Interaktivna računalna grafika

Ak. god. 2019./2020.

Dokumentacija

Laboratorijske vježbe

Ivana Barišić

Svibanj, 2020

VJEŽBA 1: Osnovne operacije u RG

U prvoj laboratorijskoj vježbi bilo je potrebno programski ostvariti osnovne matematičke operacije koje su nam kasnije bile potrebne za ovaj predmet. Operacije su se pretežno radile nad vektorima (jednodimenzionalno polje kao *Numpy array*) i matricama koje su bile pohranjene kao dvodimenzionalna polja. Operacije koje smo ostvarili nad vektorima bile su zbrajanje, oduzimanje i skaliranje, kao i skalarni i vektorski produkt. Operacije nad matricama bile su izračunavanje determinante, inverza, transponiranje matrice, međusobno množenje i zbrajanje matrica. Radni zadatak se sastojao od 3 dijela. Programsko rješenje napisano je u programskom jeziku Python. U prvom, pohranjeno kao *matrice.py*, trebalo je demonstrirati funkcionalnosti opisanih operacija nad vektorima i matricama na priloženim testnim primjerima. Sve funkcionalnosti odvijaju se u glavnom programu. U drugom dijelu radnog zadatka, pohranjeno kao *jednadzba.py*, trebalo je korisniku omogućiti unos sustava od 3 jednadžbe od 3 nepoznanice x, y ,z te se rezultat sustava jednadžbi se ispisuje na zaslon u obliku [x y z]. Korisnik preko tipkovnice redom unosi podatke o prvoj jednadžbi pa podatke o drugoj jednadžbi te konačno podatke o trećoj jednadžbi i svaki podatak odvaja zarezom, a program to pamti pomoću metode input. Primjer dobrog upisa podataka je: „1, 1, 1, 6, -1, -2, 1, -2, 2, 1, 3, 13“, što prvu jednadžbu definira kao „x + y +z = 6. Sirovi podatci se tada pretvaraju u dvije matrice: a (sadržava koeficijente u x, y i z) i b (sadrži slobodne članove) koje su pohranjene kao *Numpy array*.

Cilj trećeg zadatka , pohranjen kao baricentricne_koordinate.py, bio je ispisati baricentrične koordinate točke T s obzirom na koordinate vrhova trokuta ABC koji se nalazi u 3D prostoru. One su korisne za određivanje nalazi li se točka unutar ili van trokuta u 3D prostoru. Za neku točku T nalazimo 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. korisnik preko tipkovnice, kao i u prethodnom primjeru slijedno unosi koordinate točaka vrhova A,B,C i naposlijetku koordinate točke C, također odvojene zarezima. Izračun baricentričnih koordinata ostvaren je umnoškom matrice „a“ (dobivena invertiranjem transponirane matrice trokuta) i matrice „b“ (dobivena transponiranjem koordinata točke T). Potencijalan problem koji nisam prekrila u svom rješenju je ukoliko matrica „a“ koju je potrebno invertirati nije invertibilna, zadatak nije u mogućnosti izračunati baricentične koordinate točke. To bi se dogodilo ukoliko je jedan vrh trokuta u ishodištu ili su točke trokuta u jednoj od ravnina xy, xz ili yz i to rezultira jednom koordinatom da bude 0. U zadatku je bilo naglašeno kako se tim problemom lošeg zadavanje točaka ne moramo zamarati.

VJEŽBA 2: Crtanje linija na rasterskim prikaznim jedinicama

Zadatak druge vježbe bio je iscrtavanje linije Bresenham-ovim postupkom crtanja linije. Trebalo je učitati koordinate dviju točaka (T1 i T2) te pomoću njih iscrtati liniju koristeći Bresenham-ov algoritam. Nakon što smo liniju nacrtali pomoću Bresenham-a, za usporedbu smo morali nacrtati novu liniju pomoću naredbe LINE koja će uvijek biti pomaknuta za 20 po y ovi od orginalne y-koordinate (x1, y1 + 20), (x2, y2 + 20). Program je napisan u programskom jeziku *Python* i pohranjeno kao *pravac.py*. Korisnik klikom miša na rasterskoj prikaznoj jedinici pokreće window event *on_mouse_press* i na taj način (pomoću dva klika) zadaje točke T1 i T2. Potprogram *nacrtaj* prima x i y koordinate oba vrha i na temelju njih određuje koji kut zatvaraju odabrane točke i na temelju te vrijednosti se određuje za potrebne promjene:

•Ako je x1 manji od ili jednak x2 te ako je y1 manji od ili jednak y2 crtaj liniju metodom *nacrtaj_do_90* koja crta linije za kuteve 0 - 90 stupnjeva.
• Ako je x1 manji od ili jednak x2 te ako je y1 veći od y2 crtaj liniju metodom *nacrtaj_do_minus_90* koja crta linije za kuteve -90 - 0 stupnjeva.
• Ako je x1 veći od x2 te ako je y1 veći od ili jednak y2 crtaj liniju metodom *nacrtaj_do_90* koja crta linije za kuteve 0 – 90 stupnjeva.
• Ako je x1 veći od x2 te ako je y1 manji od y2 crtaj liniju metodom *nacrtaj_do_minus_90* koja crta linije za kuteve -90 - 0 stupnjeva.

Potprogram *nacrtaj_do_90* funkcionira na principu Bresenham-ovog algoritma koji zapravo crta linije (0 – 45) stupnjeva te modifikacijom nad vrhovima (zamjena x i y koordinata kako bi prividno namjestili da kutevi budu u tom rasponu i time ograničavamo da tangens kuta kojega računamo nikada ne prelazi 1 . Programsko ostvarenje postupka preuzeto je iz priloženih kodova iz službene knjige predmeta. Prvo odredimo nagib pravca (a) te početnu vrijednost y-koordinate (y3) i inicijaliziramo (y4) koja nam služi za provjeru možemo li osvijetliti piksel.

a= 2*(y2-y1)

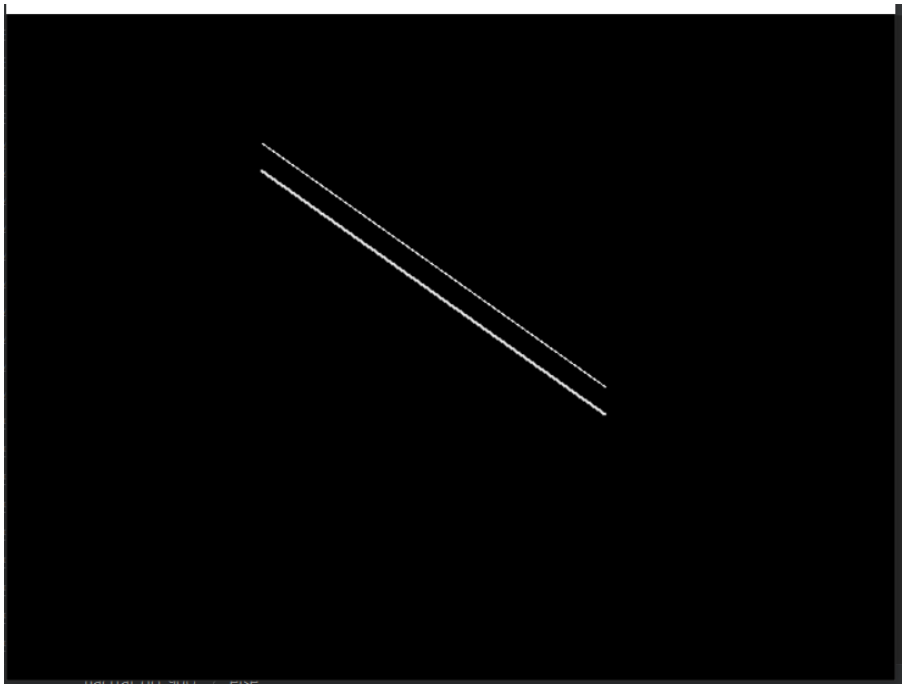
y3=y1

y4 = -(x2-x1)

Nakon toga x1 koordinatu postepeno povećavamo za jedan do vrijednosti x2 i svakim korakom osvježavamo

y4 += a

te vršimo provjeru trebamo li osvijetliti piksel (y4 > 0).Ukoliko je to istinito piksel osvjetljavamo metodom graphics.draw(GL_POINTS) iz biblioteke *Pyglet*.
Potprogram *nacrtaj_do_minus_90* radi na istom principu kao gornji potprogram, jedina razlika je u tome što se y-koordinata ne uvećava za 1, već smanjuje.
Na (slika 1.) je prikazan jedan od mogućih izlaza programa.



slika 1: pravac.py

VJEŽBA 3: Crtanje i popunjavanje konveksnog poligona

Cilj treće vježbe bio je da korisnik preko tipkovnice unese broj n koordinata vrhova konveksnog poligona kojih će potom toliko puta klikom miša na restorskom zaslonu zadavati u smjeru kazaljke na satu. Zadatak je bio iscrtati taj poligon (slika 2.), odrediti koeficjente jednadžbi bridova te naknadno klikom miša na restorsku jedinicu zadati koordinate vrha V na temelju koje će program ispitati nalazi li se totčka unutar ili van poligona. Kao posljetnji korak trebalo je obojati poligon (slika 3). Programsko rješenje napisano je u programskom jeziku *python* i nazvano *konveksni_poligon_glavni_program.py.* Koordinate unesenih vrhova pamte se u potprogramu on_mouse_press koji potom sve susjedne točke prosljeđuje potprodramu nacrtaj koji pomoću biblioteke Pyglet i metodom *graphics.draw*(GL_POINTS) vrta linije konveksnog potprograma. Koeficijenti svih bridova računaju se u potprogramu *izracunaj_koef.* Oni se računaju formulom:

$$a(i) = y(i) - y(i + 1)$$

$$b(i) = -x(i) + x(i + 1)$$

$$c(i) = x(i) * y(i + 1) - x(i +1) * y(i)$$

Nakon što smo mišem zadali točku V koju je također zapamtio potprogram *on_mouse_press* računa se odnos točke V u odnosu na iscrtani poligon u potprogramu *ispitaj_odnos* pomoću formule:

$$x * a(i) + y * b(i) + c(i) > 0$$

Koja za svaki brid provjerava ovu nejednakost i ako je istinita za samo jedan brid točka je van poligona, inače je unutar. Za postupak bojanja poligona liniju po liniju potrebno je izračunati raspon poligona tj. Minimalnu i maksimalnu vrijednost x i y koordinate.

Postepeno povećavajući korak (za 1) [ymin, ymax] postavljamo L = xmin i D = xmax kao lijevu i desnu granicu poligona. Za svaki i u [0, n-1] izvodi se sljedeće:

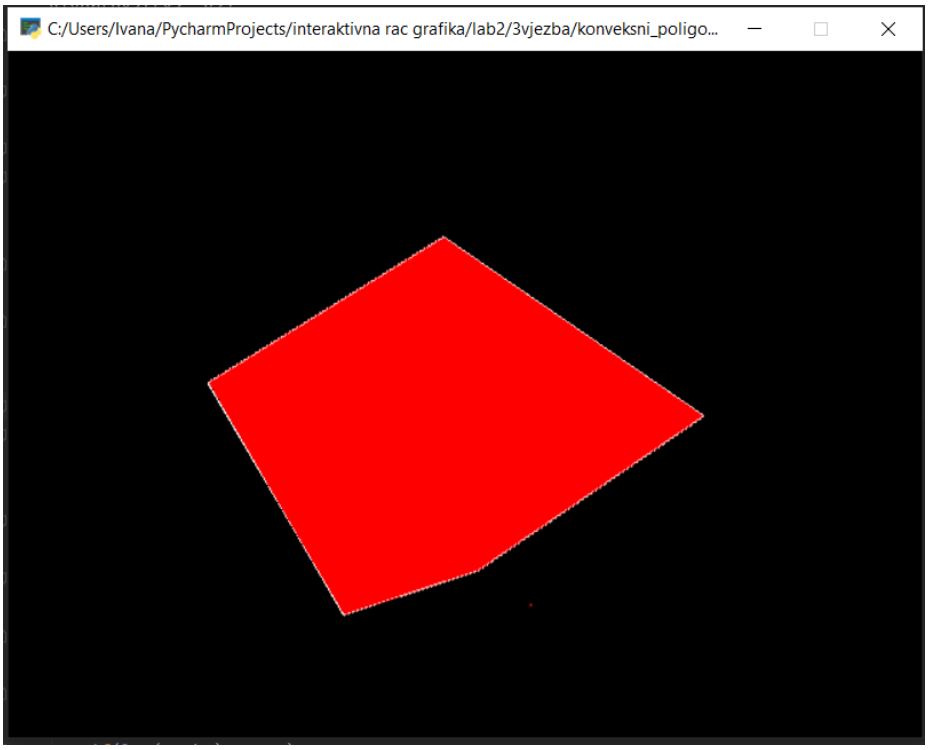
Ako A(i) != 0 računaj x koordinatu sjecišta ispitne linije y0 i *i*-tog brida:

$$x1 = (-b(i) * y0 - c(i)) / a(i)$$

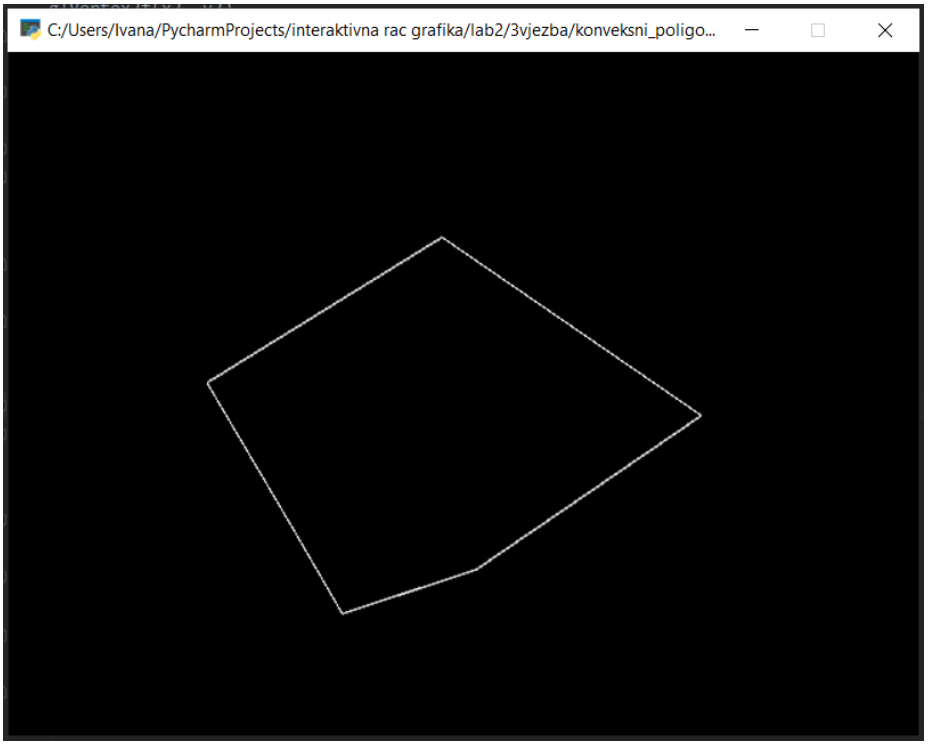
Ako je $y(i) < y(i + 1)$ i $x1 > L$ onda $L = x1$.

Ako je y(i) >= y(i + 1) i x1 < D onda D = x1.

Za svaki L i D, ukoliko je L manja od D crta se linija od točaka (L, y) i (D, y) i tako se iscrta poligon (Slika 3.)



Slika 2: iscrtan konveksni poligon



Slika 3: obojan konveksni poligon

VJEŽBA 4: Ravnina, tijelo

Cilj četvrte vježbe je iz datoteke tipa *.obj* učitati tijelo i to tijelo nacrtati pomoću poligona.

Iz datoteke je trebalo učitati broj vrhova i poligona te nakon toga i same vrhove ii poligone. Trebalo je odrediti i minimalne i maksimalne koordinate točaka x, y, z, odrediti središte tijela i postaviti ga u ishodište te tijelo skalirati na raspon [-1,1] kako bi stalo u ekran i odrediti koeficijente jednadžbe ravnine za svaki učitani poligon koje tijelo sadrži. Na kraju za zadanu točku V odrediti je li unutar ili van konveksnog tijela i iscrtati tijelo na ekran. Programski zadatak napisan je u programskom jeziku Python i spremljen kao ravnina_tijelo.py.

Nakon učitavanja vrhova i poligona iz datoteke računaju se koeficijenti za svaki poligon i spremaju se u listu koeficijenata, po formuli:

$$a = (y2-y1)(z3-z1) -(z2-z1)(y3-y1)$$
$$b = -(x2-x1)(z3-z1) + (z2-z1)(x3-x1)$$
$$c = (x2-x1)(y3-y1) - (y2-y1)(x3-x1x)$$
$$d = -x1*a - y1*b - z1*c$$

nakon izračunatih vrijednosti raspona x, y, z računamo središte tijela i maksimalni raspon:

$$središte = ((xmin+xmax)/2, (ymin+ymax)/2, (zmin+zmax)/2)$$
$$max\_raspon = max(xmax-xmin, ymax-ymin, zmax-zmin)$$

pomoću kojih se tijelo skalira na zadani raspon tako da svakoj koordinati oduzmemo vrijednost te koordinate središta i pomnožimo s 2/max_raspon.

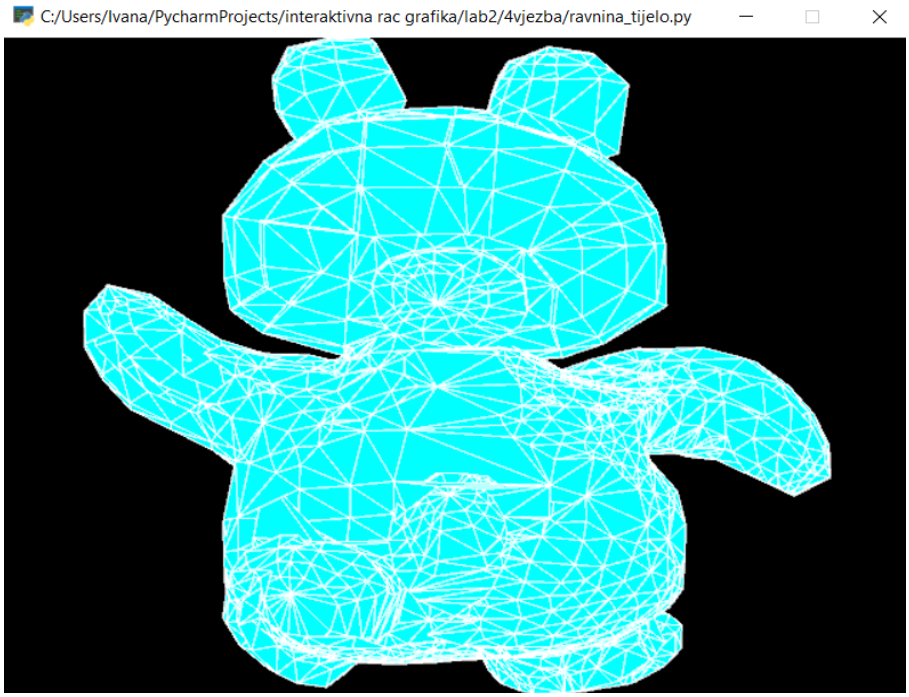
Provjera je li zadana točka V unutar ili van tijela je zapravo provjeravanje zadovoljivosti formule:

$$x * a(i) + y * b(i) + z*c(i) + d(i) > 0$$

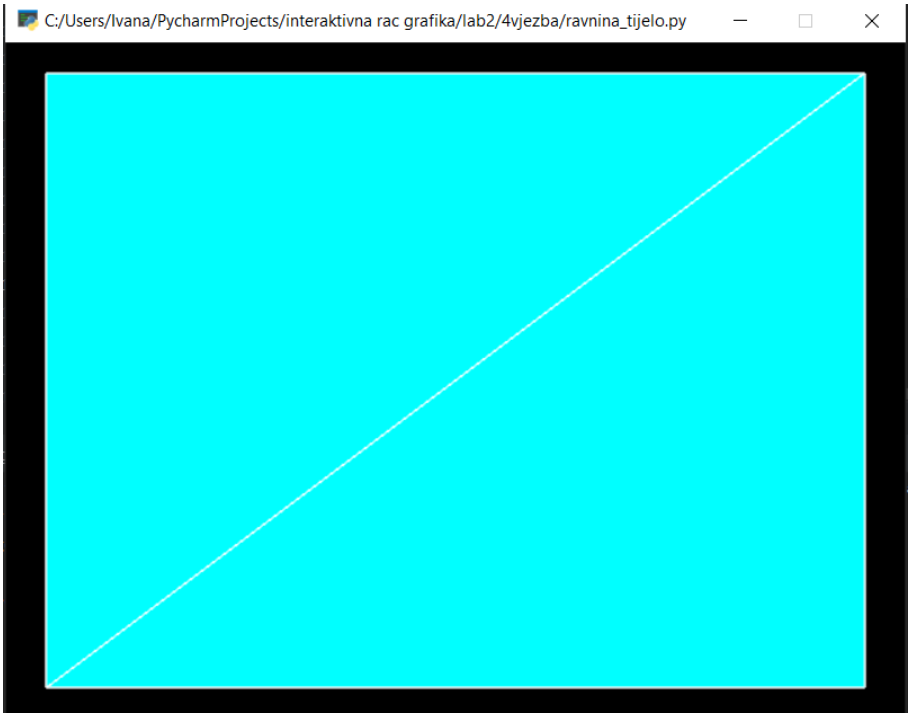
Koja za svaku ravninu provjerava ovu nejednakost i ako je istinita za samo jednu točka je van tijela, inače je unutar.

Nakon svega aktivira se metoda *on_draw* iz *Pyglet* biblioteke koja postavlja poziva potprogram *postavi_parametre* koji postavlja veličinu prozora, projekciju, početnu boju i druge parametre.

u metodi se ide po svim parametrima koji se crtaju metodom *graphics.draw*(GL_TRIANGLE_FAN) iz biblioteke *Pyglet*. Iscrtano tijelo je predloženo na (slika 4.).



Slika 4. teddy.obj



Slika 5. kocka.obj

VJEŽBA 5: Transformacija pogleda i perspektivna

U ovoj vježbi trebalo je za zadani konveksni poligon iz prethodne vježbe načiniti transformaciju pogleda i perspektivnu projekciju koje se vrše pomoću niza operacija nad matricama. Učitavanje datoteke jednako je učitavanju u prošlom zadatku. Nakon što se tijelo učitalo na zaslon se ispisuje njegov raspon kako bi korisniku koji pute tipkovnice unosi x, y, z koordinate odvojene zarezom gledišta i očišta bilo lakše. Moramo paziti kako se koordinate očišta ne stave unutar tijela (ispisom na ekran to pokušavamo izbjeći). Nakon učitanih vrhova poziva se metoda *primjeni_transformaciju* koja zatim poziva metodu *matrica_transformacijee_pogleda_T* koja čini niz operacija nad matricama kako bi načinila matricu pogleda T. operacije su redom pomak u ishodište (T1,), rotacija za kut alpha oko z-osi (T2), rotacija za kut beta oko y-osi (T3), rotacija za 90 stupnjeva oko z-osi (T4), promjena predznaka na x-os te se u zadnjem koraku sve navedene matrice objedine u jednu T matricu.

$$\mathbf{T_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & -y_0 & -z_0 & 1 \end{bmatrix} \quad \mathbf{T_2} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_3 = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

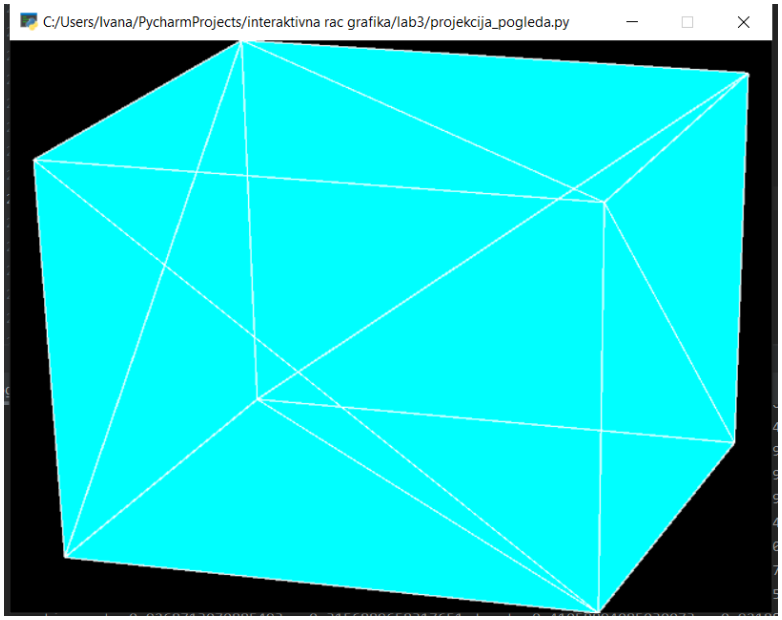
$$T_4 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_5 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Nakon toga dobiva se matrica transformacije projekcije P i ona izgleda ovako:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{H} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Na svakom vrhu obavimo transformaciju pogleda i projekcije te ga skaliramo na raspon [-1,1] kao u prethodnoj vježbi i pomoću window eventa *on_draw* odvija se crtanje objekta (opisano u prethodnoj vježbi). Pomoću metode *on_key_press*

iz *Pyglet* biblioteke omogućeno je pritiskom tipki O, I, P, L, G, F, B i V pomicanje koordinata gledišta i očišta za 0.01 i prikaz se ažurira na ekranu. Prikaz izvođenja programa je na (Slika 6.).



Slika 6. transformacija pogleda i projekcije nad kocka.obj

VJEŽBA 6: Prikaz prostornih krivulja postupkom Beziera

Zadatak je bio iz datoteke učitati n+1 točku kontrolnog poligona i iscrtati poligon te Bezier-ovim postupkom nacrtati krivulju koja najbolje aproksimira zadane točke. Zadnji korak je učitati tijelo iz rethodnih vježbi (način već poznat) i očište pomicati po dobivenim točkama Bezier-ove krivulje te s tim podatcima napraviti animaciju, uz uklanjanje stražnjih poligona) kojom će se vizualizirati mijenjanje očišta i kakve promijene to donosi nad objektom. Programski zadatak napisan je u programskom jeziku *Python* i nazvan *krivulja.py*.

U ovom programskom zadatku novost je metoda *draw_bezier* koja za svaki vrh računa težinske funkcije Bezier-a gdje za svaki *t* iz intervala [0, 1] s korakom 0,01 određuje koordinate x(t), y(t) i z(t) krivulje Beziera: b = *bin_umnozak*(n,i) * ti * (1-t)n-i s kojim pomnožimo sve koordinate vrha:

x += vrh.x * b

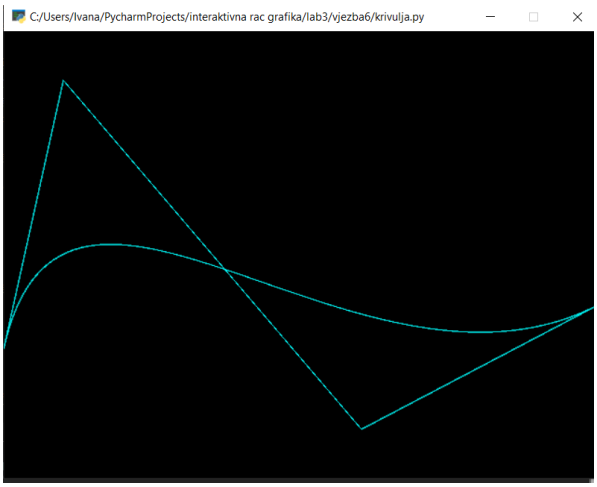
y += vrh.y * b

z += vrh.z * b

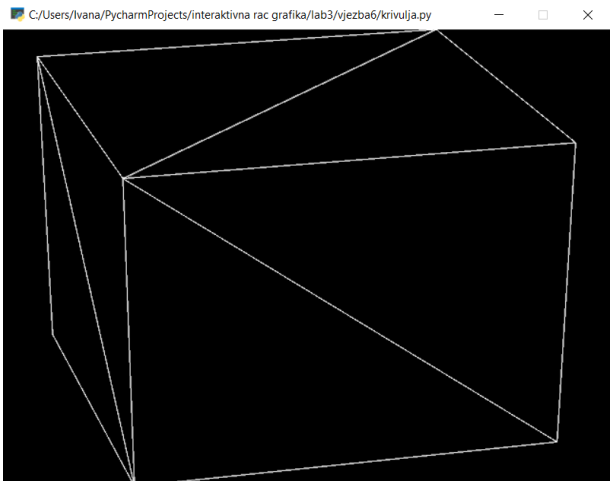
Animacija je postignuta metodom *update* koja je postavljena da ažurira novu vrijednost očišta svaku sekundu i zatim metoda *on_draw* ažurira prikaz na ekranu. (Slika 7.) prikazuje Bezier-ovu krivulju sa zadanim točkama vrhova, a (slika 8.) prikaz animacije. Uklanjanje stražnjih poligona obavlja se u metodi nacrtaj1 uz provjeru uvjeta

a*ociste.x + B+ociste.y + c*ociste.z + d < 0

za svaki poligon koji se želi iscrtati. Ukoliko je uvjet zadovoljen poligon je stražnji, izlazi se iz metode te se taj poligon ne iscrtava.



Slika 7. Bezier-ova krivulja



Slika 8. animacija objekta

VJEŽBA 7: Sjenčanje tijela

Zadatak 7.vježbe je učitati tijelo(postupkom opisanim u 4. vježbi) te nad njime načiniti transformaciju pogleda i perspektivnu projekciju (opisane u vježbi 5) te skalirati tijelo u radni prostor [-1,1] uz uklanjanje stražnjih poligona i prikaz tijela(6. vježba). Novost je zadavanje položaja, tj. Koordinata izvora svjetlosti te na osnovi normala poligona i vektora prema izvoru L iz središta poligona odrediti intenzitete poligona (prvi dio) i intenzitete u vrhovima (drugi dio).

Prvi dio zadatka bio je tijelo konstantno osjenčati i prikazati ga tako da su sva tri vrha poligona istog intenziteta. Zadatak je napisan u programskom jeziku *Python* i nazvano *k_sjencanje.py*.

Učitavanje, skaliranje, transformiranje i projiciranje tijela te uklanjanje stražnjih poligona implementirano je na isti način kao u programu u 6.vježbi. Sjenčanje je ostvareno u potprogramu *izracunaj_intenzitet* koji zao argumente prima središte zadanog poligona i koeficijente te ravnine a, b, c, d. Intenzitet I računa se prema formulama u uputama za 7. vježbu:

$$Id = Ii * kd * \max(N * L, 0)$$

$$Ig = Ia * ka$$

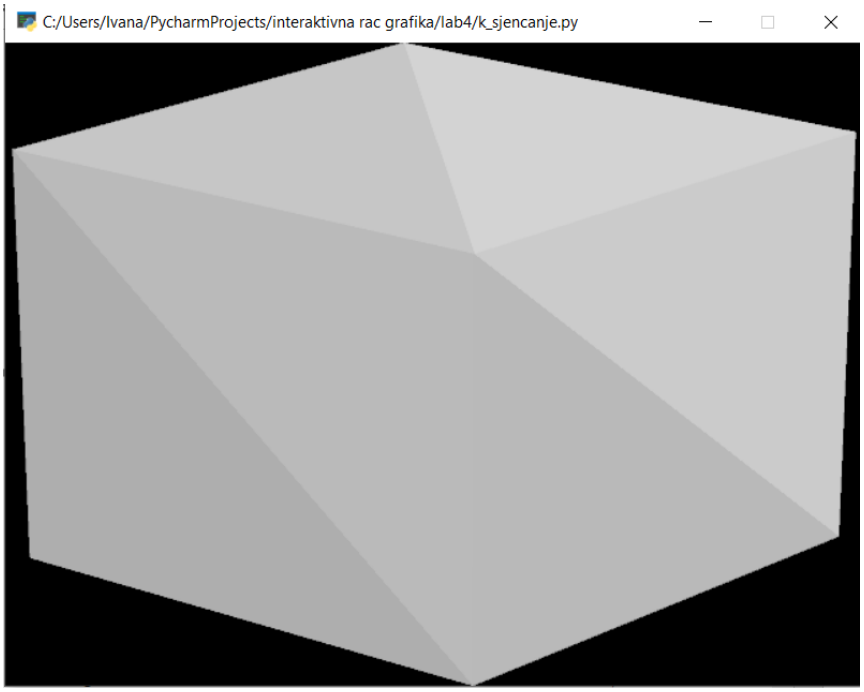
$$I = Ig + Id$$

gdje su Ia, ka, Ii i kd proizvoljno zadani argumenti. Ig je ambijentalna komponenta, a Id difuzna komponenta. L je vektor prema izvoru iz središta poligona, a N normala poligona. Primjer konstantnog sjenčanja prikazano je na (slika 9.).

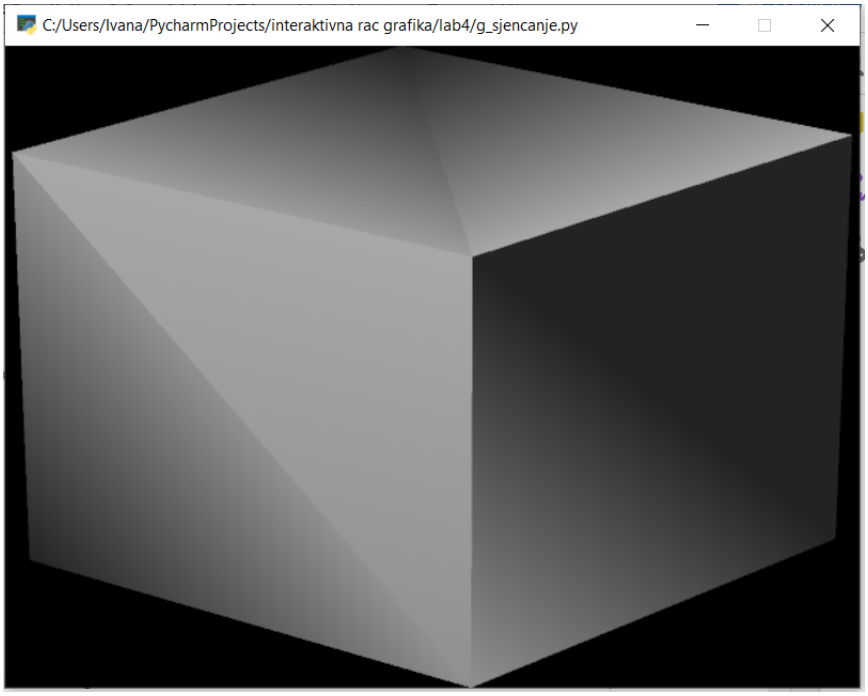
Drugi dio vježbe bio je odrediti normale svim vrhovima te na temelju njih i vektora iz vrhova prema izvoru L odrediti intenzitete u svakom vrhu.

Taj postupak zove se Gouraud-ovo sjenčanje (slika 10.), napisano je u programskom jeziku *Python* i pohranjeno je u datoteku *g_sjencanje.py*. Sjenčanje je ostvareno u potprogramima nacrtaj i *izracunaj_intenziret*. U prvo spomenutom potprogramu računaju se normale za svaki vrh unutar poligona spremljen kao klasa N_normala koja čuva koeficijente normale za vrhove pojedinog poligona. Računaju se i vektori prema izvoru iz svakog vrha tog

poligona L1, L2, L3 te se oni također spremaju kao klasa *N_*normala. Obje varijable su ulazni parametri potprograma *izracunaj_intenzitet* u kojima se za svaki vrh posebno izračunava intenzitet I po gore napisanoj formuli.



slika 9. Konstantno sjenčanje

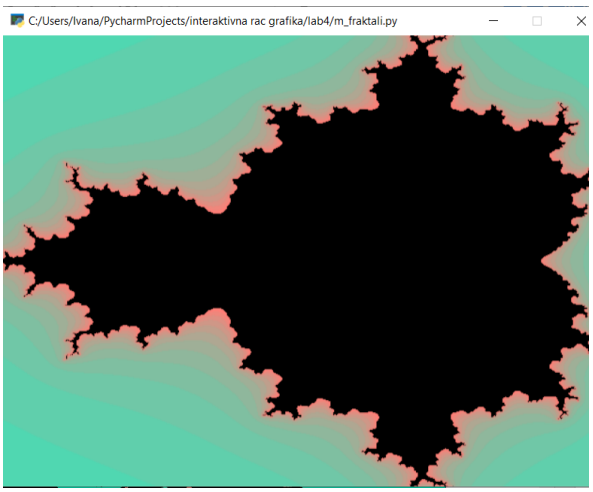


slika 10. Gouraud-ovo sjenčanje

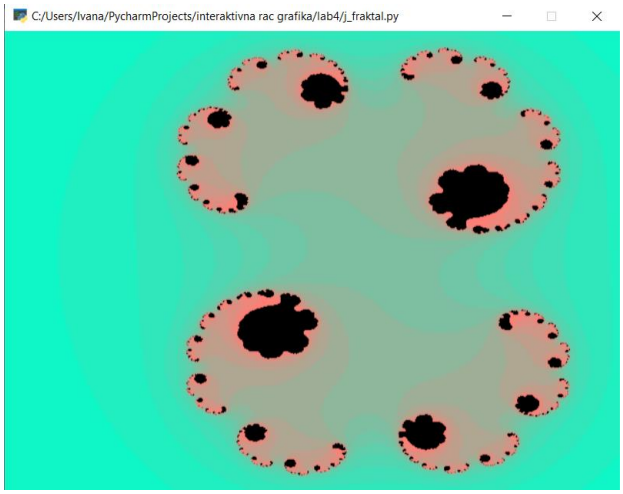
VJEŽBA 8: Fraktali

Zadatak 8. vježbe bio je slijediti postupak stvaranja i prikaza Mandelbrotovog i Julijevog fraktalnog skupa. Postupak koji je slijeđen u programskom kodu je postupak dan pseudokodom u pripremi ove vježbe. Programski kod napisan je u programskom jeziku Python i podijeljen je u dva dijela m_sjencanje.py i j_sjencanje.py. Mandelbrotov fraktal prikazan je na (slika 11) te je

Mandelbrotov skup je skup točaka u kompleksnoj ravnini i njegovu granicu nazivamo Mandelbrotovim fraktalom. U potprogramu *divergira_li* krećemo se od 0 za 1 po limitu kojeg smo zadali i u kompleksnoj rekurzivnoj funkciju u kojoj iznova računamo vrijednost parametra z koji je spremljen u obliku klase C koja pamti realan i imaginaran dio broja, uz početni uvjet z = C(0,0).
ovime provjeravamo je li modul(z) ograničen, ako pustimo da *n* teži u beskonačnost. provjeravamo uvjet modul(z) > *eps,* ako je on zadovoljen tada vraćamo koeficijent *k* kojim bojimo granicu skupa tj. fraktal, bojanje se odvija u window eventu *on_draw*. Ukoliko uvjet nije zadovoljen kompleksni broj c pripada Mandelbrotovom skupu.
Dok je Julijev skup skup točaka u kompleksnoj ravnini i njegovu granicu nazivamo Julijevim fraktalom. U potprogramu *divergira_li* krećemo se od 0 za 1 po limitu kojeg smo zadali i u kompleksnoj rekurzivnoj funkciju u kojoj iznova računamo vrijednost parametra z koji je spremljen u obliku klase C koja pamti realan i imaginaran dio broja, uz promjenu početnog uvjeta z, tj. on se određuje u svakoj točki ekrana z = C(x,y). Epsilon vrijednost određena je s max(c.re + c.im, 2).
Zadovoljivost uvjeta i crtanje obavlja se na isti način kao i u prošlom primjeru.



Slika 11. Mandelbrotov fraktal



Slika 12. Julijev fraktal