Maribor

Podpis mentorja:

UNIVERZA V MARIBORU

Fakulteta za elektrotehniko, računalništvo in informatiko
(ime fakultete)

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUCNEGA DELA IN OBJAVI OSEBNIH PODATKOV AVTORJA

Ime in priimek avtorja (avtorice):	Grega Žlahtič
Vpisna številka:	E1012905
Študijski program:	UNI Računalništvo in informacijske tehnologije
Naslov zaključnega dela:	Preprosti geometrijski modelirnik
Mentor:	doc. dr. Simon Kolmanič
Somentor:	red. prof. dr. Nikola Guid

Podpisani-a Grega Žlahtič izjavljam, da sem za potrebe arhiviranja oddal-a elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah (Ur. l. RS, št. 16/2007) dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija zaključnega dela je istovetna elektronski verziji, ki sem jo oddal-a za objavo v Digitalno knjižnico Univerze v Mariboru. Podpisani-a izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zagovora, naslov zaključnega dela) na spletnih straneh in v publikacijah UM.

Kraj in datum: Maribor, 19. 9. 2011 Podpis avtorja (avtorice): Grega Žlahtič