DOKUMENTACIJA LABORATORIJSKIH VJEZBI IZ PREDMETA INTERAKTIVNA RACUNALNA GRAFIKA

Rješenja svih laboratorijskih vježbi su nakodirana u programskom jeziku C/C++. Cilj laboratorijskih vježbi je bio praktično upotrijebiti teoriju stečenu kontinuiranim pohodom na predavanja te čitanjem službene literature. Većinski dio programskog rješenja vježbi je ostvaren po natuknicama, predloženoj strukturi i ideji rješenja te priloženim kodovima iz službene knjige predmeta (http://www.zemris.fer.hr/predmeti/irg/knjiga.pdf). Zadaci laboratorijskih vježbi se mogu pronaći na stranici predmeta (http://www.zemris.fer.hr/predmeti/irg/laboratorijske_vjezbe.html).

VJEŽBA 1

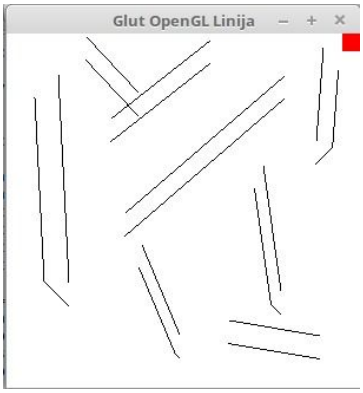
Zadatak prve vježbe je programski ostvariti osnovne matematičke operacije koje se koriste u računalnoj grafici. Definirane su funkcije za rad s vektorima i matricama. Vektori su pohranjeni u strukturu podataka std::vector, a matrice u std::vector>. Ostvarene bazne funkcije za rad s vektorima su vektorski i skalarni produkt, zbrajanje, oduzimanje, komplement, duljina vektora, ispis i normiranje, a za rad s matricama su ostvarene funkcije izračuna determinante, transponiranje, inverz, množenje, zbrajanje te ispis. S obzirom na to da su matrice ostvarene poput dvodimenzionalnih vektora, prilikom ostvarenja funkcija za rad s matricama iskorištena su samostalna programska rješenja matematičkog rada s vektorima.
Radni zadatak prve vježbe se sastojao od 3 dijela. Prilikom rješavanja radnih zadataka u glavnom programu iskorištene su ostvarene funkcije za rad s matricama i vektorima.
Cilj prvog dijela je bio ispitivanje rada ostvarenih funkcija s priloženim testnim primjerima.
Cilj drugog dijela zadatka je bio omogućiti korisniku unos sustava 3 jednadžbe s 3 nepoznanice (x, y, z) te potom ispisati rješenje tog sustava. Programsko ostvarenje rješenja ovog problema je pomnožiti invertiranu matricu lijeve strane sustava odnosno nepoznanica koje je korisnik unio s transponiranom matricom konstanti s desne strane. Navedeni umnožak kao izlazni ispis izbacuje rješenje sustava 3 jednadžbi s 3 nepoznanice.
Cilj trećeg dijela zadatka je bio ispisati baricentrične koordinate točke T s obzirom na trokut ABC. Baricentrične koordinate korisne su kod određivanja da li se točka nalazi u trokutu u 3D prostoru. Ako su vrhovi trokuta A, B, C za neku točku T imamo baricentričnu kombinaciju: T= t1*A + t2*B + t3*C gdje su t1, t2, t3 baricentrične koordinate. Za baricentričnu kombinaciju nužno vrijedi da je suma jednaka jedan tj t1 + t2 + t3 = 1. Korisniku je bilo potrebno omogućiti unos točaka trokuta te točke koju ćemo istraživati u odnosu na navedeni trokut. Izračun baricentričnih koordinata je ostvaren umnoškom matrica M1 i M2. Matricu M2 dobijemo transponiranjem koordinata točke T, a matricu M1 dobijemo invertiranjem transponirane matrice trokuta. U zadatku je bilo naglašeno da se ne trebamo zamarati lošim unosom točaka sto je potencijalni problem jer nismo pokrili sve slučajeve. U programskom rješenju smo koristili invertiranje matrice, a može se desiti da matrica nije invertibilna. Primjer takvog slučaja je kada je jedan vrh trokuta u ishodištu ili kada su točke trokuta u jednoj od ravnina xy, xz ili yz pa je jedna od koordinata nula.

VJEŽBA 2

Zadatak druge vježbe je iscrtavanje linije Bresenhamovim postupkom. Bresenhamov algoritam je jedan od temeljnih postupaka pretvorbe iz kontinuiranog oblika u diskretni oblik potreban prilikom prikaza na zaslonu. Za iscrtavanje linije na zaslonu najčešće se koristi Bresenham-ov postupak za 45 stupnjeva koji se modificira kako bi radio i za ostale stupnjeve. Između dvije točke zaslona potrebno je iscrtati ravnu liniju što dovodi do problema izbora točke za osvjetljavanje. Kriterij izbora točaka rastera sastoji se u računanju udaljenosti okolnih točaka rastera od linije. Ideja algoritma je postaviti pravokutnik oko okolnih točaka rastera. Zatim za trenutnu x i y vrijednost izračunati tangens kuta. Ako je tangens 45 ili vise, vrijednost tangensa će biti jednaka ili veća od 1. U našem algoritmu, varijabla D koju određuje tangens kuta je skalirana na vrijednosti na simetričnom intervalu od -0.5 do 0.5 kako bi olakšali odlučivanje trebamo li povećati koordinatu y. Koordinatu x konstantno povećavamo dok ne dođemo do kraja ekrana, a odluku o povećavanju y donosi varijabla D. S obzirom na to da je interval simetričan, provjeravamo ponašanje varijable D oko nule. Za pozitivnu vrijednost D, potrebno je povećati y koordinatu i zatim vratiti varijablu D natrag na interval 0.5 do 0.5 tj. umanjiti ju za 1. Nakon svake iteracije uvećamo varijablu D za vrijednost dy/(double)dx odnosno tangens kuta. Opisani algoritam funkcionira ako je kut linije manji od 45 stupnjeva te ako je dx veće od dy. Ako je dx manji od dy, treba opisani algoritam malo modificirati odnosno samo zamijeniti uloge x i y osi.

Korisniku je bilo potrebno omogućiti unos koordinata točaka V1 i V2 koje su se zatim iscrtale na prozoru ekrana. Korisnik je imao izbor unosa koordinata pomoću tipkovnice ili pak mišem kliknuti na željene točke. Osim iscrtavanja linije pomoću Bresenhamovog algoritma, za usporedbu je bilo potrebno iscrtati liniju s malo pomaknutim y koordinatama pomoću glutovog primitiva GL_LINES tj za y+20.

Primjer iscrtavanja linija s mišem, vidljiva je gruba aproksimacija linije za male kuteve blizu koordinatnih osi:



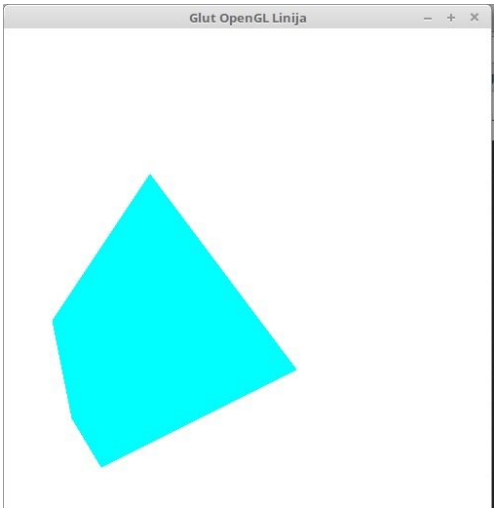
VJEŽBA 3

Radni zadatak vježbe 3 je iscrtavanje obojanog poligona na zaslon. Cilj vježbe je bio shvatiti na koji način funkcionira algoritam bojanja poligona. Za konveksna tijela je radio idealno, međutim s konkavnim tijelima je došlo do problema. Ograničavanjem da desno mora biti veće od lijevog malo popravimo kod da se ne iscrtavaju dijelovi van tijela, međutim i dalje nije tijelo idealno obojano.

Implementirana je funkcija koja provjerava odnos točke i poligona. Ako za točku A i sve bridove bi vrijedi A*bi > 0, i=1,...,n tada je točka izvan poligona, u suprotnom je unutar poligona. Za bridove je bitno da su svi određeni na isti način tj. da su svima vrhovi zadani u smjeru kazaljke na satu ili pak obrnuto. Ako je neki brid krivo zapisan u datoteci koji iščitavamo, najlakše rješenje problema bi bilo da pronađemo taj redak i zamijenimo ispravnim zapisom.

```
int ProvjeriTockuIPoligon (iPolyElem *polel, int n, iTocka2D dot) {
	int i0=n-1, ok = 1;
	for(int i=0; i<n; i++) {
		int count = dot.x*polel[i0].Brid.a + dot.y*polel[i0].Brid.b + polel[i0].Brid.c;
		if (count>0) {
			ok = -1;
			break;
		}
		i0=i;
	}
	return ok;
}
```

Primjer bojanja konveksnog poligona:



VJEŽBA 4

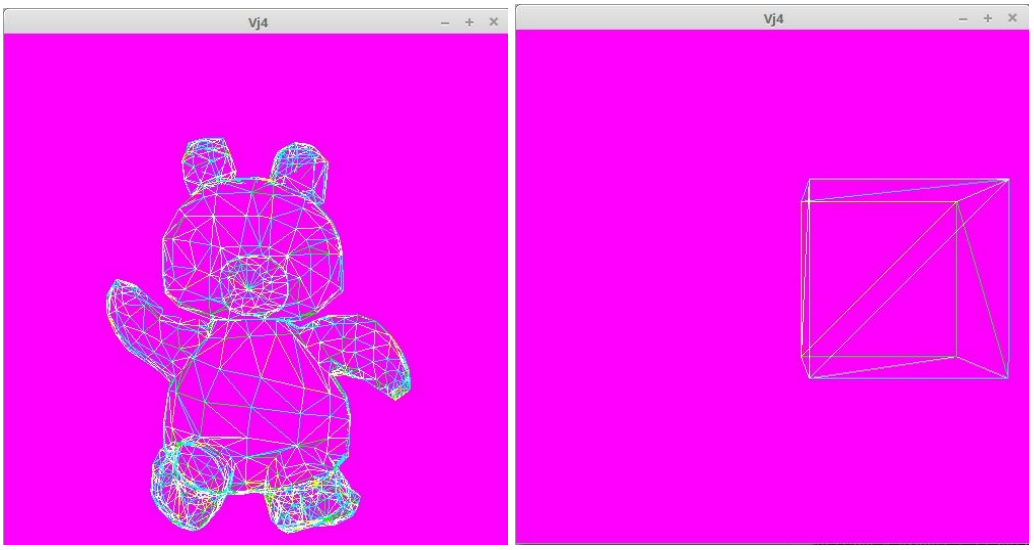
Cilj četvrte vježbe je bio iz datoteke iščitati objekt i nacrtati ga. Nakon učitavanja objekta, potrebno je bilo izračunati koeficijente ravnina (poligona). Pomoću koeficijenata ravnina smo odredili nalazi li se ispitna točka T izvan ili unutar objekta.

Korištene strukture podataka u programu su:

```
typedef struct { float x; float y; float z; } iTocka3D;
typedef struct { float a; float b; float c; float d; } KoefRavn;
typedef struct { vector<iTocka3D> Vrh;
```

KoefRavn Ravnina; int lijevi; } iPolyElem;
typedef struct { vector<string> Tocke; vector<iTocka3D> Vrh; vector<iPolyElem> Poligon; } Objekt;

Primjer teddy.obj i kocka.obj:



VJEŽBA 5

Cilj vježbe pet je zadati poligon te načiniti transformaciju pogleda i perspektivnu projekciju. Pri unosu očista i gledišta pomoću tipkovnice trebalo je pripaziti da je očiste izvan objekta. Kako bi korisniku bio olakšan odabir koordinata očista i gledišta, pri parsiranju datoteke objekta je izračunati maksimum i minimum varijabli x, y i z te ispisan na ekran. Transformaciju pogleda T smo dobili umnoškom pet matrica elementarnih transformacija: T1 - pomak ishodišta u točku O, T2 - rotacija za kut oko z osi, T3 - rotacija za kut oko y osi, T4 - rotacija za kut 90° oko z osi, T5 - promjena predznaka na x osi. Matricu perspektivne projekcije smo izračunali kao P = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1/H}, {0,0,0,0}}; pri čemu H iznosi H = G3.z {gledište iz prethodne transformacije}. Pomoću matrice perspektivne projekcije i matrice transformacije pogleda transformiramo sve vrhove prije crtanja.

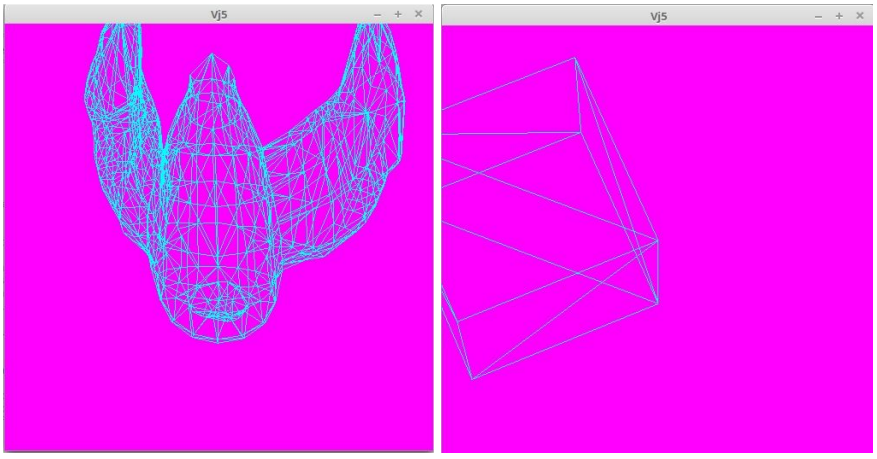
Funkcija za crtanje poligona tada izgleda:

```
for (iPolyElem poligon: obj.Poligon) {
                glBegin(GL_LINE_LOOP);
                for (iTocka3D vrh: poligon.Vrh) {
                        vrh = transformacija(vrh, T);
                        vrh = transformacija(vrh, P);
                        glVertex3f(vrh.x, vrh.y, vrh.z);
                        printf("crtaj: %f %f %f\n", vrh.x, vrh.y, vrh.z);
                }
                glEnd();
}
```

Za tipove podataka su korištene sljedeće strukture:

typedef struct { double x; double y; double z; double h=1.0; double Intenzitet; Normala NormalaVrha; } iTocka3D;
typedef struct { double a; double b; double c; double d; } KoefRavnine;
typedef struct { vector<iTocka3D> Vrh; iBrid3D Brid; int lijevi; } iPolyElem;
typedef struct { vector<vector<int>> Tocke; vector<iTocka3D> Vrh; vector<iPolyElem> Poligon; } Objekt;

Primjer: bird.obj, kocka.obj

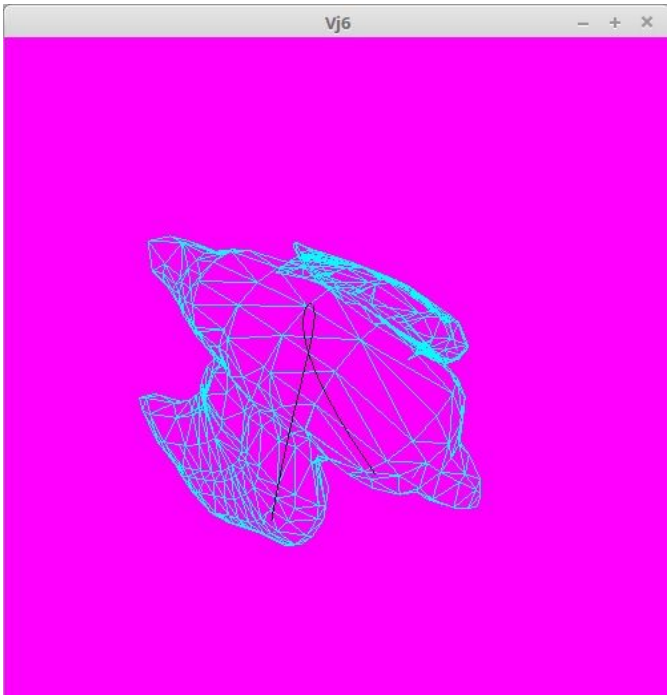


VJEŽBA 6

Cilj šeste vježbe je bio ostvariti provlačenje glatke krivulje između zadanog niza točaka pomoću Bézierove krivulje te sakriti stražnje poligone. Programsko rješenje ovog zadatka se bazira na Bernsteinovim težinskim funkcijama. Točke na krivulji aproksimiramo pomoću Bernsteinovog polinoma b = binomni_umnozak(n,i) * pow(t, i) * pow(1-t, n-i); s kojim pomnožimo sve koordinate vrha {x += vrh.x * b; y += vrh.y * b; z += vrh.z * b;}. Parametar t uzmemo što manji i lagano ga uvećavamo kako bi dobili što finiju krivulju. Ako za pomak parametra t uzmemo npr. 0.5, dobit ćemo grubu aproksimaciju krivulje koja će biti nazubljena, dok će za 0.02 biti finija. Za transformaciju pogleda i perspektive smo morali odrediti odgovarajuće matrice transformacija vrhova koju smo iskoristili iz prethodne vježbe. Vrhove Béziera smo isčitali iz datoteke i parsirali u funkciji void parseBezier(string fileName);, a vrhove objekta parsirali u void parseFile(string fileName);. Datoteka koja sadrži koordinate Béziera i datoteka koja sadrži objekt su zadane kao argumenti naredbenog retka.

Za tipove podataka su korištene sljedeće strukture:
typedef struct { double A; double B; double C; } Normala;
typedef struct { double x; double y; double z; double h=1.0; double Intenzitet; Normala NormalaVrha; } iTocka3D;
typedef struct { double a; double b; double c; double d; } KoefRavnine;
typedef struct { vector<iTocka3D> Vrh; KoefRavnine Ravnina; iTocka3D SredisnjaTocka; int lijevi; Normala NormiranaNormalaPoligona; } iPolyElem;
typedef struct { vector<vector<int>> Tocke; vector<iTocka3D> Vrh; vector<iPolyElem> Poligon; } Objekt;

Primjer iscrtavanja Bézierove krivulje na brid.obj kroz 5 vrhova:



VJEŽBA 7

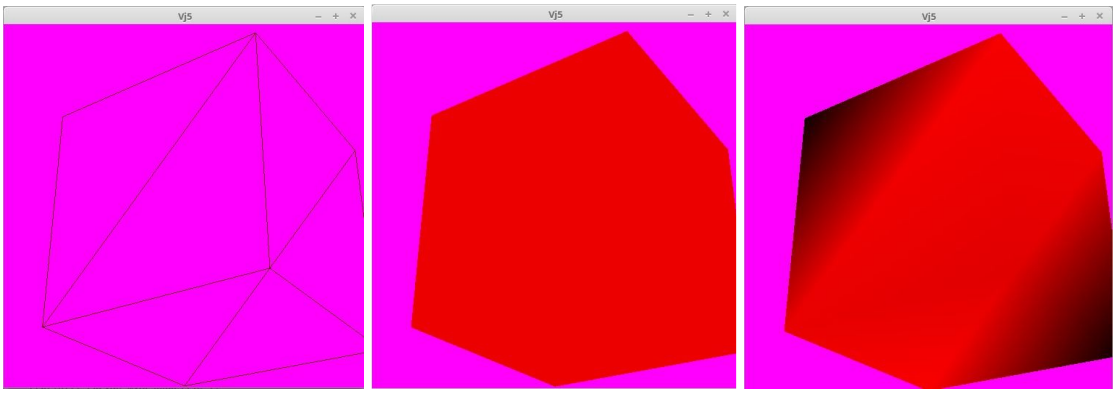
Zadatak sedme vježbe je usporedba konstantnog sjenčanja i Gouraud-ovog sjenčanja na osnovi proračuna intenziteta te postupak uklanjanja stražnjih poligona. Za konstantno sjenčanje potrebno je poznavati normale poligona. Za Gouraud-ovo sjenčanje potrebno je odrediti normale u vrhovima objekta. Na osnovi normala određuje se intenzitet i boji se poligon. Iscrtavanje objekata poželjno je ubrzati tako da uklonimo stražnje poligone, no taj način nije onda pogodan za animacije.

Za tipove podataka su korištenje sljedeće strukture:
typedef struct { double A; double B; double C; } Normala;
typedef struct { double x; double y; double z; double h=1.0; double Intenzitet; Normala NormalaVrha; } iTocka3D;
typedef struct { double a; double b; double c; double d; } KoefRavnine;
typedef struct { vector<iTocka3D> Vrh; KoefRavnine Ravnina; iTocka3D SredisnjaTocka; int lijevi; Normala NormiranaNormalaPoligona; } iPolyElem;
typedef struct { vector<vector<int>> Tocke; vector<iTocka3D> Vrh; vector<iPolyElem> Poligon; } Objekt;

Vrh je definiran kao vektor trodimenzionalnih točki. Ravnina je definirana pomoću koeficijenata ravnine. Objekt je definiran pomoću vrhova i poligona objekta koji se dobiju pomoću funkcije void parseFile(string fileName), a ime datoteke fileName korisnik unosi kao argument naredbene linije. Vidljivost poligona je ostvarena pomoću funkcije iz prethodne vježbe usporedbom normala. Kada pomnožimo normalu gledatelja i normalu poligona, ako je umnožak negativan poligon je skriven i ne bojamo ga. Iako ovaj način nije pogodan za animaciju, ubrzava čitav proces bojanja. Za konstantan intenzitet poligona nam je potrebna središnja točka ravnine. Normala L će ovisiti o položaju izvora i središnje točke ravnine koju promatramo. Normiranjem vektora L i normiranjem normale ravnine dobijemo intenzitet

pomoću formule {Ia*ka + Ii*kd*multiplyNorm(normiranL, normiranN);}. Cijeli poligon ima isti intenzitet određen na osnovi modela osvjetljenja. Gouraudov postupak sjenčanja koristi izračunavanje intenziteta u vrhovima na osnovi normala u vrhovima. Tako dobiveni intenzitet linearno se interpolira na poligonu. To znači da ćemo na osnovi nekoliko ili beskonačno različitih normala u vrhovima objekta odrediti intenzitete u vrhovima.

Primjer bez sjenčanja, s konstantnim i Gouraudovo sjenčanje:



VJEŽBA 8

Zadatak osme vježbe je jednostavan postupak za izradu i prikaz fraktalnih skupova. Fraktali su geometrijski objekti koji daju jednaku razinu detalja neovisno o razlučivosti koju koristimo. Programsko rješenje oba fraktala se sastoji od ispitivanja svake točke zaslona pripada li skupu ili ne i ovisno o tome ju obojali. Pripadnost točke skupu smo provjeravali testom divergencije koji kao rezultat vraća brzinu divergencije ako uopće postoji. Ako točka konvergira obojali smo ju crnom bojom, a ako točka divergira obojali smo ju intenzitetom boja RGB ovisno o brzini divergencije.
Radni zadatak se sastojao od 2 dijela, programski ostvariti iscrtavanje Mandelbrotovog skupa te Julijevog skupa. Mandelbrotov skup je skup točaka u kompleksnoj ravnini i čiju granicu nazivamo Mandelbrotovim fraktalom. Definirajmo prvo kompleksnu rekurzivnu funkciju zn+1 kao zn+1 = z 2 n + c uz početni uvjet z0 = 0. Zatim provjerimo je li modul ovih kompleksnih brojeva koje generira promatrana rekurzivna funkcija ograničen, ako pustimo da n teži u beskonačnost tj ∀n zn < eps. U geometrijskom smislu, zanima nas ako oko ishodišta kompleksne ravnine nacrtamo kružnicu radijusa eps, hoće li svi kompleksni brojevi koje generira ova rekurzivna jednadžba ostati unutar te kružnice. Ako je odgovor pozitivan, tada kompleksni broj c za koji smo radili ispitivanje pripada Mandelbrotovom skupu, a ako je negativan tada divergira i ne pripada. Praksa je pokazala da nakon eps > 2 limes sigurno divergira tj točka ne pripada Mandelbrotovom skupu pa u kodu možemo ograničiti eps s gornje strane s 2. Algoritam za određivanje Mandelbrotovog skupa funkcionira tako da odredimo brzinu divergencije ako za točku funkcija divergira i primijenimo odgovarajući intenzitet boja, a ako konvergira obojamo fraktal crnom bojom. Za Julijev skup definirajmo kompleksnu funkciju oblika f(z) = z^2 + c. Za razliku od Mandelbrotovog skupa gdje je z0 = 0, kod Julijevog skupa parametar c se fiksira na početku, a svaka točka ekrana (x, y) odredit će zasebnu početnu točku z0 = (u, v). Svi Julijevi skupovi mogu se podijeliti u dvije grupe: povezani i nepovezani Julijev skup. Povezani Julijev skup nastaje kada kao vrijednost parametra c odaberemo vrijednost koja je u Mandelbrotovom skupu, a nepovezani kada dobijemo beskonačan broj izoliranih točaka poput prašine. Algoritam za Julijev skup je sličan

Mandelbrotovom. Prvo smo provjerili je li točka pripada Mandelbrotovom skupu tj. duljina vektora c=(xc,yc) je manja od 2 (gornja granica radijusa za kojeg Mandelbrotov skup konvergira), a zatim računali test divergencije pomoću funkcije za Julijev skup. {Određivanje radijusa kružnice: double eps = max(sqrt(c.re*c.re + c.im*c.im), 2.0); } Ovisno o konvergenciji i divergenciji točku smo bojali crno, odnosno za divergenciju odgovarajućim intenzitetom.
Primjer crteža Mandelbrotovog skupa i Julijevog skupa(primjer je neobičan zbog testiranja parametara):