

Interaktivna računalna grafika - dokumentacija za laboratorijske vježbe

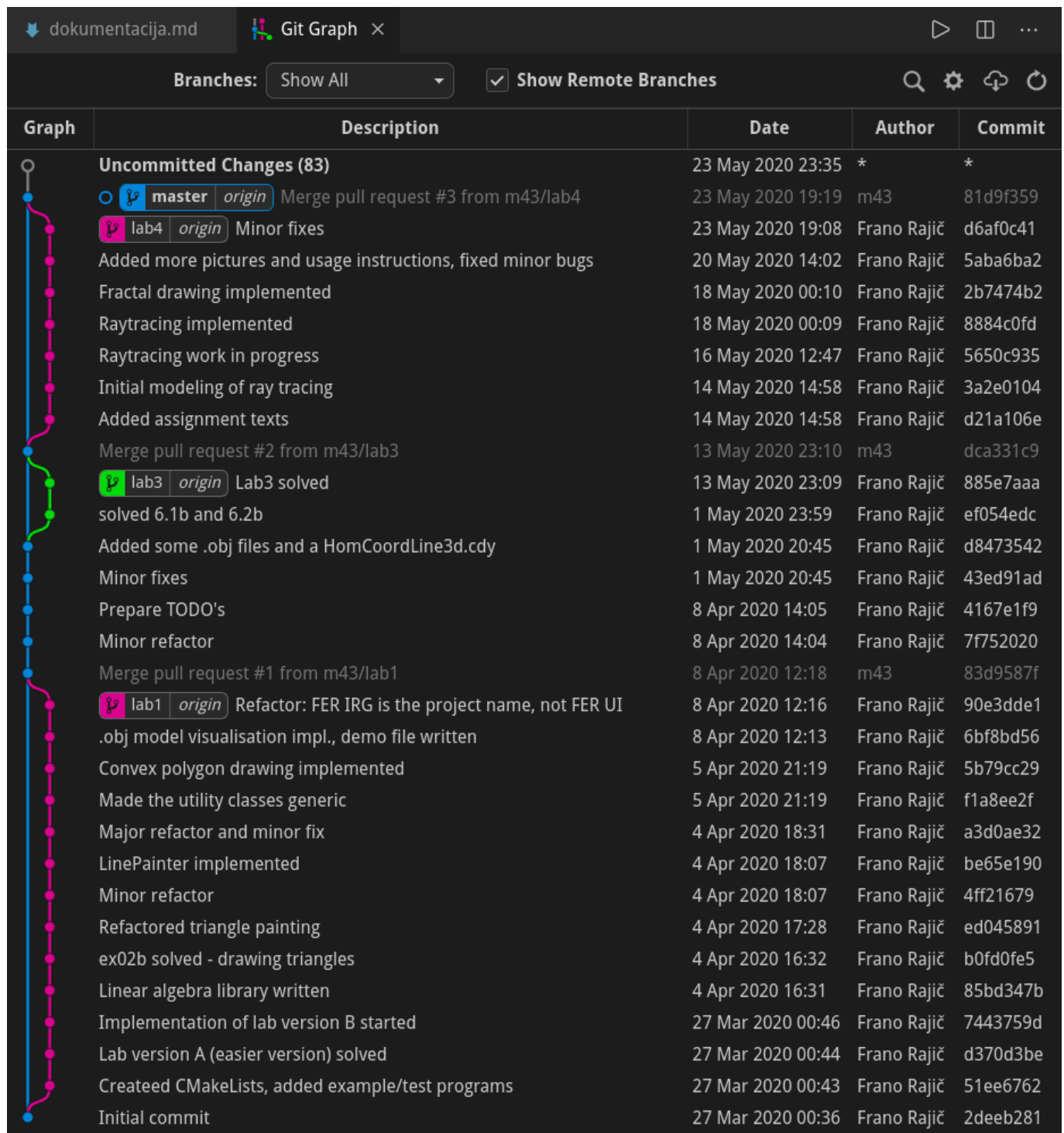
Ovaj dokument sadrži dokumentirana rješenja laboratorijskih vježbi iz predmeta Interaktivna računalna grafika akademske godine 2019./2020.. Kroz dokument su opisani sljedeći elementi implementiranog rješenja:

- upute za korištenje programa
- kratak opis programa/potprograma i međusobne povezanosti
- korištene strukture podataka
- promjene načinjene s obzirom na upute
- komentar rezultata (brzina, moguće promjene, problemi, nedostaci)

Postoje četiri ciklusa laboratorijskih vježbi i svaki sadrži dvije ili tri vježbe. Za svaki ciklus se neovisno o drugim ciklusima bira varijanta A ili varijanta B. Upute za pripadne vježbe se mogu pronaći u korijenu ovog repozitorija ([A_Labosi.pdf](#) i [B_Labosi.pdf](#)). Odabrao sam riješiti varijantu A prvog i drugog labosa te varijantu B trećeg i četvrtog. Pored rješenja predanih kroz sustav Ferko, dodatno sam riješio varijantu B prvog labosa.

Za jednostavno pokretanje svih programa, priložen je [CMakeLists.txt](#) u korijenu repozitorija. Izvršni programi su unutar te datoteke strukturirani po redoslijedu laboratorijskih vježbi.

Tijekom izrade svih rješenja, korišten je GIT za verzioniranje pripadnog repozitorija. Kako je izgledala povijest commitova je prikazno na slici ispod.



The image shows a Git Graph window for the file 'dokumentacija.md'. It displays a vertical timeline of commits on the left, with a blue line for 'master' and a pink line for 'lab4'. The main table lists the commit details.

Graph	Description	Date	Author	Commit
	Uncommitted Changes (83)	23 May 2020 23:35	*	*
○ master origin	Merge pull request #3 from m43/lab4	23 May 2020 19:19	m43	81d9f359
● lab4 origin	Minor fixes	23 May 2020 19:08	Frano Rajič	d6af0c41
	Added more pictures and usage instructions, fixed minor bugs	20 May 2020 14:02	Frano Rajič	5aba6ba2
	Fractal drawing implemented	18 May 2020 00:10	Frano Rajič	2b7474b2
	Raytracing implemented	18 May 2020 00:09	Frano Rajič	8884c0fd
	Raytracing work in progress	16 May 2020 12:47	Frano Rajič	5650c935
	Initial modeling of ray tracing	14 May 2020 14:58	Frano Rajič	3a2e0104
	Added assignment texts	14 May 2020 14:58	Frano Rajič	d21a106e
	Merge pull request #2 from m43/lab3	13 May 2020 23:10	m43	dca331c9
● lab3 origin	Lab3 solved	13 May 2020 23:09	Frano Rajič	885e7aaa
	solved 6.1b and 6.2b	1 May 2020 23:59	Frano Rajič	ef054edc
	Added some .obj files and a HomCoordLine3d.cdy	1 May 2020 20:45	Frano Rajič	d8473542
	Minor fixes	1 May 2020 20:45	Frano Rajič	43ed91ad
	Prepare TODO's	8 Apr 2020 14:05	Frano Rajič	4167e1f9
	Minor refactor	8 Apr 2020 14:04	Frano Rajič	7f752020
	Merge pull request #1 from m43/lab1	8 Apr 2020 12:18	m43	83d9587f
● lab1 origin	Refactor: FER IRG is the project name, not FER UI	8 Apr 2020 12:16	Frano Rajič	90e3dde1
	.obj model visualisation impl., demo file written	8 Apr 2020 12:13	Frano Rajič	6bf8bd56
	Convex polygon drawing implemented	5 Apr 2020 21:19	Frano Rajič	5b79cc29
	Made the utility classes generic	5 Apr 2020 21:19	Frano Rajič	f1a8ee2f
	Major refactor and minor fix	4 Apr 2020 18:31	Frano Rajič	a3d0ae32
	LinePainter implemented	4 Apr 2020 18:07	Frano Rajič	be65e190
	Minor refactor	4 Apr 2020 18:07	Frano Rajič	4ff21679
	Refactored triangle painting	4 Apr 2020 17:28	Frano Rajič	ed045891
	ex02b solved - drawing triangles	4 Apr 2020 16:32	Frano Rajič	b0fd0fe5
	Linear algebra library written	4 Apr 2020 16:31	Frano Rajič	85bd347b
	Implementation of lab version B started	27 Mar 2020 00:46	Frano Rajič	7443759d
	Lab version A (easier version) solved	27 Mar 2020 00:44	Frano Rajič	d370d3be
	Created CMakeLists, added example/test programs	27 Mar 2020 00:43	Frano Rajič	51ee6762
	Initial commit	27 Mar 2020 00:36	Frano Rajič	2deeb281

Prvi ciklus laboratorijskih vježbi - varijanta A

Prvi ciklus se bavi osnovnim operacijama u računalnoj grafici i crtanjem pravca odnosno linije. U nastavku su opisana rješenja vježbi 1A i 2A, te kratak komentar vježbi iz B inačice.

Vježba 1A - Osnovne operacije u računalnoj grafici

Prva vježba daje osnovne matematičke operacije koje se koriste u računalnoj grafici. To su na primjer skalarni produkt vektora, vektorski produkt vektora, množenje matrica. Za njihovo korištenje sam koristio **OpenGL Mathematics (GLM)** biblioteku preuzetu ovdje: <https://glm.g-truc.net/0.9.9/index.html>. U nastavku je kratak opis biblioteke preveden sa službenih stranica:

OpenGL Mathematics (GLM) je header only C++ matematička biblioteka za grafički softver zasnovana na specifikacijama OpenGL Shading Language (GLSL).

GLM pruža klase i funkcije dizajnirane i implementirane s istim konvencijama imenovanja i funkcionalnostima kao GLSL tako da svatko tko poznaje GLSL, može koristiti GLM i u programskom jeziku C++.

Ovaj projekt nije ograničen na GLSL značajke. Sustav proširenja, temeljen na konvencijama GLSL proširenja, pruža proširene mogućnosti: matrice transformacije, kvaternioni, pakiranje podataka, slučajni brojevi, šum itd.

Struktura rješenja je sljedeća:

```
src/ex01a
├── 1.cpp
├── 2.cpp
├── 3.cpp
src/utility
└── utilities.h
```

U mapi `utility` se općenito nalaze funkcionalnosti koje se dijele između različitih programa i različitih laboratorijskih vježbi. U ovoj vježbi, header only datoteka `utilities.h` sadrži funkcionalnost urednog prikazivanja GLM matrica te rješavanja sustava triju jednadžbi. Spomenute funkcionalnosti se nalaze u sljedeće dvije funkcije:

```
namespace ex_utilities {
    std::string pretty_print_matrix(const double *pSource, int rows, int
columns);
    glm::dvec3 solve_equation_system(glm::dvec3 a, glm::dvec3 b, glm::dvec3
c, glm::dvec3 t).;
}
```

Demonstracijski programi se nalaze u datotekama `1.cpp` `2.cpp` `3.cpp`. Prvi program `1.cpp` sadrži jednostavne primjere primjene biblioteke GLM za potrebu vektorskog i matičnog računa. `2.cpp` rješava uneseni sustav s tri nepoznanice. `3.cpp` je program koji prima podatke vrhovima trokuta (A, B i C) i točki T u 3D prostoru i zatim računa i ispisuje baricentrične koordinate točke T s obzirom na zadani trokut.

Za pokretanje programa u naredbenom retku, može se koristiti primjerice g++:

```
# 1.1
g++ -o 1a-1.out src/ex01a/1.cpp -Wall -lglut -lGL -lGLU
./1a-1.out

# 1.2
g++ -o 1a-2.out src/ex01a/2.cpp -Wall -lglut -lGL -lGLU
./1a-2.out
```

```
# 1.3
g++ -o 1a-3.out src/ex01a/3.cpp -Wall -lglut -lGL -lGLU
./1a-3.out
```

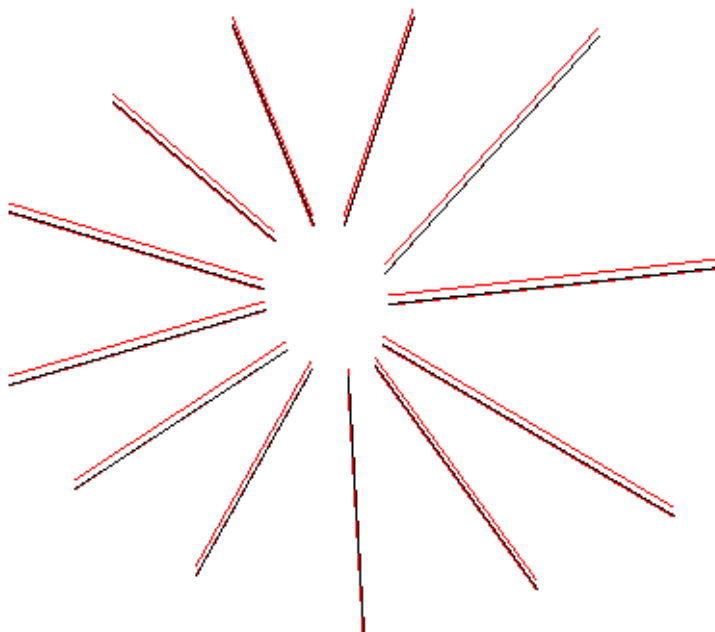
Vježba 2A - Pravic, linija

Druga vježba je iscrtavanje linije Bresenhamovim postupkom. Ova vježba daje temeljni postupak pretvorbe iz kontinuiranog oblika u diskretni oblik potreban prilikom prikaza na zaslonu.

Rješenje je strukturirano u dvije datoteke kao u što je prikazano nastavku:

```
src/ex02a
└─ 1.cpp
src/utility
└─ utilities.h
```

1.cpp je demonstracijski program koji omogućuje crtanje linije Bresenhamovim postupkom. Prvi klik miša definira početak linije, a drugi klik kraj linija. Nakon drugog klika, linija se iscrtava crnom bojom. Iznad nje se iscrtava linija crvene boje koja predstavlja liniju nacrtanu korištenjem gotovih funkcionalnosti OpenGL biblioteke. Konkretna implementacija Bresenhamovog postupka se nalazi u utility datoteci **utilities.h**. Ispod je slika pokrenutog programa nakon nekoliko nacrtanih linija.



Za pokretanje programa mogu se koristiti sljedeće naredbe:

```
g++ -o 2a.out src/ex02a/1.cpp -Wall -lglut -lGL -lGLU
./2a.out
```

B inačica

Za rješenja prvog ciklusa B inačice prilažem upute za pokretanje te par slika.

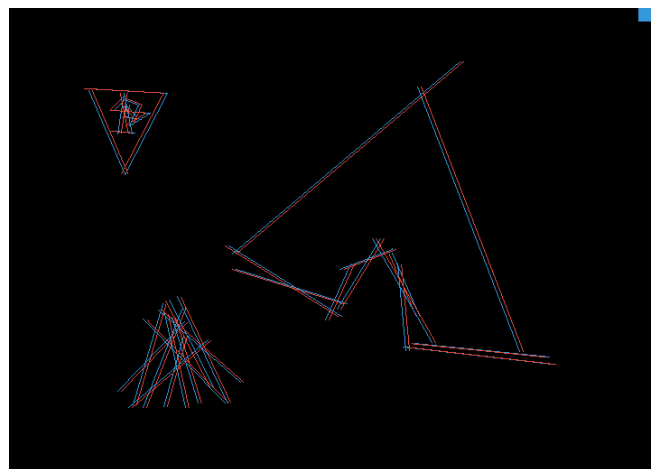
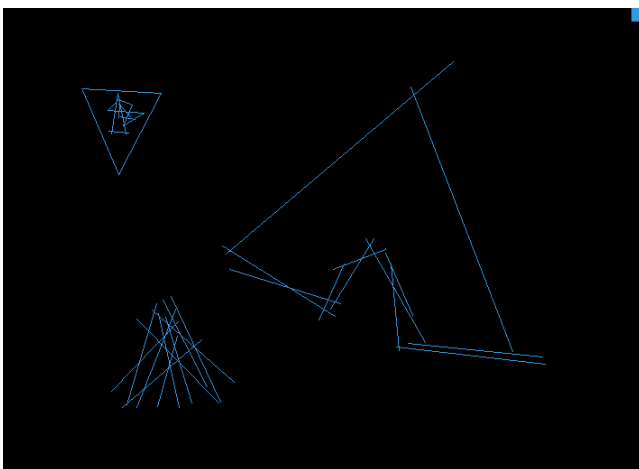
```
# linalg-demo-01
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/linalg/demo_01.cpp -o linalg-demo-01.out
./linalg-demo-01.out
#linalg-demo-02
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/linalg/demo_02.cpp -o linalg-demo-02.out
./linalg-demo-02.out

# 1.1B
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex01b/1.cpp -o ex01b-1.out
./ex01b-1.out
# 1.2B
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex01b/2.cpp -o ex01b-2.out
./ex01b-2.out
# in:
# 1 1 1 6
# -1 -2 1 -2
# 2 1 3 13
# expected output:
# [
# |      1 |
# |      2 |
# |      3 |
# |      ]
#
# 1.3B
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex01b/3.cpp -o ex01b-3.out
./ex01b-3.out
# in:
# 1 1 0
# 6 11 2
# 11 1 0
# 6 6 1
# out:
# (0.25 ,0.5 ,0.25)
# in:
# 1 0 0
# 5 0 0
# 3 8 0
```

```
# 3 4 0
# out:
# (0.25 ,0.25 ,0.5)

# 2B triangles
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex02b/triangle_demo.cpp -o ex02b.out
./ex02b.out

# 3B lines
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex03b/line_demo.cpp -o ex03b.out
./ex03b.out
```



Drugi ciklus laboratorijskih vježbi - varijanta A

Drugi ciklus se bavi crtanjem i bojanjem konveksnih poligona te učitavanjem i prikazivanjem .obj datoteka. U nastavku su opisana rješenja vježbi 4B i 4A.

Vježba 3A/4B - Crtanje i popunjavanje poligona

U trećoj vježbi upotrebljava se algoritam za popunjavanje konveksnog poligona i obavljaju se neka osnovna geometrijska izračunavanja prilikom ispitivanja odnosa točke i poligona u prostoru. Popunjavanje poligona temelji se na izračunavanju sjecišta pravaca. Kako je vježba 4 inačice B proširena vježba 3A funkcionalnošću, tako sam zapravo implementirano rješenje izveo slijedeći uputu B inačice.

Struktura implementiranog rješenja je sljedeća:

```
./src/ex04b/  
├─ convex_polygon_data_model.h  
├─ convex_polygon_demo.cpp  
└─ convex_polygonPainter.h  
./src/utility/  
├─ color.h  
├─ edge.h  
├─ line.h  
├─ point.h  
├─ polygon.h  
├─ triangle.h  
└─ utilities.h
```

Ova vježba je prva u kojoj sam implementaciju odvojio na tri različite cjeline. Prva je demonstracijski program `convex_polygon_demo.cpp` odnosno njegova metoda `main` kao ulazna točka u program. Brine se o stvaranju i inicijaliziranju prozora, te ga poveže s crtačem. Drugi dio je crtač `convex_polygonPainter.h` odnosno razred `ConvexPolygonPainter` koji se bavi iscrtavanjem sadržaja prozora, razvlačenjem prozora i obradom događaja vezanih uz tipkovnicu i miš. Crtača sam u ovoj vježbi nazvao Painter, u pojedinim narednim će dobiti naziv Renderer. U nastavku je isječak koda u kojem se funkcija `main` povezuje na crtač:

```
glutDisplayFunc(ConvexPolygonPainter::display);  
glutReshapeFunc(ConvexPolygonPainter::reshape);  
glutKeyboardFunc(ConvexPolygonPainter::keyPressed);  
glutMouseFunc(ConvexPolygonPainter::mousePressedOrReleased);  
glutMotionFunc(ConvexPolygonPainter::mouseMovedOrDragged);  
glutPassiveMotionFunc(ConvexPolygonPainter::mouseMovedOrDragged);
```

Treća cjelina koja čini implementaciju se bavi pohranom podataka potrebnih za iscrtavanje. Nalazi se u datoteci `convex_polygon_data_model.h` u kojoj je modelirana razredom `ConvexPolygonDataModel`. Do kraja laboratorijskih vježbi koristim konvenciju imenovanja ovog razreda DataModel-om. Za ovu vježbu je DataModel koristio sljedeće članske varijable kako bi upamtio potrebne informacije za crtanje:

```
Color foregroundColor_  
Color backgroundColorWithoutConvexityCheck_  
Color backgroundColorWithConvexityCheck_  
ConvexPolygonState state_ = DRAWING;
```

```
Polygon currentPolygon_;  
Point<int> lastMousePosition_ = Point<int>(0, 0);  
int width_;  
int height_;  
bool pFlag_{};  
bool kFlag_{};
```

Direktorij `./src/utility` i ovaj put sadrži dijeljene funkcionalnosti. Primjerice, razred `Polygon` se nalazi u `./src/utility/polygon.h` i koristi se za pohranu informacija o poligonu i računanje svojstava poligona kao što je konveksnost.

Za pokrenuti vježbu, mogu se izvesti sljedeće naredbe:

```
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw  
./src/ex04b/convex_polygon_demo.cpp -o 4b.out  
./4b.out
```

U programu se klikom miša dodaju vrhovi poligona i istovremeno se crta poligon. Kako se miš povlači preko zaslona, tako se pokazuje poligon koji bi bio rezultat klika mišom na tu točku.

Pritiskom tipke "k" se pozadina mijenja u zelenu i uključuje uvjet konveksnosti te se više ne mogu dodavati vrhovi poligona ako bi poligon dodavanjem tog vrha prestao biti konveksan. Ako je poligon prije pritiska tipke "k" već bio konveksan, onda se više ne može dodati niti jedan vrh u ovom načinu crtanja. Ponovnim pritiskom tipke, ovaj uvjet konveksnosti se uklanja.

Pritiskom tipke "p" se uključuje odnosno isključuje bojanje poligona. Poligoni koji nisu konveksni neće biti dobro obojeni.

Pritiskom tipke "n" se mijenja stanje iz dodavanje točaka poligona u ispitivanje položaja pritisnute lokacije. Pritisne li se mišom izvan konveksnog poligona, ispisuje se u konzolu poruka da ta točka ne pripada poligonu *Point (580, 73) is inside of polygon.*, odnosno da pripada ako je točka unutar poligona *Point (590, 73) is outside of polygon.* Ako poligon nije konveksan ispisuje se poruka *Cannot determine relationship between point and polygon if the polygon is concave.* Za ovaj način rada moraju postojati barem 3 točke, odnosno mora postojati neki poligon. Ponovnim pritiskom tipke "n" sve kreće ispočetka, odnosno iznova počinje definiranje točaka poligona.

Vježba 4A - Ravnina i 3D tijela

U četvrtoj vježbi potrebno je izgraditi topološku strukturu tijela i ispitati odnos točke i tijela.

Rješenje je strukturirano na sljedeći način:

```
./src/ex04a  
├─ 3d_object_data_model.h  
├─ 3d_object_demo.cpp  
└─ 3d_object_renderer.h  
./src/utility/  
├─ color.h
```



```
├─ edge.h
├─ line.h
├─ object_model.h
├─ point.h
├─ polygon.h
├─ triangle.h
└─ utilities.h
```

```
./src/objects/dragon.obj 0.3 0.3 0.3 quit
```

Za pokretanje programa, mogu se koristiti naredbe:

```
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex04a/3d_object_demo.cpp -o 4a.out
./4a.out
```

Po pokretanju programa se traži unos putanje do .obj datoteke koju je potrebno iscrtati, npr

`./src/objects/dragon.obj`. Zatim se ispisuju informacije o učitanoj objektu i traži unos točaka za koje je potrebno ispitati jesu li unutar objekta i konačno se iscrta objekt na ekranu. Objekt se može rotirati gore-dolje odnosno lijevo-desno pomoću kursora tipki i može se pritisnuti tipka "f" da bi se isključilo odnosno uključilo bojanje objekta. Primjer nekih unosa izgleda ovako:

```
./src/objects/kocka.obj
0 0 0
-0.25 -0.25 -0.25
0.62 -0.71 0.89
0.99 0.99 0.99
-0.99 -0.99 -0.99
0.99 0.99 -0.99
0 -1 -1
-1 -1 -1
1 1 1
-0.3 0.9 0.3
0.3 0.9 -0.3
2 2 2
3 3 3
-2 -2 -2
-2 2 -2
-1.1 -1.1 -1.1
1 1 1.0001
1 1 1.0000001
quit

./src/objects/dragon.obj
0.3 0.3 0.3
quit

./src/objects/bird.obj
```

```
0.3 0.3 0.3
quit
```

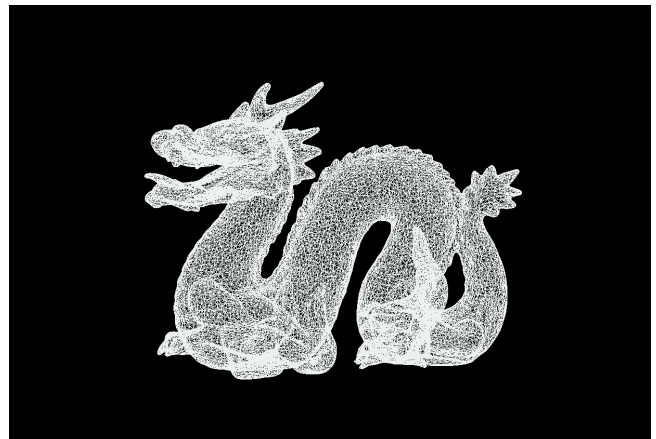
Primjer mogućeg ispisa je sljedeći:

```
*****
***** OBJECT MODEL DEMO *****
*****
Please enter path to file:
./src/objects/dragon.obj
Loading file "./src/objects/dragon.obj"...
File loaded. Found 25542 vertices and 50000 faces.
Object normalization performed.
xmin: -0.500000
xmax: 0.500000
ymin: -0.352505
ymax: 0.352505
zmin: -0.223584
zmax: 0.223584
Scaling factor 2
Object center: (0.000000,0.000000,0.000000)

*****
***** LINE TESTING *****
*****
NOTE: This program assumes that the user wants to enter a test point in the
coordinate system that is the result of all the transformations applied to
the original object (aka after translating and scaling).

Please specify the coordinates of the test point as three numbers separated
by spaces(all in range [-1,1]) or just enter 'q' (followed by an enter) to
continue to OpenGL:
0.3 0.3 0.3
The testing point (0.300000, 0.300000, 0.300000) is ***OUTSIDE*** of object
quit
```

Učitavanje zmaja rezultira sljedećim slikama:



Treći ciklus laboratorijskih vježbi - varijanta B

Treći ciklus se bavi projiciranjem objekta na prikazno platno odnosno ekran, uklanjanjem skrivenih poligona tijela i crtanjem bezierovih krivulja. U nastavku su opisana rješenja vježbi 6B, 7B i 8B.

Vježba 6B - Projekcije

Šesta vježba je sačinjena od tri zadatka. Prvi i drugi zadatak je prepustiti OpenGL-u sav posao matričnog izračuna projekcije učitano objekta. Treći zadatak je samostalno implementirati posao transformiranja i projiciranja objekta.

Rješenje je strukturirano kao u sljedećem ispisu:

```
./src/ex06b
├─ demo1.cpp
├─ demo2.cpp
├─ demo3.cpp
├─ pitanja.md
├─ pitanja.pdf
├─ projecting_data_model.h
├─ projecting_demo.cpp
├─ projecting_renderer.h
./src/utility
├─ color.h
├─ edge.h
├─ line.h
├─ object_model.h
├─ point.h
├─ polygon.h
├─ triangle.h
├─ utilities.h
./src/linalg
├─ abstract_matrix.cpp
├─ abstract_matrix.h
├─ abstract_vector.cpp
├─ abstract_vector.h
├─ demo_01.cpp
├─ demo_02.cpp
├─ i_matrix.h
├─ irg.cpp
├─ irg.h
├─ i_vector.h
├─ matrix.cpp
├─ matrix.h
├─ matrix_sub_matrix_view.cpp
├─ matrix_sub_matrix_view.h
├─ matrix_transpose_view.cpp
├─ matrix_transpose_view.h
├─ matrix_vector_view.cpp
├─ matrix_vector_view.h
├─ vector.cpp
├─ vector.h
```

```
├─ vector_matrix_view.cpp
└─ vector_matrix_view.h
```

Direktorij `linalg` sadrži implementaciju matričnih i vektorskih operacija izvedenu u sklopu vježbe 1B (koja u ovom dokumentu nije opisana). Zadatci 1, 2 i 3 su respektivno napisani u datotekama `./src/ex06b/demo1.cpp`, `./src/ex06b/demo2.cpp` i `./src/ex06b/demo3.cpp`. Sva tri programa imaju zajedničkog crtača `./src/ex06b/projecting_renderer.h`, zajednički `DataModel` `./src/ex06b/projecting_data_model.h` i zajednički dio vezan uz pokretanje demonstracijskog programa u datoteci `./src/ex06b/projecting_demo.cpp` (učitavanje i stvaranje `.obj` datoteke).

Programi se mogu pokrenuti sljedećim naredbama:

```
# 6.1B
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex06b/demo1.cpp -o ex06b-1.out
./ex06b-1.out ./src/objects/kocka.obj
./ex06b-1.out ./src/objects/dragon.obj

# 6.2B
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex06b/demo2.cpp -o ex06b-2.out
./ex06b-2.out ./src/objects/kocka.obj
./ex06b-2.out ./src/objects/dragon.obj

# 6.3B
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex06b/demo3.cpp -o ex06b-3.out
./ex06b-3.out ./src/objects/kocka.obj
./ex06b-3.out ./src/objects/dragon.obj
```

Nakon pokretanja programa se u konzolu ispisuju kratke upute korištenja, odnosno koju funkcionalnost koja tipka pruža. Ispis kratkih uputa je sljedeći:

```
USAGE INSTRUCTIONS:
  Press 1/2/3/4 to pick drawing algorithm (if available)
  Press 'f' to turn on fill (if available)
  Press 'r' or 'l' to rotate the object right or left
  Press 'q' or ESC to quit
```

Vježba 7B - Uklanjanje skrivenih poligona

U sedmoj vježbi je potrebno ukloniti skrivene poligone na dva načina. Prvi je način korištenjem gotove OpenGL funkcionalnosti i on je demonstriran programom `./src/ex07b/demo1.cpp`. Drugi način je samostalno implementirati uklanjanje skrivenih poligona, što je demonstrirano programom `./src/ex07b/demo1.cpp`. Struktura rješenja je sljedeća:

```
./src/ex07b
├─ demo1.cpp
├─ demo2.cpp
./src/utility/
├─ ...
./src/linalg/
├─ ...
```

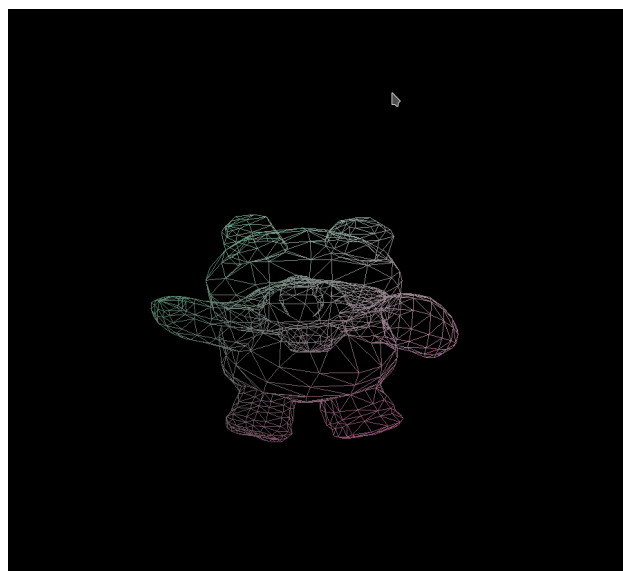
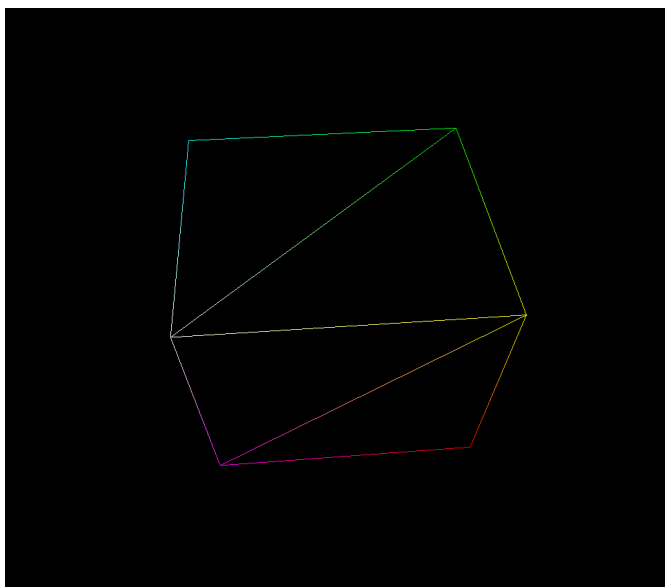
Za pokretanje programa, mogu se koristiti sljedeće naredbe:

```
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex07b/demo1.cpp -o ex07b-1.out
./ex07b-1.out ./src/objects/kocka.obj
./ex07b-1.out ./src/objects/dragon.obj

g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex07b/demo2.cpp -o ex07b-2.out
./ex07b-2.out ./src/objects/kocka.obj
./ex07b-2.out ./src/objects/teddy.obj
./ex07b-2.out ./src/objects/dragon.obj
```

Nakon pokretanja programa se u konzolu ispisuju kratke upute korištenja i one su jednake onima iz vježbe 7B jer se dijeli funkcionalnost implementirana u [./src/ex06b/projecting_demo.cpp](#).

Neke slike nastale uklanjanjem poligona su ispod.



Vježba 8B - Bezierova krivulja

Osma vježba se bavi izradom prostorne krivulje postupkom Bezijera. Ovo je rješenje sljedeće strukture:

```
./src/ex08b
├─ bezier_data_model.h
├─ bezier_demo.cpp
├─ bezierPainter.h
./src/utility/
├─ ...
./src/linalg/
├─ ...
```

Za pokretanje programa je moguće koristiti sljedeće naredbe:

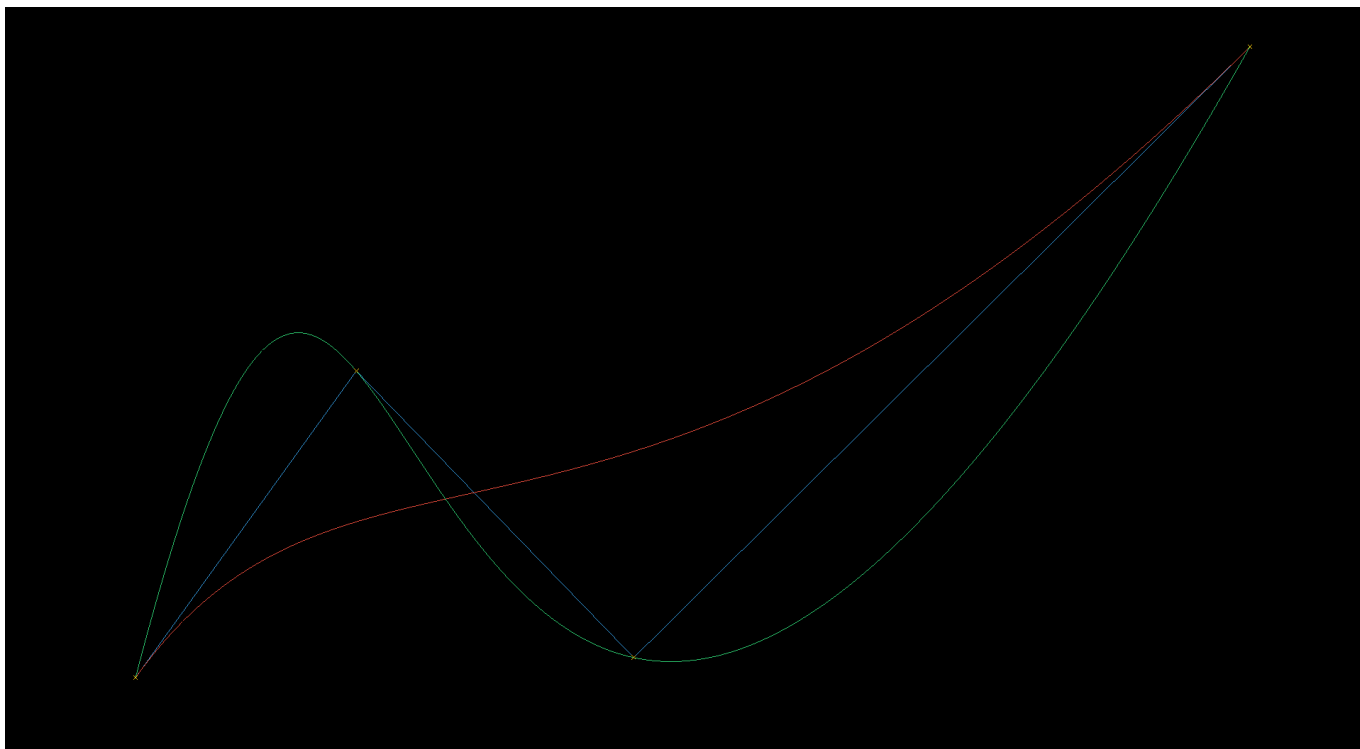
```
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex08b/bezier_demo.cpp -o ex08b.out
./ex08b.out
```

Nakon pokretanja programa se ispisuje upute korištenja i one su sljedeće:

```
### COLORING ###
    Press 1 to switch background color.
    Press 2 to switch approximation curve color.
    Press 3 to switch interpolation curve color.
    Press 4 to switch line strip color.
    Press 5 to switch cross color.
### INTERACTION USING MOUSE ###
    Press anywhere with left mouse click to add an point.
    Press and drag any of the points using right mouse button.
    Press r to reset all curve points.
Press q to quit.
```

Točaka Bezijerove krivulje se dodaju klikom miša i mogu se pomicati povlačenjem pojedine točke mišom. Kako se točke mijenjaju, tako se iscrtavaju aproksimacijska i interpolacijska Bezijerova krivulja, kao i niz linija koje povezuju dodane točke redoslijedom kojem su dodane na platno. Dodavanjem većeg broja (>6) točaka se može primijetiti znatno sporije iscrtavanje krivulja. Mislim da je uzrok toga računaska složenost i neefikasnost implementirane `./src/linalg/` biblioteke.

Primjer nacrtanih krivulja je na slici ispod.



Četvrti ciklus laboratorijskih vježbi - varijanta B

Četvrti se ciklus bavi RayCasting-om i crtanjem jednostavnih fraktala. Rješenja obuhvaćaju laboratorijske vježbe 10B i 11B i opisana su u nastavku dokumenta.

Vježba 10B - Algoritam praćenja zrake

U okviru ove vježbe je zadatak napraviti program koji će kao argument komandne linije primiti naziv datoteke s opisom scene te koji će potom algoritmom za praćenje zrake obaviti prikaz zadane scene. Rješenje je strukturirano kao u prikazano u nastavku:

```
./src/ex10b
├─ input.txt
├─ raytracing_data_model.h
├─ raytracing_demo.cpp
├─ raytracingPainter.h
./src/utility
├─ bezier.h
├─ color.h
├─ edge.h
├─ line.h
├─ object_model.h
├─ point.h
├─ polygon.h
├─ rayTracing
│   ├── light.h
│   ├── material.h
│   ├── patch.h
│   ├── ray.h
│   ├── scene.h
│   └── scene_object.h
```

```
|   └─ sphere.h  
|   └─ triangle.h  
|   └─ utilities.h
```

Za potrebe rješavanja ovog zadatka je korištena GLM biblioteka za matematiku. Tu sam odluku donio zbog sumnje u neefikasnost biblioteke `linalg` implementirane u vježbi 1B. Ipak, mislim da korištenje GLM biblioteke nije znatno promijenilo vrijeme iscrtavanja slike, sudeći po subjektivnom dojamu proteklog vremena. Jasno je da je RayTracing računalno zahtjevan zadatak te se očekuju sporije računanje na prosječnim laptopima kao što je ovaj koji sam koristio. Tomu bi se moglo doskočiti paralelizacijom, jer je postupak računanja obojenja piksela sramotno lako za paralelizirati. To bi značajno poboljšalo vremenske performanse.

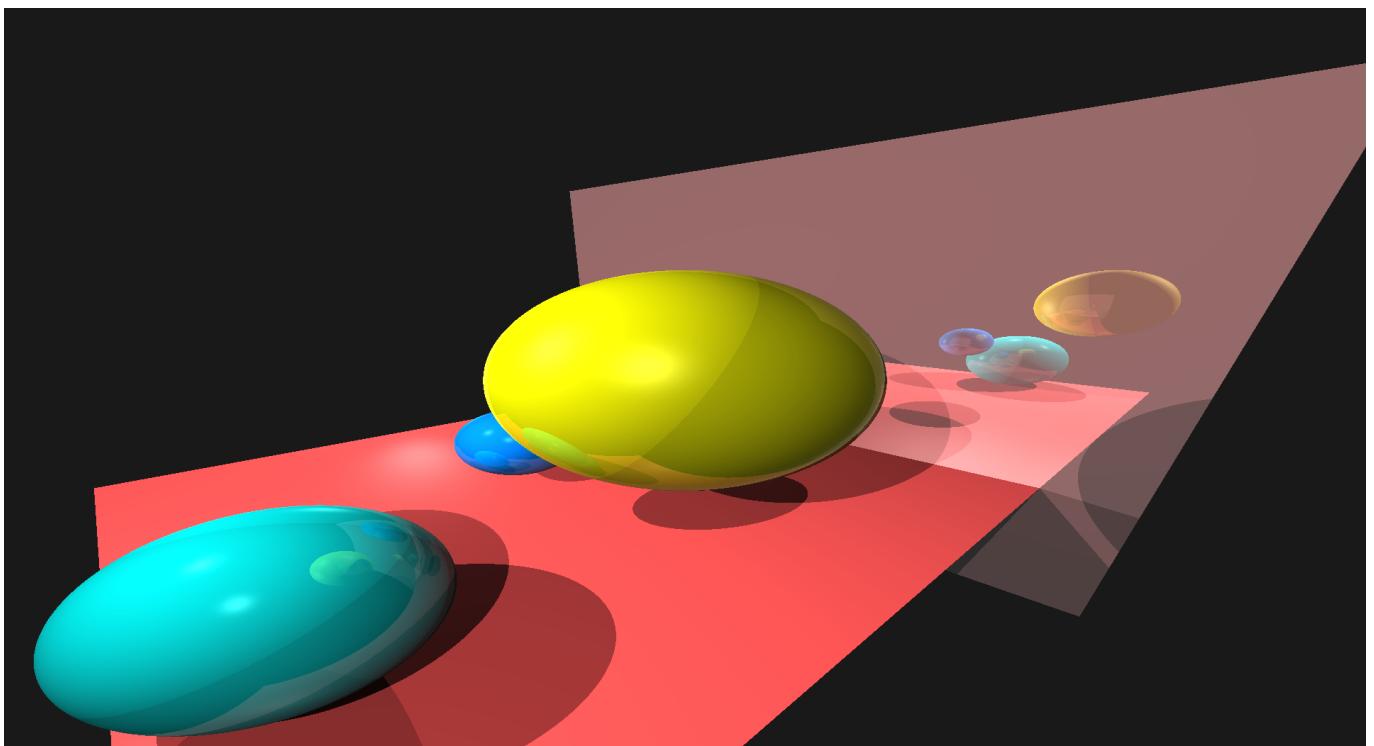
Program se može pokrenuti sljedećim naredbama:

```
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw  
./src/ex10b/raytracing_demo.cpp -o ex10b.out  
./ex10b.out ./src/ex10b/input.txt
```

Nakon pokretanja ispisuje kratke upute korištenja:

```
Press 1 to decrease max recursion limit.  
Press 2 to increase max recursion limit.
```

Slika jedne scene koju je program nacrtao je prikazana na slici ispod.



Jedanaesta vježba demonstrira jednostavan postupak za izradu i prikaz fraktalnih skupova, najprije Mandelbrotovih fraktala i zatim IFS fraktala.

Projekt je strukturiran kao u što je prikazano u nastavku:

```
./src/ex11b
├── ifs-fractals
│   ├── ifs_data_model.h
│   ├── ifs_demo.cpp
│   ├── ifsPainter.h
│   ├── paprat.txt
│   ├── sierpinski_carpet.txt
│   └── sierpinski_triangle.txt
└── mandelbrot
    ├── mandelbrot_data_model.h
    ├── mandelbrot_demo.cpp
    └── mandelbrotPainter.h
```

Za pokrenuti programe moguće je unijeti navedene naredbe u naredbeni redak, ovisno o željenom fraktalu:

```
# MANDELBROT:
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex11b/mandelbrot/mandelbrot_demo.cpp -o ex11b-mandelbrot.out
./ex11b-mandelbrot.out
# IFS FRAKTALI:
g++ -std=c++17 -Wall -Wextra -Werror -lglut -lGL -lGLU -lglfw
./src/ex11b/ifs-fractals/ifs_demo.cpp -o ex11b-ifs.out
./ex11b-ifs.out ./src/ex11b/ifs-fractals/paprat.txt
./ex11b-ifs.out ./src/ex11b/ifs-fractals/sierpinski_carpet.txt
./ex11b-ifs.out ./src/ex11b/ifs-fractals/sierpinski_triangle.txt
```

Mandelbrotov program pri pokretanju ispiše sljedeće upute za korištenje dostupnih funkcionalnosti na standardni izlaz:

```
*****
***** MANDELBROT *****
*****

### DRAWING ALGORITHM ###
    Press 1 to use algorithm 1.
    Press 2 to use algorithm 2.
    Press 3 to decrease max recursion limit.
    Press 4 to increase max recursion limit.
### COLORING ALGORITHM ###
    Press b to use color algorithm "black&white"
    Press v to use color algorithm "blues"
    Press c to use color algorithm "colorful"
### ZOOMING ###
```

Press anywhere using mouse to zoom **in** at that point. The old panel dimensions will be saved.
 Press **x** to restore last saved panel dimensions.
 Press **ESC** to restore initial panel state.
 Use arrow keys the eye position above the drawing panel.

Press **q** to quit.

Program za crtanje IFS fraktala ispisuje podatke o učitanoj konfiguracijskoj datoteci zajedno s uputama za korištenje. Primjer takvog ispisa je u nastavku:

```
*****
***** IFS FRACTALS *****
*****

Press 1 to decrease depth limit.
Press 2 to increase depth limit.
Press 3 to decrease number of points by 10000.
Press 4 to increase number of points by 10000.
Press q to quit.

Loading configuration from path './src/ex11b/ifs-fractals/paprat.txt'
200000
25
80 300
60 0
0.00 0.00 0.00 0.16 0.00 0.00 0.01
0.85 0.04 -0.04 0.85 0.00 1.60 0.85
0.20 -0.26 0.23 0.22 0.00 1.60 0.07
-0.15 0.28 0.26 0.24 0.00 0.44 0.07
```

Stvorene slike Mandelbrotovih i IFS fraktala su u nastavku dokumenta.

