Interaktivna računalna grafika.

Zadatci za laboratorijske vježbe u modernom OpenGL-u.

Marko Čupić Željka Mihajlović Hrvoje Nuić

29. ožujka 2021.

Sadržaj

Predgovor

Ovaj dokument predstavlja radnu verziju novih uputa za laboratorijske vježbe iz kolegija Interaktivna računalna grafika: *Interaktivna računalna grafika. Zadatci za laboratorijske vježbe u Modernom OpenGL-u*. Molimo sve pogreške, komentare, nejasnoće te sugestije dojaviti na Marko.Cupic@fer.hr, Zeljka.Mihajlovic@fer.hr ili Hrvoje.Nuic@fer.hr.

Verzija dokumenta: 0.1.2021-03-05.

Uvod

Laboratorijske vježbe sastavni su dio izučavanja gradiva kolegija interaktivne računalne grafike. Praktičnim radom u laboratoriju usvaja se, utvrđuje i proširuje znanje računalne grafike. Ujedno tim se putem stiče osjećaj težine pojedinih postupaka. Osnovne ideje temeljnih postupaka potrebno je razumjeti i praktičnim putem usvojiti, kako bi temeljni princip bio upotrebljiv i prilikom rješavanja problema u nekom drugom okruženju. Gotovo u svakom postupku postoje i posebni slučajevi o kojima je potrebno posebno voditi računa, a taj dio se uglavnom ostavlja kao dodatni izazov istraživanja.

Izabrani dijelovi gradiva uključeni su u laboratorijske vježbe. Prvenstveno, tu se radi o:

- grafičkim primitivima
- geometrijskim izračunavanjima
- transformacijama i projekcijama
- izračunavanju osvjetljenja
- postupcima sjenčanja
- skrivenim linijama i površinama
- postupcima interpolacije prostorne krivulje
- preslikavanju tekstura na objekte
- ostvarivanju sjena

U svakoj vježbi potrebno je izraditi radni zadatak, odnosno radni program. Uz radni zadatak, u obliku podsjetnika, sažeto je ponovljeno nastavno gradivo. Također je naznačeno moguće rješenje radnog zadatka, u obliku postupka, tj. specifikacije radnog programa, te očekivani rezultati izvođenja radnog programa.

Cilj je da na kraju laboratorijskih vježbi imate dobru podlogu i pregled područja kako biste mogli samostalno ostvariti napredniji program za rad s računalnom grafikom, te da znate što se događa u pozadini prilikom korištenja programa poput Unity, Blender ili Autocad koji u svojoj pozadini koriste brojne algoritme i koncepte iz računalne grafike.

Svaka laboratorijska vježba sadrži i dodatne zanimljive zadatke koje nije potrebno rješavati, ali mogu dobro poslužiti za uvježbavanje.

Predložak i upute za početak

Provjerite podržava li Vaše računalo OpenGL verziju veću od 3.3. Ne zaboravite instalirati drivere za Vašu grafičku karticu. Predložak laboratorijske vježbe se nalazi u git repozitoriju na adresi: `https://gitlab.com/irgtim/irglab`. Molimo koristite navedeni predložak po uputama.

Za uspješno postavljanje projekta, potrebno je instalirati git[1] sustav za verzioniranje kôda i u željenom direktoriju pokrenuti naredbu:

```
git clone --recursive https://gitlab.com/irgtim/irglab.git
```

[1] `https://git-scm.com/`

Primijetite korištenje *recursive* opcije kako bi se dohvatile i biblioteke koje ćemo koristiti za izradu laboratorijskih vježbi. Ako ste već klonirali repozitorij bez *recursive* opcije, repozitorij možete popraviti pokretanjem naredbe:

```
git submodule update --init --recursive
```

Potom je potrebno instalirati cmake[2] ili neki drugi odgovarajući alat. Za windows gui verziju cmakea, pokrenite generiranje projekta za željeno razvojno okruženje tako da pod *source* direktorij odaberete korijenski direktorij repozitorija ".../irgLabosi/", a pod *build* direktorij napravite novi direktorij ".../irgLabosi/build/". Pritisnuti *Configure*, odabrati željeni generator projekta (npr. Visual Studio 15 2017 Win64), pritisnuti *Finish* te potom *Generate*. Cmake će na temelju konfiguracije napraviti projekt za odabrano razvojno okruženje.

Ako ste odabrali Visual Studio kao razvojno okruženje, otvoriti projekt *irgLabosi.sln*. Desnim klikom na *vjezba1* otvoriti izbornik i pritisnuti *Set as StartUp Project*. Pokrenuti izgradnju vjezbe1. Prilikom prvog pokretanja izgrađuju se i biblioteke, pa može potrajati nešto duže.

Biblioteke koje se koriste u okviru laboratorijskih vježbi su:

- GLFW[3] – Jednostavan API za izradu prozora, OpenGL konteksta i rukovanje korisničkim ulazom/izlazom.
- glad[4] – Određivanje koje su sve OpenGL funkcionalnosti dostupne programeru.
- GLM[5] (OpenGL Mathematics) – Olakšava rad s vektorima i matricama.
- assimp[6] (Open Asset Import Library) – Učitavanje različitih formata za opis scene i objekata.
- stb[7] - Učitavanje tekstura objekata.

Napomene

Za otkrivanje grešaka u programskom kodu vezano uz pozivanje OpenGL funkcionalnosti možete koristiti alate poput RenderDoc[8] ili NVIDIA Nsight[9].

Nove datoteke koje planirate dodati u projekt uvijek postavljajte u odgovarajući direktorij projekta unutar "../irgLabosi/irgLabosi/" direktorija, a ne unutar *build* direktorija. Takav način organizacije projekta zove se *out of source tree*, a izbjegava miješanje programskog koda, konfiguracijskih i izvršnih datoteka projekta. Visual studio ne podržava *out of source tree* organizaciju, pa ćete morati ručno dodavati nove datoteke u projekt.

Pomoću Cmake datoteke definirani su događaji koji se izvršavaju nakon izgradnje projekta, a prije pokretanja izvršne datoteke. Takvi događaji se nazivaju *post build* događaji. Za predložak su definirani *post build* događaji za kopiranje resursa (na primjer .png, .obj datoteke) i kopiranje sjenčara (na primjer .vert, .geom, .frag) u *build* direktorij.

Ako prilikom izrade laboratorijskih vježbi između dva pokretanja programa niste promijenili datoteke vezane uz izvorni kod, neće se izvršiti *post build* događaji (u Visual Studiju). Sjenčari i resursi nisu dio izvornog koda projekta, pa ako samo njih promijenite između dva pokretanja, **neće se kopirati u izvršni direktorij**.

[2] https://cmake.org/
[3] https://www.glfw.org/
[4] https://glad.dav1d.de/
[5] https://glm.g-truc.net/0.9.9/index.html
[6] https://assimp.org/
[7] https://github.com/nothings/stb
[8] https://renderdoc.org/
[9] https://developer.nvidia.com/nsight-graphics

Inicijalno, predložak je napravljen u Windows 10 operacijskom sustavu, Microsoft Visual Studio 19 razvojnom okruženju i cmake 3.13. alatu, no kompatibilan je i testiran i na linuxu s različitim razvojnim okruženjima.

Osnovne matematičke operacije u računalnoj grafici

Osnovna matematička podloga interaktivne računalne grafike je linearna algebra i analitička geometrija, pa kako bi uspješno savladali laboratorijske vježbe, potrebno je u predlošku proučiti primjer rada s *glm* bibliotekom i ukratko se podsjetiti matematičke podloge dane u idućim potpoglavljima.

Operacije s vektorima

Za zadana dva vektora $\vec{v}_1 = (x_1, y_1, z_l)$ i $\vec{v}_2 = (x_2, y_2, z_2)$ definiran je njihov zbroj i razlika.

$$\begin{aligned} \vec{v}_1 + \vec{v}_2 &= (x_1 + x_2, y_1 + y_2, z_1 + z_2) \\ \vec{v}_1 - \vec{v}_2 &= (x_1 - x_2, y_1 - y_2, z_1 - z_2) \end{aligned} \tag{1}$$

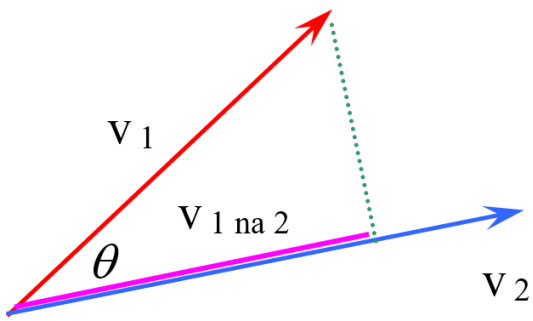
Skalarni produkt $\vec{v}_1 = (x_1, y_l, z_l)$ i $\vec{v}_2 = (x_2, y_2, z_2)$ računa se kao zbroj umnožaka pojedinih komponenti, a rezultat je skalar:

$$\vec{v}_1 \cdot \vec{v}_2 = x_1 * x_2 + y_1 * y_2 + z_1 * z_2 \tag{2}$$

Skalarni produkt vektora koristi se kod izračuna kuta θ između dva vektora prema izrazu:

$$cos(\theta) = \frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1||\vec{v}_2|} = \frac{x_1x_2 + y_1y_2 + z_1z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2}\sqrt{x_2^2 + y_2^2 + z_2^2}} \tag{3}$$

gdje su u nazivniku moduli pojedinih vektora, odnosno njihove duljine. Vrlo korisno je znati kako odrediti projekciju jednog vektora na drugi. Projekcija $\vec{v}_1 = (x_1, y_l, z_l)$ na $\vec{v}_2 = (x_2, y_2, z_2)$ je $\vec{v}_{1na2}$.



Slika 1: Projekcija $\vec{v}_1$ na $\vec{v}_2$.

Kod projekcije jednog vektora na drugi možemo promatrati skalarnu vrijednost, odnosno duljinu projiciranog $\vec{v}_1$ na $\vec{v}_2$.

$$|\vec{v}_{1na2}| = cos(\theta)|\vec{v}_1| = \frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_2|} \tag{4}$$

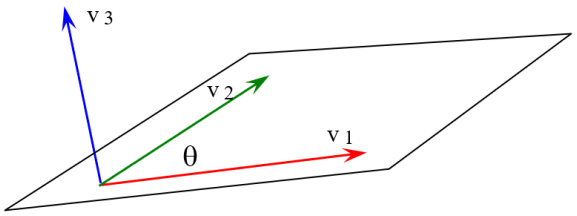
do čega dolazimo iz izraza 3 i slike 1 uz interpretaciju definicije kosinusa kuta kao omjera prileže stranice trokuta ($\vec{v}_{1na2}$) i hipotenuze trokuta ($|\vec{v}_1|$. Znači, projekciju jednog vektora na drugi možemo odrediti iz poznavanja tih vektora ili kuta i vektora kojeg projiciramo. Ako nam je potreban vektor koji predstavlja projekciju $\vec{v}_1$ na $\vec{v}_2$ prethodno dobiveni rezultat ćemo pomnožiti jediničnim vektorom u smjeru vektora $\vec{v}_2$.

$$\vec{v}_{1na2} = cos(\theta)|\vec{v}_1|\frac{\vec{v}_2}{|\vec{v}_2|} = \frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_2|}\frac{\vec{v}_2}{|\vec{v}_2|} \tag{5}$$

Vektorski produkt $\vec{v}_1 = (x_1, y_l, z_l)$ i $\vec{v}_2 = (x_2, y_2, z_2)$ rezultira vektorom i definiran je

$$\vec{v}_1 \times \vec{v}_2 = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_1 & y_1 & h_1 \\ x_2 & y_2 & h_2 \end{bmatrix} = \begin{bmatrix} y_1h_2 - y_2h_1 \\ -x_1h_2 + x_2h_1 \\ x_1y_2 - x_2y_1 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{6}$$

Geometrijska interpretacija vektorskog produkta dva vektora je vektor koji je okomit na ravninu u kojoj leže $\vec{v}_1$ i $\vec{v}_2$. Važan je i redoslijed ova dva vektora u vektorskom produktu a definiran je po pravilu desne ruke. Ako zakretanje $\vec{v}_1$ prema $\vec{v}_2$ određuju prsti desne ruke, palac je u smjeru rezultantnog $\vec{v}_3$.



Slika 2: Vektorski produkt $\vec{v}_1$ i $\vec{v}_2$ daje $\vec{v}_3$.

Operacije s matricama

U računalnoj grafici transformirat ćemo vrhove objekta koji su zadani kao jedno-redčane matrice 1×3 ili 1×4. Matrice transformacija zadaju se kao matrice 3×3 ili 4×4. Za to će nam biti potrebno množenje matrica. Također, transformacije mogu biti zadane kao niz matrica koje je potrebno množiti. Kod inverznih transformacija ponekad se koriste i inverzne matrice te je potrebno imati i ovu funkcionalnost za matrice 3×3 ili 4×4.

Baricentrične koordinate

Baricentrične koordinate korisne su kod određivanja nalazi li se točka u trokutu u 3D prostoru, a može poslužiti i prilikom interpolacije vrijednosti po trokutu. Ako su vrhovi trokuta A, B, C. Za neku točku T imamo baricentričnu kombinaciju:

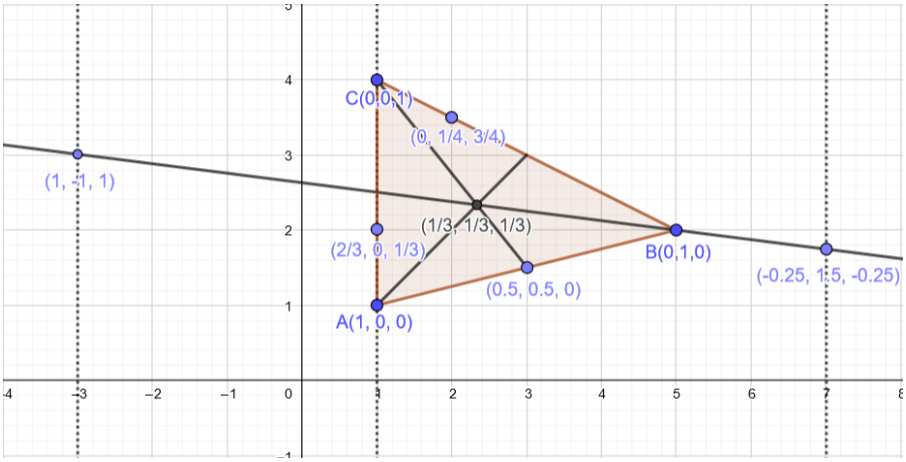
$$T = t_1A + t_2B + t_3C \tag{7}$$

gdje su t_1, t_2, t_3 baricentrične koordinate. Za baricentričnu kombinaciju vrijedi $t_1 + t_2 + t_3 = 1$. Ovo možemo prikazati kao sustav jednadžbi, raspisan po koordinatama:

$$\begin{aligned} A_x t_1 + B_x t_2 + C_x t_3 &= T_x \\ A_y t_1 + B_y t_2 + C_y t_3 &= T_y \\ A_z t_1 + B_z t_2 + C_z t_3 &= T_z \end{aligned} \tag{8}$$

ili matrično: $\begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ A_z & B_z & C_z \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$, pa je rješenje $\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ A_z & B_z & C_z \end{bmatrix}^{-1} \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$.

Znači, da bi odredili baricentrične koordinate bit će potrebno invertirati matricu. Može se desiti da matrica nije invertibilna. To je na primjer slučaj 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. Tada je matrica singularna i ne možemo ju invertirati. U tom slučaju koristimo uvjet koji smo naveli da vrijedi za baricentrične koordinate $t_1 + t_2 + t_3 = 1$ umjesto retka matrice koji je problematičan.



Slika 3: Primjer baricentričnih koordinata različitih točaka

Slika 3 može poslužiti za dobivanje bolje intuicije ovisnosti baricentričnih koordinata o kartezijevim koordinatama, gdje je baricentrični koordinatni sustav definiran s $\triangle ABC$. Točke $A(1,1)$, $B(5,2)$ i $C(1,4)$ imaju baricentrične koordinate $A(1,0,0)$, $B(0,1,0)$ i $C(0,0,1)$.

Homogene koordinate

Točka iz n-prostora može biti preslikana u homogenu točku u (n+1) h-prostoru. Obrnuto, homogena točka iz (n+1) h-prostora može biti projicirana u točku n-prostora. Homogene koordinate nam služe kako bi lakše zapisali i primijenili nad vektorima afine i projektivne transformacije. Promotrimo za primjer 2-prostor i njemu odgovarajući homogeni 3h-prostor.

Preslikavanje točke V u 2-prostoru u točku V' u 3h-prostoru se zapisuje kao $V(x,y) \to V'(x,y,h)$ pri čemu je

$$\begin{aligned} x' &= x * h \\ y' &= y * h \end{aligned} \tag{9}$$

dok se projekcija zapisuje kao $V'(x',y',h') \to V(x,y)$ pri čemu vrijedi:

$$\begin{aligned} x &= x'/h \\ y &= y'/h \end{aligned} \tag{10}$$

Komponenta h zove se faktor proporcionalnosti ili homogena koordinata. Vrijednost homogene koordinate h je proizvoljna, najčešće se koristi slučaj $h = 1$. Ako je $h = 0$ tada se radi o točki koja je

u beskonačnosti u n-prostoru. Ako su pravci paralelni u n-prostoru tada su paralelni i u homogenom (n+1) h-prostoru. Sačuvanost paralelnosti pravaca u homogenom prostoru važno je svojstvo.

Jednadžba pravca

Pravac je određen s dvije točke, na primjer točke V_1 i V_2. Koristi se homogeni prostor, tj. $V_1 = (x_1, z_l, h_l)$, $V2 = (x_2, y_2, h_2)$. Vektorski oblik jednadžbe pravca određen je vektorskim produktom

$$P = V_1 \times V_2 = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_1 & y_1 & h_1 \\ x_2 & y_2 & h_2 \end{bmatrix} = \begin{bmatrix} y_1h_2 - y_2h_1 \\ -x_1h_2 + x_2h_1 \\ x_1y_2 - x_2y_1 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{11}$$

Analitički oblik jednadžbe pravca, uz $h_1 = h_2 = 1$ je $\frac{y-y_1}{y_2-y_1} = \frac{x-x_1}{x_2-x_1}$ odnosno

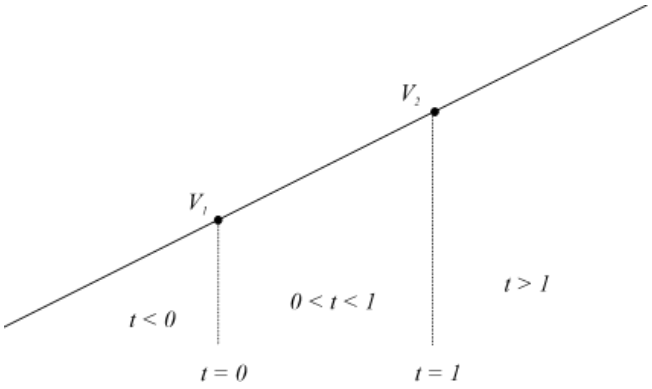
$$(y_1 - y_2)x + (-x_1 + x_2)y + x_1y_2 - x_2y_1 = ax + by + c = 0 \tag{12}$$

Parametarski oblik jednadžbe pravca je $P = (V_2 - V_1)t + V_1$, ili po koordinatama

$$\begin{aligned} x &= (x_2 - x_1)t + x_1, \\ y &= (y_2 - y_1)t + y_1, \\ h &= (h_2 - h_1)t + h_1. \end{aligned} \tag{13}$$

Pri tome je točki V_1 pridružena vrijednost parametra $t = 0$ a točki V_2, vrijednost parametra $t = 1$. Na slici pokazano je pridjeljivanje vrijednosti parametra t dijelovima pravca, koji su od interesa.

Na slici 4 pokazano je pridjeljivanje vrijednosti parametra t dijelovima pravca, koji su od interesa.



Slika 4: Vrijednost parametra t i dijelovi pravca.

Ispitivanje odnosa točke i pravca

Skalarni produkt točke $V(x, y, 1)$ i pravca $P[abc]^T$ određuje odnos točke i pravca, pri tome vrijedi dogovor: $V \cdot P = ax + by + c \begin{cases} > 0 \text{ točka } V \text{ je iznad pravca } P \\ = 0 \text{ točka } V \text{ je na pravcu } P \\ < 0 \text{ točka } V \text{ je ispod pravca } P \end{cases}$

Laboratorijska vježba 1

Crtanje linija na rasterskim prikaznim jedinicama

Programi za rad s računalnom grafikom najčešće barataju modelima koji su definirani točkama, linijama i poligonima. Programi modele moraju prikazati na ekranu u dvije dimenzije i to nakon što su ih transformirali na željenu poziciju u sceni i primijenili projekciju nad njima. Ekran na kojemu se prikazuje slika je diskretan, sastavljen od niza slikovnih elemenata i na njemu ne možemo beskonačno precizno nacrtati niti linije, niti krivulje niti apstraktne geometrijske likove. Umjesto toga, crtanje različitih pravaca, krivulja i likova u praksi se svodi na uporabu algoritama koji će brzo i efikasno na ekranu upaliti slikovne elemente koji će dati najbolju aproksimaciju linije koju je korisnik htio nacrtati.

Moderni OpenGL te algoritme za rasterizaciju linja i poligona skriva od programera jer su efikasno implementirani na grafičkoj kartici, no bitno je dobiti dobar osjećaj što se događa u pozadini. Kako bi Vam olakšali susret s računalnom grafikom, ostvaren je razred *Grafika* koji će Vam poslužiti za jednostavno osvjetljavanje fragmenata rastera.

U okviru ove vježbe Vaš je zadatak proučiti na koji se način na rasterskim prikaznim jedinicama mogu efikasno crtati linijski segmenti, te na koji je način moguće ograničiti crtanje na dio podprostora kojim prikazna jedinica raspolaže. Da biste riješili ovaj zadatak, proučite u knjizi poglavlje 4 i to potpoglavlje 4.1 koje opisuje Bresenhamov algoritam za crtanje linijskih segmenata, te unutar poglavlja 8 podpoglavlje 8.8 koje opisuje izvedbu odsijecanja algoritmom Cohen Sutherlanda. Također, dostupni su video materijali *IRG 4. Rasterska grafika* i minilekcije na adresi `https://ferko.fer.hr/minilessons/` koje pokrivaju gradivo laboratorijske vježbe.

1.1 Pitanja

1. Proučiti i podsjetiti se matematičke podloge dane u prethodnom poglavlju.
2. Proučite osnovnu verziju Bresenhamovog algoritma za crtanje linije.
   (a) Taj algoritam će korektno iscrtati liniju samo u slučaju da je nagib linije u rasponu od $0°$ do $45°$. Objasnite zašto?
   (b) Što će taj algoritam nacrtati ako se zada crtanje pravca određenog početnom točkom $(0, 0)$ i završnom točkom $(4, 12)$? Skicirajte rezultat. Objasnite zašto je rezultat takav?
   (c) Što bi taj algoritam nacrtao za pravac iz prethodne točke ako bismo redak u kojem se y-koordinata inkrementira zamijenili retkom u kojem se y-koordinata uvećava za izračunati tangens kuta pod kojim je zadan pravac? Skicirajte rezulata. Objasnite ga.
   (d) Vratimo se na osnovnu verziju algoritma gdje se y-koordinata uvećava za 1. Što će taj algoritam nacrtati ako se zada pravac pod kutem od -30, a što ako se zada pravac čiji je tangens nagiba jednak -3? Slicirajte to na konkretnom primjeru i objasnite rezultat.

3. Objasnite kako se izvodi Bresenhamov algoritam s decimalnim brojevima? Koja je osnovna prednost tog algoritma?
4. Objasnite na koji se način Bresenhamov algoritam s decimalnim brojevima prevodi na cjelobrojnu varijantu? Je li ta inačica povoljnija u odnosu na inačicu s decimalnim brojevima? Argumentirajte Vaš odgovor.
5. Objasnite kako radi algoritam za odsijecanje linija?

1.2 Zadatak

Programi za rad s računalnom grafikom poput *game enginea* uglavnom slijede strukturu programa danu idućim pseudokodom:

1. Definiraj osnovne parametre za rad s OpenGLom poput postavljanja visine i širine prozora, definiraj funkcije za prihvat korisničkih akcija i slično.
2. ponavljaj sve dok korisnik ne završi s radom programa:
   (a) Obradi korisničke naredbe (poput pritiska tipkovnice ili miša).
   (b) Osvježi pozicije objekata unutar scene.
   (c) Obriši platno prozora (prekrij sve jednom bojom).
   (d) Iscrtaj osvježenu scenu.
   (e) Pričekaj (16ms za 60FPS).
3. Oslobodi zauzetu memoriju i uništi prozor.

U predlošku, u sklopu *vježbe1* je implementiran prikaz plohe s uzorkom šahovske ploče i dohvaćanje korisničkih naredbi mišem preko prethodno navedene strukture programa. Koristeći predložak i razred *Grafika* za crtanje, modificirajte *vježbu1* tako da korisniku omogućite crtanje proizvoljnog broja linija. Korisnik treba moći linije zadavati mišem – prvi klik definira početak segmenta a drugi kraj segmenta; u tom trenutku segment se dodaje u listu definiranih segmenata. Svi segmenti iscrtavaju se modrom bojom. Prilikom crtanja konačne slike na ekranu, program treba konzultirati pomoćnu *boolean* varijablu odsijecanje.

Varijabla odsijecanje omogućava definiranje podprostora u kojem se iscrtavaju linije. Ako je varijabla odsijecanje==**false**, linije se iscrtavaju na čitavoj površini prozora. Ako je odsijecanje==**true**, aktivira se podprostor širine pola prozora i visine pola prozora koji je centriran u prozoru. Vaša implementacija Bresenhamovog algoritma treba provjeravati je li definiran podprostor u kojem se iscrtava, te ako je, treba iscrtati samo dio segmenta koji se nalazi unutar podprostora. To treba implementirati na način da se segmenti koji su u cijelosti izvan podprostora uopće ne crtaju (naprosto se odbace), a u suprotnom se računa početak i kraj podsegmenta koji je u cijelosti u tom podprostoru i samo se to iscrtava. Ako je ovaj podprostor aktivan, tada se kao prvi korak u iscrtavanju slike na praznu površinu prozora, zelenom bojom iscrta rub tog područja. Ako korisnik pritisne desni gumb miša, program treba invertirati trenutnu vrijednost varijable odsijecanje (i osvježiti prikaz).

Dodatni napredniji zadaci za vježbu

- Mijenjati boju linije ovisno o omjeru udaljenosti između početne i krajnje točke linije.
- Izmijeniti algoritam Bresenhama tako da sve varijable postanu cjelobrojne kako bi se izbjegle sporije operacije s realnim brojevima.
- Ovakvim crtanjem je linija „nazubljena“. Izmijeniti algoritam tako da su prijelazi između redaka blaži (aliased i antialiased crtanje).
- Proučiti na koji način razred Grafika prikazuje raster na ekranu.

Laboratorijska vježba 2

Crtanje i popunjavanje poligona

Crtanje i popunjavanje poligona možda je i najčešći zadatak u računalnoj grafici. Danas je ta operacija direktno prisutna u obliku sklopovske implementacije svih modernih grafičkih kartica. Kada govorimo o poligonima, postoji nekoliko zadataka koji mogu biti zadani.

- Crtanje poligona – temeljem zadanih vrhova poligona treba nacrtati poligon; pri tome se misli samo nacrtati obrub poligona, tj. linijama spojiti susjedne vrhove. Ovo je u OpenGL-u direktno podržano preko primitiva GL_LINE_LOOP.
- Popunjavanje poligona – temeljem zadanih vrhova poligona treba popuniti poligon, odnosno njegovu unutrašnjost ispuniti zadanom bojom. Ovo je u OpenGL-u direktno podržano preko primitiva GL_POLYGON.
- Ispitivanje smjera u kojem su zadani vrhovi poligona – u smjeru kazaljke na satu ili u smjeru suprotnom od smjera kazaljke na satu.
- Ispitivanje vrste poligona – je li poligon konveksan ili konkavan.
- Pronalaženje konveksnog poligona – uz niz vrhova zadanih proizvoljnim poretkom traži se pronalaženje poretka vrhova tako da budu u zadanom smjeru (primjerice, u smjeru kazaljke na satu).
- Ispitivanje odnosa točke i poligona – temeljem zadanog poligona i zadane proizvoljne točke potrebno je utvrditi je li zadana točka unutar poligona, na bridu poligona ili izvan poligona.

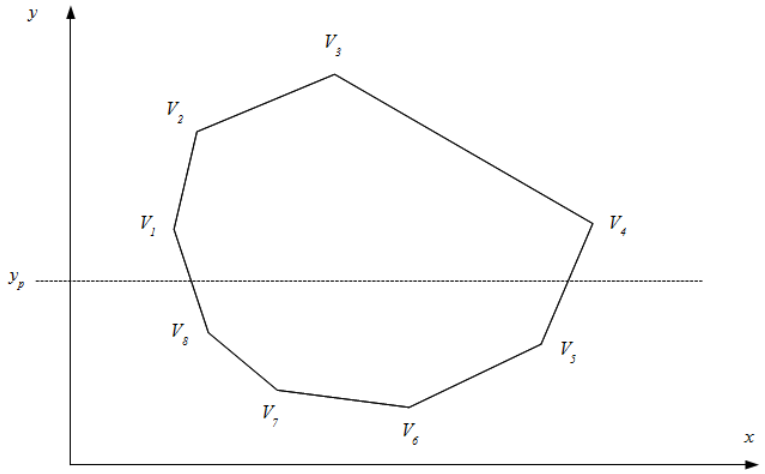
U okviru ove vježbe, pozabavit ćemo se nekim od tih pitanja. Kako bi se pripremili za vježbu, Vaš je zadatak pažljivo proučiti u knjizi poglavlje 4 i to potpoglavlje 4.2 koje daje matematički tretman poligona i opisuje niz postupaka koji služe crtanju poligona, popunjavanju poligona te otkrivanju različitih svojstava poligona. Također, dostupna je i minilekcija koja pokriva gradivo popunjavanja poligona.

2.1 Pitanja

1. Koja je razlika između konveksnog i konkavnog poligona?
2. Što znači da je redosljed vrhova u smjeru kazaljke na satu?
3. Kako se za zadani poligon može ispitati je li mu redosljed vrhova u smjeru kazaljke na satu ili suprotnom od smjera kazaljke na satu?
4. Neka je poligon zadan tako da su mu vrhovi zadani u smjeru kazaljke na satu. Kako možemo ispitati je li točka T unutar, izvan ili na nekom bridu poligona?

5. Neka je poligon zadan tako da ne znamo jesu li u smjeru kazaljke na satu ili u smjeru suprotnom od smjera kazaljke na satu. Ne radeći eksplicitnu provjeru načina na koji su bridovi zadani, kako možemo ispitati je li točka T unutar, izvan ili na nekom bridu poligona?

Sljedeća pitanja se odnose na bojanje poligona algoritmom iz knjige opisanim u poglavlju 4.2.4 i sliku 2.1.



Slika 2.1: Primjer poligona i postupak popunjavanja

1. Koliko će se presjecišta računati s pravcem $y = y_p$?

2. Koliko poligon ima lijevih bridova? Kako se definiraju lijevi bridovi?

3. Koliko poligon ima desnih bridova? Kako se definiraju desni bridovi?

4. Koliko će se puta ažurirati točka L a koliko puta točka D nakon što se postave na svoje inicijalne vrijednosti? Koje su uopće njihove inicijalne vrijednosti?

5. Pretpostavite da je poligon zadan tako da mu je redosljed vrhova suprotan od smjera kazaljke na satu. Kako bi se tada modificirao postupak bojanja?

2.2 Zadatak

Koristeći *vježbu2* kao podlogu i razred *Grafika* za crtanje, napišite program koji će korisniku omogućiti da mišem definira vrhove poligona i koji će potom korisniku prikazati bridove poligona, obojati unutrašnjost poligona, te omogućiti korisniku da zadaje točke koje će se obojati ovisno o pripadnosti poligonu.

Korisnik pritiscima lijevog gumba miša redom definira vrhove poligona. Korisniku se za to vrijeme iscrtavaju svi bridovi poligona pomoću Bresenhamovog algoritma iz prošle vježbe. Ako korisnik tijekom zadavanja točaka, doda novu točku koja bi napravila poligon konkavnim, potrebno je korisnika upozoriti i u konzolu ispisati odgovarajuću poruku. U trenutku kada korisnik pritisne desni gumb, prestaju se prikazivati bridovi i prikazuje se popunjeni poligon. Proučite u knjizi algoritam za popunjavanje konveksnih poligona, razmotrite kako taj algoritam radi i implementirajte ga. Daljnjim pritiskom na desni gumb, korisnik zadaje točke za koje je potrebno ispitati nalaze li se unutar ili izvan poligona. Program, ovisno o ispitivanju, osvjetljava fragment rastera na zadanoj poziciji crvenom bojom ako je izvan poligona ili zelenom ako je fragment unutar ili na rubu poligona. Algoritam za ispitivanje odnosa točke i poligona treba temeljiti na opisu danom u knjizi, u potpoglavlju 4.2.3.

Dodatni napredniji zadaci za vježbu

- Prilikom dodavanja novih točaka poligona, ispisujte vrijednost unutarnjeg kuta koji zatvaraju zadnje tri dodane točke.
- Obojite fragmente u unutrašnjosti poligona ovisno o udaljenosti do najbliže točke poligona.
- Obojite fragmente u unutrašnjosti poligona ovisno o udaljenosti do najbližeg brida poligona.
- Obojite sve fragmente rastera ovisno o udaljenosti do najbližeg brida

2.2.1 Dodatna pitanja

1. Isprobajte na primjeru kako će algoritam za popunjavanje poligona napraviti popunjavanje ako je zadani poligon konkavan. Objasnite rezultat. Znate li sada olovkom na papiru objasniti za proizvoljni konkavan poligon kako će izgledati rezultat popunjavanja ovim algoritmom?
2. Isprobajte na primjeru kako će algoritam koji ste trebali implementirati za ispitivanje odnosa točke i poligona klasificirati točke ako mu se zada konkavan poligon? Hoće li taj algoritam baš za sve točke raditi krivo? Objasnite.
3. Razmislite biste li problem utvrđivanja odnosa točke i konkavnog poligona mogli riješiti ispucavanjem polupravca iz zadane točke (u bilo kojem smjeru, a možda je najjednostavnije vodoravno u desno) te brojanjem sjecišta s bridovima poligona na koje se naiđe u tom smjeru? Skicirajte pseudokod takvog algoritma.

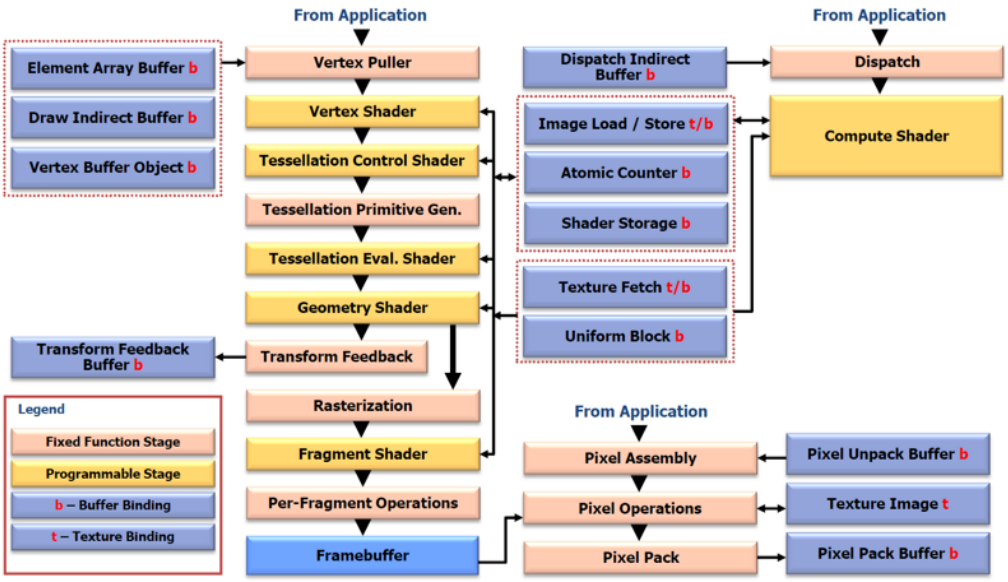
Laboratorijska vježba 3

Prvi program u OpenGL-u

U predlošku pod projektom *vjezba3* je implementiran OpenGL program kojeg možete koristiti kao početnu točku za razvoj idućih laboratorijskih vježbi. Program je napisan u proceduralnom stilu kako bi imali dobar pregled nad svim komponentama koje su potrebne za iscrtavanje scene na ekran. Preporučamo da slijedite objektnu paradigmu, a komentarima su odvojeni dijelovi programskog koda koji se mogu enkapsulirati u zasebne razrede.

Dostupan je pomoćni razred *Shader* koji iz datoteka učita programske kodove željenih sjenčara, prevede ih i poveže s aktivnim OpenGL kontekstom. Razred *Shader* sadrži identifikator *ID* koji se koristi prilikom iscrtavanja zajedno s naredbom glUseProgram(). Također, dostupan je i razred *FPSManager*, koji može poslužiti za dohvaćanje trajanja iscrtavanja, te prikaz i postavljanje broja sličica u sekundi(FPS).

U nastavku slijedi kratko objašnjenje modernog OpenGL protočnog sustava.



Slika 3.1: Protočni sustav OpenGL 4.6.

Na slici 3.1 je prikazan blok dijagram OpenGL protočnog sustava i slijed izvođenja različitih komponenti. Glavna ideja je da grafičkoj kartici pošaljemo podatke pomoću *buffer* objekata, opišemo kako su ti podaci organizirani, a da grafička kartica pokretanjem sjenčara (*shaders*) i ostalih komponenti protočnog sustava u *frambuffer* zapiše završnu sliku.

U laboratorijskim vježbama ćete primarno za ulazna polja podataka koristiti *Vertex Buffer Object* (VBO) i *Element Array Buffer* (EBO). S VBO i EBO ste se upoznali kroz predavanje "5. Modeliranje i reprezentacija objekata". Pomoću EBO možemo ostvariti indeksirani pristup organizaciji podataka.

Framebuffer je naziv za dvodimenzionalno polje podataka koje sadrži završnu sliku. Iz njega se šalju podaci ekranu na prikaz.

Od sjenčara, primarno ćemo se fokusirati na sjenčar vrhova (*Vertex shader*) i sjenčar fragmenata (*Fragment shader*). Za implementaciju laboratorijskih vježbi nije potrebno pisati sjenčare teselacije i geometrije koji uglavnom služe za umnažanje broja grafičkih primitiva na temelju početnih podataka.

Sjenčar vrhova je dio protočnog sustava koji primarno služi za transformaciju točaka modela. *Vertex Puller* njemu prosljeđuje podatke o jednom vrhu modela, a on na temelju uniformnih podataka o matrici modela, matrici pogleda i perspektivnoj matrici transformira vrh u osnovni OpenGL koordinatni sustav. Bojanje pojedinih fragmenata koje može biti na primjer ostvareno na temelju udaljenosti od kamere, od vrha poligona, svjetla i slično obavlja sjenčar fragmenata.

Grafička kartica je savršena za ove zadatke, jer može paralelno pozivati sjenčare za različite dijelove modela. Npr. sjenčar vrhova će se izvršiti onoliko puta koliko ima vrhova u modelu, a sjenčar fragmenata onoliko puta koliko se određeni fragment proba osvijetliti.

Nakon što ste proučili program *vježba3*, proučite i *primjerOpenGL*. U njemu je prikazano 5 načina iscrtavanja objekata. Svaki primjer se sastoji od dijela u kojemu se šalju podaci na grafičku karticu i dijela u kojem se poziva naredba za iscrtavanje. Svaki primjer se nadograđuje na prethodni. Proučite i razlike između pojedinih sjenčara.

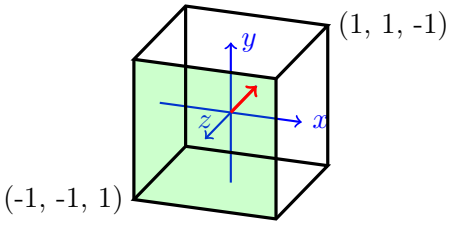
Prije nego nastavite, proučite i dokumentacije naredbi koje se koriste u predlošku. Morali bi moći odgovoriti na ova pitanja:

1. Čemu služi naredba `glfwSetFramebufferSizeCallback`? Pogledajte i slične naredbe poput `glfwSetCursorPosCallback`.
2. Čemu služe naredbe `glGenVertexArrays` i `glGenBuffers`, te `glBindVertexArray` i `glBindBuffer`?
3. Čemu služi naredba `glBufferData`?
4. Čemu služe naredbe `glVertexAttribPointer` i `glEnableVertexAttribArray`?
5. Čemu služi naredba `glVertexAttribDivisor`?
6. Čemu služi naredba `glUseProgram`?
7. Čemu služe naredbe `glGetUniformLocation` i npr. `glUniform3f`?
8. Čemu služi naredba `glViewport`?
9. Koja je razlika između `glDrawArrays`, `glDrawElements` i `glDrawElementsInstanced`?
10. Što deklariramo linijom `layout (location = 0) in vec3 aPos;` u sjenčaru vrhova?
11. Što deklariramo linijom `uniform mat4 tMatrica;` u sjenčaru vrhova?
12. Što deklariramo linijom `out vec4 FragColor;` u sjenčaru fragmenata?
13. Što je izlaz sjenčara fragmenata, a što sjenčara vrhova?

3.1 OpenGL osnovni volumen pogleda

U prošlim laboratorijskim vježbama ste se susreli s postupkom rasterizacije. Rasterizacija se izvršava nakon sjenčara vrhova, a prije sjenčara fragmenata. OpenGL prilikom rasterizacije prikazuje samo osnovni volumen pogleda. On se sastoji od prostora unutar intervala $[-1, 1]$ po svim osima, a sve ostalo se odsijeca. Zato, da bi prikazali željenu scenu, na neki način moramo modele transformirati tako da oni upadnu u taj prostor. To se radi pomoću matrica modela, pogleda i projekcije. Kamera se nalazi u ishodištu koordinatnog sustava i to tako da gleda u negativnom smjeru z osi. Osnovni volumen pogleda je prikazan na slici 3.2 gdje je crvenom strelicom označen smjer pogleda kamere, a svijetlo zelenom bojom površina na koju se projicira iscrtana scena.

U idućim laboratorijskim vježbama će biti više riječi o matricama modela, pogleda i projekcije, a za sada je samo bitno da razumijete kako je definiran OpenGL osnovni volumen pogleda.



Slika 3.2: Osnovni OpenGL volumen pogleda

3.2 Zadatak

Ovu laboratorijsku vježbu ćete rješavati unutar *vjezba3*. Izmijenite *vjezba3* tako da omogućite korisniku crtanje većeg broja ispunjenih trokuta u odabranoj boji. Za prenošenje podataka na grafičku karticu i iscrtavanje možete koristiti način iscrtavanja s indeksiranjem bez instanciranja, kao što je prikazano u primjeru 3 u predlošku pod *primjerOpenGL*. Program treba pamtiti trenutnu (aktivnu) boju, i tu boju treba korisniku prikazati u uglu prozora kao mali kvadratić koji je ispunjen tom bojom.

Zadavanje trokuta u programu potrebno je obavljati mišem. Program za svaki trokut koji je korisnik tako zadao treba pamtiti koordinatu vrha i boju koja je u trenutku zadavanja trokuta bila postavljena kao trenutna.

Dodavanje novog trokuta obavlja se na sljedeći način. Kada korisnik prvi puta klikne negdje na površinu prozora, zapamti se pozicija prvog vrha. Potom se prati pomicanje miša, i iscrtava linija u trenutnoj boji od pokazivača miša do zapamćene prve točke. U trenutku kada korisnik klikne drugi puta mišem, pamti se i drugi vrh trokuta. Sada se opet prate pomaci mišem i pri svakom pomaku crtaju bridovi privremenog trokuta određen s prethodna dva zapamćena vrha i trenutnom pozicijom pokazivača miša kao trećim vrhom. Kada korisnik klikne mišem treći puta, pamti se i treći vrh trokuta te se temeljem zapamćene tri lokacije i trenutne boje stvara novi trokut koji se dodaje u listu obojanih trokuta. Svakim idućim klikom miša, dodaje se novi trokut zadan s prethodne dvije i novom koordinatom koje je korisnik unio.

Trokute iscrtavajte pomoću `glDrawElements` naredbe i `GL_TRIANGLES` grafičke primitive, a trenutne linije pomoću naredbe `glDrawArrays` i `GL_LINE_STRIP` grafičke primitive. Koordinate vrhova trokuta se u polju podataka ne smiju ponavljati.

Omogućite korisniku da može mijenjati aktivnu boju po njenim komponentama. Npr. pritiskom na tipke `r`, `g` i `b` označava trenutnu komponentu, a pritiscima na `+` ili `-` pojačava ili smanjuje utjecaj pojedine komponente. Promjena trenutne boje treba odmah biti popraćena i vizualnom indikacijom.

3.3 Naputci

Organizirajte program tako da postoji zasebni objekt (odnosno objekti) koji čine stanje programa odnosno model podataka, te zaseban dio koji se bavi crtanjem – takva organizacija upravo je prirodna za OpenGL. Stanje Vašeg programa odnosno model podataka čine strukture podataka koje omogućavaju pamćenje trokuta, lista obojanih trokuta, trenutno odabrana boja, širina i visina samog prozora, je li korisnik već dodao prvu ili drugu točku za definiranje novog trokuta i slično.

Obrada svakog događaja treba biti napravljena tako da promijeni model podataka (evidentira novu trenutnu boju, doda trokut u polje trokuta i slično).

Kako je boja vezana uz svaki vrh i kako se ona šalje između sjenčara vrhova i sjenčara fragmenata, OpenGL će prilikom rasterizacije boju interpolirati po površini trokuta. Interpolacija vrijednosti se izvršava za sve vrijednosti koje šaljemo na taj način.

Podaci koji se nalaze na grafičkoj kartici koje smo prenijeli pomoću naredbe `glBufferData` se ne mijenjaju automatski, pa je potrebno prilikom izmjena podataka u glavnoj memoriji računala osvježiti

podatke i u memoriji grafičke kartice ponovnim pozivom naredbe `glBufferData`. Potrebno je najaviti koji *VBO* se mijenja pozivanjem naredbe `glBindBuffer`.

Potrebno je registrirati *callback* funkcije koje se pozivaju prilikom korisničkog unosa pomoću naredbi `glfwSetKeyCallback`, `glfwSetCursorPosCallback` i `glfwSetMouseButtonCallback`.

Dodatni napredniji zadaci za vježbu

- Proučite kako radi naredba `glBufferSubData`. Omogućite korisniku da pritiskom na tipku `h` invertira boju zadnja tri vrha. Osvježavanje polja podataka izvedite tom naredbom.
- Modificirajte polje indeksa tako da možete iscrtati trokute pozivom naredbe `glDrawElements` i `GL_TRIANGLE_STRIP` grafičke primitive.
- U sjenčar vrhova pomoću uniformne varijable pošaljite proteklo vrijeme od početka programa, te mijenjajte boje vrhova trokuta na temelju te varijable.
- U sjenčaru fragmenata pomoću varijable `gl_FragCoord` mijenjajte boje fragmenata. To je zadana ulazna varijabla koja nastaje kao rezultat interpolacije vrhova trokuta, tj. podataka zapisanih nakon sjenčara vrhova u varijabli `gl_Position`.

Laboratorijska vježba 4

3D tijela

Prvi korak u upoznavanju s 3D svijetom računalne grafike jest modeliranje objekata. Dok smo u 2D svijetu često zadovoljni s linijama, kružnicama i drugim krivuljama, u 3D svijetu naješće se bavimo modeliranjem i prikazivanjem različitih tijela. Postoji više načina kako 3D tijelo može biti zadano. Najjednostavnija tijela imaju jasan matematički opis. Tako primjerice, kuglu radijusa R čije je središte u prostoru smješteno u točku (x_c, y_c, z_c) matematički možemo opisati izrazom:

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = R^2.$$

Međutim, nema jednostavnog matematičkog izraza kojim bismo mogli opisati Zagrebačku katedralu ili pak Vašeg omiljenog lika iz igre *Doom* ili *Unreal tournament*. Stoga je u današnje doba velik naglasak stavljen na modeliranje tijela zadavanjem njegovog oplošja. Primjerice, oplošje najobičnije kocke možemo zamisliti da je sastavljeno od šest kvadrata koji su u prostoru tako posloženi da u potpunosti zatvaraju volumen kocke. Ovaj pristup primjenjiv je i na modeliranje složenijih tijela – sve što je potrebno jest uzeti veći dio dovoljno malih "pločica" koje se u prostoru poslože tako da aproksimiraju čitavo oplošje objekta. Kod jednostavnih tijela površine tih pločica mogu biti i velike – primjer je upravo kocka ili kvadar proizvoljnih dimenzija za koje je uvijek dovoljno samo šest pločica. Oplošje kugle na ovaj način nikada nećemo uspjeti savršeno opisati, ali uz dovoljan broj pločica rezultat može biti sasvim zadovoljavajući.

Spomenute pločice u praksi mogu biti poligoni, pri čemu je svaki poligon zadan s određenim brojem vrhova koji svi leže u istoj ravnini, ili pak mogu biti površine modelirane na različite načine (poput Bezierovih krpica koje spadaju u parametarske plohe). Međutim, rad s parametarskim plohama je računski vrlo zahtjevan – linearno opisane površine su bitno jednostavnije, pa su one i raširenije u uporabi. Kod linearnih ploha koje su zadane kao poligoni, situacija u kojoj se poligon definira s više od 3 točke mogu biti problematične – što ako četiri ili više zadanih vrhova poligona naprosto ne leže u ravnini? Kako bi se riješio ovaj problem, vrlo često se u praksi koriste najjednostavniji mogući poligoni – trokuti, i njih ćemo koristiti u ovoj vježbi.

Zadamo li tri točke koje međusobno nisu kolinearne, one sigurno leže u ravnini. Zadavanjem tri točke nismo međutim u potpunosti definirali sve parametre ravnine. Prije no što nastavite dalje, pažljivo prođite kroz poglavlje 2 u knjizi, a posebice kroz potpoglavlje 2.5. Vidjet ćete da ono što nam još nedostaje jest normala ravnine – tri točke ne mogu istovremeno odrediti i jednadžbu ravnine i normalu ravnine (odnosno smjer vektora normale ravnine); tri točke su nam dovoljne kako bismo odredili sve točke prostora koje leže u toj ravnini, i ništa više. Da bismo odredili normalu ravnine (a time i smjer u kojem ona gleda), trebamo još nekakav podatak. To bi trebalo biti jasno vidljivo i iz činjenice da je implicitni oblik jednadžbe ravnine u 3D prostoru jednak

$$ax + by + cz + d = 0 \tag{4.1}$$

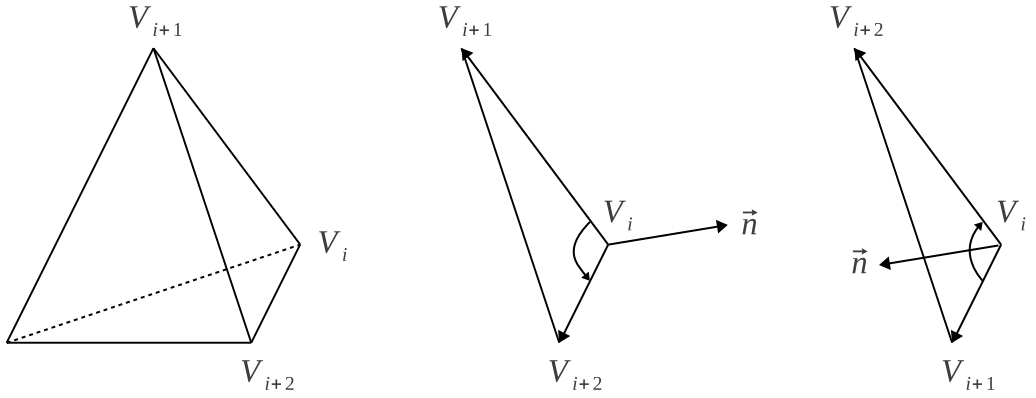
što je jednadžba s četiri nepoznanice. Ako znamo samo tri točke koje leže u toj ravnini (pa time zadovoljavaju prethodni izraz) nemamo dovoljno informacija kako bismo riješili sustav i odredili sve četiri nepoznanice: a, b, c i d.

Ovdje u pomoć priskaču konvencije kojih se želimo držati prilikom opisivanja oplošja tijela trokutima. Svaki trokut koji čini oplošje tijela (odnosno ravnina kojoj on pripada) dijeli prostor u dva podprostora. U jednom od ta dva podprostora smješten je i volumen tijela čije oplošje opisujemo dok u drugom poluprostoru nema ničega. Ovo dakako vrijedi ako je tijelo koje opisujemo konveksno; ako nije, onda izjava sigurno vrijedi barem za dio podprostora koji je s jedne i druge strane samog trokuta – s jedne strane je volumen tijela a s druge strane praznina; kada to ne bi bio slučaj, onda promatrani trokut ne bi mogao biti dio oplošja. Konvencija koje se ovdje želimo držati je sljedeća: želimo da normale ravnina za sve trokute koji čine oplošje tijela konzistentno gledaju ili u unutrašnjost tijela ili prema vanjštini tijela. Potpuno je nebitno odabere li se jedno ili drugo, važno je samo da odabir vrijedi za sve trokute i da ga unaprijed znamo. Normale trokuta za tijela koja ćemo koristiti u ovoj vježbi gledat će prema vanjštini tijela.

No kako ćemo određivati normale? Pretpostavimo da je trokut zadan s tri točke: V_i, V_{i+1} i V_{i+2} (upravo tim redoslijedom). Fiksiramo li točku V_i, možemo izračunati dva vektora koja razapinju ovu ravninu: $V_{i+1} - V_i$ i $V_{i+2} - V_i$. Prema dogovoru ćemo normalu računati kao vektorski produkt ovih dvaju vektora (i to upravo redoslijedom kojim smo ih napisali). Dakle, normalu ravnine razapete s tri točke V_i, V_{i+1} i V_{i+2} (zadane tim redoslijedom) računat ćemo kao vektorski produkt:

$$\vec{n} = (V_{i+1} - V_i) \times (V_{i+2} - V_i). \tag{4.2}$$

Zamjena redoslijeda točaka V_i, V_{i+1} i V_{i+2} rezultira dobivanjem kolinearnog vektora suprotnog smjera, pa na to svakako treba pripaziti. Prema pravilu desne ruke vektorski produkt jasno određuje smjer u kojem će vektor normale biti orijentiran. I sada imamo sve potrebno da izvedemo zaključak o tome kako vrhovi moraju biti zadani. Prethodno smo rekli kako želimo da normala trokuta gleda u vanjštinu tijela; potom smo rekli da trokut zadajemo s tri vrha V_i, V_{i+1} i V_{i+2} (tim redoslijedom) i konačno da normalu računamo vektorskim produktom prema izrazu (4.2). Ako je to istina i ako trokut promatramo iz onog poluprostora u koji pokazuje normala, vrhovi V_i, V_{i+1} i V_{i+2} moraju biti tako zadani da generiraju obilazak koji je u smjeru suprotnom od smjera kazaljke na satu. Ovo je ilustrirano na slici 4.1. Ako je tijelo koje opisujemo konveksno (poput kocke), pravilo je još jednostavnije: gledano izvan tijela prema tijelu trokut mora biti zadan tako da obilaskom njegovih vrhova radimo gibanje koje je u smjeru suprotnom od smjera kazaljke na satu. Primjerice, tijelo prikazano na slici 4.1 je konveksno pa je lako uočiti direktno sa slike da ovaj zaključak također vrijedi.



Slika 4.1: Važnost redoslijeda kojim su zadani vrhovi tijela. Lijevo: primjer nepravilnog tetraedra s tri istaknuta vrha. Sredina: izdvojena stranica. Uz prikazani redoslijed normala gleda prema vanjštini tijela. Desno: izdvojena stranica ali uz drugačiji redoslijed vrhova. Uz prikazani redoslijed normala gleda prema unutrašnjosti tijela.

Ponovimo još jednom: zaključak koji je iznesen vrijedi samo ako vrijede i sve prethodno iznesene ograde: želimo da sve normale budu usmjerene konzistentno, želimo da taj smjer bude prema vanjštini tijela, želimo raditi s trokutima koje zadajemo preko tri vrha i konačno, normalu računamo upravo prema izrazu (4.2). Ako bismo odstupili od ovih zahtjeva i promijenili jedan od njih, primjerice, da želimo da sve normale gledaju u unutrašnjost tijela, i naš bi se zaključak promijenio – vrhovi bi morali biti zadani tako da, ako ih promatramo iz vanjštine tijela, sada činimo obilazak koji odgovara smjeru

kazaljke na satu. Međutim, ako promijenimo priču još malo pa kažemo da trokut promatramo iz unutrašnjosti tijela (u koju sada pokazuju i normale), zaključak bi opet bio da vrhovi moraju biti zadani tako da njihovim obilaskom činimo gibanje koje je u smjeru suprotnom od smjera kazaljke na satu. Za konzistentnost ovoga obično se brinu programi koji korisniku omogućavaju 3D modeliranje objekata.

Jednom kad smo odredili normalu, još nam ostaje do kraja odrediti sve koeficijente jednadžbe ravnine. U jednadžbi ravnine, koeficijenti a, b i c upravo odgovaraju komponentama normale. Ako je $\vec{n} = (n_x, n_y, n_z)$, tada vrijedi $a = n_x$, $b = n_y$, $c = n_z$ (pogledajte u knjizi izvod izraza (2.34)). Koeficijent d tada možemo odrediti uporabom bilo kojeg od vrhova. Naime, s obzirom da smo izračunali a, b i c (jer smo izračunali normalu), jedina nepoznanica jest d. Slijedi:

$$d = -ax - by - cz. \tag{4.3}$$

S obzirom da znamo čak tri točke koje zadovoljavaju zadanu jednadžbu ravnine (a to su V_i, V_{i+1} i V_{i+2}), d možemo odrediti preko bilo koje od njih; primjerice, vrijedi:

$$d = -aV_{i,x} - bV_{i,y} - cV_{i,z}. \tag{4.4}$$

Uz pretpostavku da je tijelo čije smo oplošje opisali trokutima uz pridržavanje prethodno definirane konvencije konveksno, ispitivanje odnosa točke i tijela provodi se na identičan način kako smo to radili u vježbi s poligonima. Vrijedi sljedeće: ako je tijelo zadano tako da normale svih trokuta kada ih računamo prema izrazu (4.2) gledaju prema vanjštini tijela, to znači da će za proizvoljnu točku $T = (x, y, z)$ vrijednost izraza $ax + by + cz + d$ biti pozitivna ako točka T ne leži u ravnini već je negdje u poluprostoru u koji pokazuje normala, bit će jednaka 0 ako točka T leži u ravnini (jer tada zadovoljava jednadžbu ravnine), i bit će manja od nule ako točka T leži u poluprostoru u koji normala ne pokazuje (odnosno gdje bi pokazivala $-\vec{n}$). No tada je proizvoljna točka T unutar tijela ako je za svaki trokut i pripadnu ravninu vrijednost izraza $ax+by+cz+d$ negativna (drugim riječima, ako točka za svaki trokut oplošja leži ispod ravnine u kojoj se nalazi taj trokut). Ako je točka T za sve trokute oplošja ispod ravnina u kojima se nalaze odnosni trokuti osim za neke za koje je vrijednost izraza jednaka nuli, točka je na oplošju tijela. Konačno, ako postoji trokut za koji je točka iznad njegove ravnine (odnosno za koji je $ax + by + cz + d$ pozitivno), točka je izvan tijela.

Razmislite kako biste riješili ispitivanje za slučaj konkavnog tijela. Vrijede li tada prethodno opisani zaključci? Biste li zadatak mogli riješiti ispucavanjem polupravca i utvrđivanjem sjecišta s trokutima?

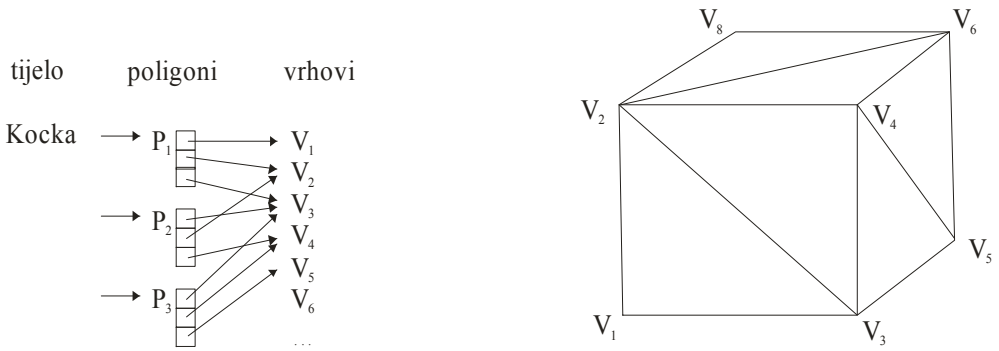
4.1 Zapis oplošja modela

U okviru ove vježbe radit ćemo s modelima tijela koja su definirana zadavanjem njihovog oplošja i to preko niza trokuta. Opis tjela pri tome ćemo čitati iz datoteke. Zapisivanje trodimenzijskih objekata obično je propisano specifikacijama proizvođača programske opreme koji ujedno daju i alate za modeliranje i prikazivanje 3D scena te učitavanje i pohranu tijela u datoteku. Primjeri različitih formata za pohranu 3D tijela su datoteke s ekstenzijom `.3ds`, `.max`, `.dxf`, `.ply`, `.obj`, `.xgl`, `.stl`, `.wrl`, `.iges` i slični. Ovisno o zapisu, zapisani mogu biti trokuti ili poligoni s više vrhova, grupe poligona, boje, teksture, normale u poligonima ili vrhovima, krivulje, površine, način konstrukcije objekata, podaci o sceni, izvori svjetla, podaci o animaciji i slično. U ovoj vježbi bit će korišten zapis `.obj`.

Zapis `.obj` podržava zapis modela pomoću poligona s proizvoljnim brojem vrhova, ali glava robota i modeli na web stranicama su pripremljeni tako da zapis koristi samo trokute. Kako više trokuta može dijeliti isti vrh, u datoteci je najprije dan popis vrhova, a potom slijede definicije trokuta koji referenciraju indeksom te vrhove. Opis tijela preko trokuta ilustriran je na slici 4.2.

Konkretan primjer `.obj` datoteke u kojoj je zapisan model kocke prikazan je u nastavku.

```
v 0.0 0.0 0.0
```



Slika 4.2: Opis tijela trokutima

```
v 0.0 0.0 1.0
v 1.0 0.0 0.0
v 1.0 0.0 1.0
v 1.0 1.0 0.0
v 1.0 1.0 1.0
v 0.0 1.0 0.0
v 0.0 1.0 1.0
f 1 3 2
f 3 4 2
f 3 5 4
f 5 6 4
f 5 7 6
f 7 8 6
f 7 1 8
f 1 2 8
f 1 5 3
f 1 7 5
f 2 4 6
f 2 6 8
```

Primjer definira 8 vrhova te 12 trokuta. Vrhovi se nalaze u redcima koji počinju s `v` (od engl. *vertex*); za svaki je vrh navedena x, y i z koordinata. Trokuti su zapisani u redcima koji počinju s `f` (od engl. *face*); svaki trokut navodi indekse pripadnih vrhova pri čemu su indeksi numerirani od 1. Skicirajte ovu kocku i uvjerite se da obilazak vrhova, ako trokut promatrate iz točke koja je izvan kocke, generira gibanje u smjeru suprotnom od smjera kazaljke na satu.

Model glave robota sadrži i redke koji počinju slovima `vn` i `vt`. Ti redci zapisuju uv koordinate i normale u vrhovima objekta, a poligoni imaju dodatne reference kao što je prikazano u nastavku:

```
v -0.294176 -0.303604 0.108356
vt 0.612800 0.004200
vn 0.2651 -0.9642 -0.0000
f 753/753/617 791/803/655 2030/2142/1705 1998/2098/1673
```

Redci koji započinju s `vt` su uv koordinate na teksturi, `vn` su normale, a redci koji započinju s `f` indeksiraju redom `v/vt/vn`. U ovom slučaju se radi o poligonu s četiri vrha. Ti dodatni podaci o uv koordinatama, normalama, pripadna .mtl datoteka i teksture će biti potrebni u kasnijim laboratorijskim vježbama, pa ih za sada možete zanemariti.

4.2 Pitanja

1. Na koji se način može opisati 3D-tijelo?
2. Kako 3D-tijela opisujemo u ovoj vježbi?
3. Na koji se način temeljem triju točaka dolazi do jednadžbe ravnine? Je li ta jednadžba do kraja definirana?
4. Čime je određen smjer normale trokuta?
5. Koje zahtjeve postavljamo na zapis modela 3D-tijela u ovoj vježbi, odnosno kojih se konvencija pridržavamo?
6. Na koji se način ispituje odnos točke i 3D-tijela ako je tijelo konveksno?
7. Bi li način ispitivanja odnosa točke i 3D-tijela koji je opisan u ovoj vježbi radio za konkavna tijela?
8. Biste li zadatak ispitivanja odnosa točke i konkavnog 3D-tijela mogli riješiti ispucavanjem polupravca i utvrđivanjem sjecišta s trokutima? Objasnite.
9. Kakva je struktura `.obj` datoteke?

4.3 Zadatak

Unutar *vjezba4* napravite program koji će učitati i iscrtati model tijela. Za učitavanje `.obj` datoteka koristite biblioteku *assimp*. U predlošku pod nazivom *primjerASSIMP* se nalazi primjer učitavanja modela glave robota. Primjer ispisuje različite podatke koji su zapisani u toj datoteci.

Program treba omogućiti normiranje modela objekata. Drugim riječima, da skaliranjem i translacijom koordinata, objekt postavite unutar kocke dimenzija $2 \times 2 \times 2$ i središtem u centru koordinatnog sustava. Prilikom normiranja modela potrebno je proći kroz sve vrhove modela i utvrditi minimalne i maksimalne iznose svih koordinata: x_{min}, x_{max}, y_{min}, y_{max} te z_{min}, z_{max}. Potom se računa središte objekta kao:

$$\bar{x} = \frac{x_{min} + x_{max}}{2}$$
$$\bar{y} = \frac{y_{min} + y_{max}}{2}$$
$$\bar{z} = \frac{z_{min} + z_{max}}{2}$$

te maksimalni raspon po bilo kojoj od osi:

$$M = \max(x_{max} - x_{min}, y_{max} - y_{min}, z_{max} - z_{min}).$$

Sve je vrhove potrebno translatirati za $(-\bar{x}, -\bar{y}, -\bar{z})$ čime se vrh $V_i = (V_{i,x}, V_{i,y}, V_{i,z})$ preslikava u $(V_{i,x} - \bar{x}, V_{i,y} - \bar{y}, V_{i,z} - \bar{z})$. Konačno, koordinate svih vrhova potrebno je skalirati s $\frac{2}{M}$. Ovim postupkom osigurava se da je raspon koordinata po svim osima za model u rasponu ne većem od $[-1, 1]$ (a za onu os za koju je tijelo izvorno imalo najveći raspon bit će upravo $[-1, 1]$).

Za operacije skaliranja i translacije izgradite pripadne matrice transformacije kojima ćete množiti koordinate objekta. Poslužite se bibliotekom *glm*.

Idući naputci bi vam trebali olakšati izradu idućih laboratorijskih vježbi, ali ih se ne morate striktno držati.

Napišite razred `Transform` koji će enkapsulirati poziciju, orijentaciju i veličinu objekta koji se može postaviti u scenu. Kasnije će kamera i svjetlo također koristiti taj razred. On će sadržavati i različite medote kojima možete lako upravljati navedenim parametrima poput `setPosition`, `setOrientation`, `move`, `rotate`, `scale` i slično.

Napišite razred `Mesh` koji će enkapsulirati podatke o mreži poligona jednog objekta, poput polja vrhova i indeksa. On bi trebao omogućiti izradu *VAO* objekata te prenošenje i osvježavanje podataka na grafičkoj kartici. Napišite metodu `getBoundingBox` koja vraća minimalne i maksimalne koordinate objekta. Napišite i metodu `applyTransform` koja na temelju matrice transformacije osvježi koordinate objekta i pripadni *VAO*. Ta metoda će vam poslužiti za normiranje modela.

Napišite razred `Object` koji će enkapsulirati podatke vezane uz jednu instancu modela objekta. On je zadužen za povezivanje informacije o mreži poligona koju koristi, transformaciji objekta i sjenčaru kojim se iscrtava.

Napišite razred `Renderer` koji će sadržavati instance objekata koje se nalaze u sceni. Njegov zadatak će biti prikupiti sve objekte koji se iscrtavaju na isti način i iscrtati ih.

Napomena

U predlošku pod *primjerASSIMP* unutar direktorija *resources* nalazi se model glave robota. Ostale `.obj` datoteke (uključivo i datoteku `kocka.obj`) dostupne su na web stranicama kolegija[1].

[1] `http://www.zemris.fer.hr/predmeti/irg/labosi/lab_objekti.html`

Laboratorijska vježba 5

Matrice modela, pogleda i projekcije

U prethodnoj vježbi upoznali smo se modeliranjem 3D tijela preko opisivanja njegovog oplošja. Također, pogledali smo jedan primjer formata datoteke koji služi za pohranu takvog opisa – format `.obj` datoteke. U ovoj vježbi, Vaš krajnji zadatak je ostvariti mogućnost instanciranja više modela te omogućiti korisniku da upravlja kamerom pomoću miša i tipkovnice.

Kao što je bilo ukratko objašnjeno u potpoglavlju 3.1, OpenGL koristi implicitno definirani volumen pogleda koji je kocka koja se po x-koordinati, y-koordinati i z-koordinati proteže od -1 do $+1$. Od ove kocke važno je izdvojiti njezine dvije plohe, obje paralelne s xy ravninom:

- ploha paralelna s xy ravninom koja leži na $z = -1$ – to je bliža ploha, i vrijednost $z = -1$ još se označava i sa z_{near};
- ploha paralelna s xy ravninom koja leži na $z = +1$ – to je dalja ploha, i vrijednost $z = +1$ još se označava i sa z_{far}.

Na bližoj plohi razapet je dvodimenzijski koordinatni sustav čije su koordinatne osi paralelne s izvornim x- i y- osima, samo što su pomaknute na $z = z_{near}$. Raspon tog 2D koordinatnog sustava je po obje osi i dalje od -1 do $+1$.

Pogledajmo sada što se događa kada OpenGL-u kažemo da nacrta neki slikovni element. Ako je vrijednost x koordinate, y koordinate ili z koordinate izvan volumena pogleda koji je određen granicama $-1 \leq x \leq 1$, $-1 \leq y \leq 1$ i $-1 \leq z \leq 1$, slikovni element će biti ignoriran – ništa se neće nacrtati. Ako slikovni element leži unutar volumena pogleda, daljnja obrada ovisi o tome je li uključena uporaba z-spremnika ili nije. Ako nije uključena, točka će se preslikati u točku s koordinatama (x, y, z_{near}) – dakle, konceptualno, preslikat će se na bližu plohu i tu će stvarati sliku.

Ako je uporaba z-spremnika omogućena, tada će se pogledati kolika je udaljenost te točke od bliže plohe i je li već prethodno na koordinate (x, y) nacrtana neka točka. Ako prethodno ništa nije nacrtano na koordinatama (x, y), na bližoj plohi nacrtat će se zadani slikovni element (odnosno točka (x, y, z_{near})) i u z-spremniku će se zapamtiti njezina izvorna udaljenost od bliže plohe. Ako je već nešto nacrtano, tada će se u z-spremniku pogledati na kojoj se je udaljenosti od bliže plohe nalazila ta točka. Novi slikovni element precrtat će se preko starog samo ako je za stari slikovni element u z-spremniku zapisana veća udaljenost no što je udaljenost novog slikovnog elementa, i tada će se udaljenost u z-spremniku korigirati tako da postane jednaka udaljenosti novog slikovnog elementa. Konceptualno, možete zamisliti da je promatrač koji gleda scenu smješten na $-z$-osi (negdje daleko) i gleda prema $+z$-osi; sada je jasno da će vidjeti, kao konačni slikovni element, onaj koji je najbliži bližoj plohi.

Jednom kada je slika stvorena na bližoj plohi, ona se prenosi na dio zaslona na kojem će biti prikazana. Za ovaj prijenos zaduženo je definiranje otvora (*viewport*) naredbom `glViewport` koja kao parametre prima koordinate donjeg lijevog kuta otvora, širinu i visinu željenog otvora. Ishodište koordinatnog sustava za naredbu `glViewport` je u donjem lijevom uglu prozora. Primjerice, ako imamo prozor dimenzija 640×480 i podesimo da je otvor upravo dio prozora koji se po x-u proteže od 320 do

640 a po y-u od 240 do 480, slika dobivena na bližoj plohi bit će će prikazana u desnoj gornjoj četvrtini prozora.

Da biste se pripremili za ovu vježbu, pažljivo proučite poglavlja 5 i 6 u knjizi. Obratite pažnju na dvije konvencije djelovanja operatora: množenje točke matricom operatora te množenje matrice operatora točkom. U prvom slučaju točku ćemo prikazati kao jednoretčanu matricu; u drugom slučaju točku ćemo prikazati kao jednostupčanu matricu. Za matrice uz navedene konvencije vrijedi odnos prikazan izrazom 5.3 u knjizi. U ovoj vježbi koristit ćemo drugu konvenciju (množenje matrice operatora točkom). Proučite kako se definiraju matrice operatora translacije, skaliranja i rotacije u 3D sustavu. Također proučite kako se rade ortografska i perspektivna projekcija i kako izgledaju pripadne matrice.

5.1 Pitanja

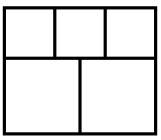
1. Što je to *volumen pogleda*?
2. Kako je određen inicijalni volumen pogleda kod *OpenGL*-a?
3. U kakvoj su vezi volumen pogleda i pojam odsijecanja? Što OpenGL odsijeca? Dajte primjer.
4. Napišite matricu operatora translacije za (d_x, d_y, d_z) uz obje konvencije. Kako glase matrice koje točku vraćaju u početnu?
5. Napišite matricu operatora skaliranja s faktorima s_x, s_y i s_z uz obje konvencije. Kako glase matrice koje točku vraćaju u početnu?
6. Je li transformacija pogleda jednoznačno definirana zadavanjem točke očišta i gledišta? Objasnite.
7. Što je to *view-up* vektor i čemu služi? Leži li on u ravnini projekcije?
8. Biblioteka *glm* nudi naredbe glm::rotate, glm::scale, glm::translate. Čemu služe te naredbe, koji su njihovi argumenti i kako izgledaju pripadne matrice?
9. Biblioteka *glm* nudi naredbu glm::lookAt. Čemu služi ta naredba, kakvu transformaciju pogleda radi, koji su njezini argumenti i kako izgleda pripadna matrica?
10. Biblioteka *glm* nudi naredbu glm::ortho. Čemu služi ta naredba, kako izgleda volumen pogleda koji definira, koji su njezini argumenti i kako izgleda pripadna matrica?
11. Biblioteka *glm* nudi naredbu glm::frustum. Koji je prijevod riječi *frustum*? Čemu služi ta naredba, kako izgleda volumen pogleda koji definira, koji su njezini argumenti i kako izgleda pripadna matrica?
12. OpenGL nudi naredbu glViewport. Čemu služi ta naredba i koji su njezini argumenti?

5.2 Zadatak

Priprema

Ako pogledate *primjerOpenGL* možete primijetiti da se za iscrtavanje trokuta na različite dijelove ekrana koristila naredba `glViewport`. Isto tako, možete primijetiti da postoji samo 5 primjera iscrtavanja koji su raspoređeni u dva reda i tri stupca, pa ostaje jedno "prazno" mjesto. Za vježbu prilagodite program tako da primjeri iscrtavanja popločaju prozor na način kako je prikazano na slici 5.1. Omjer visine i širine svakog otvora treba biti jednak.

U okviru ove vježbe napravit ćete dva programa, pri čemu ćete koristiti dijelove koda koje ste već prethodno napisali. Oba programa omogućavaju izradu žičanog prikaza 3D tijela. Žičani model možete



Slika 5.1: Zadatak popločavanja u *primjerOpenGL*

iscrtati tako da umjesto *GL_ TRIANGLES* koristite *GL_ LINE_ LOOP* grafičku primitivu. Struktura programa je u oba slučaja ista:

1. Program učitava tijelo i radi normalizaciju koordinata objekta (ovo ste implementirali u okviru prethodne vježbe, sada to samo iskoristite).
2. Program stvara prozor u kojem ćete uporabom OpenGL-a napraviti crtanje tijela.
3. Trebate pomaknuti i zarotirati **dvije** instance istog objekta te izračunati pripadne matrice modela.
4. Trebate mijenjati poziciju i rotaciju kamere pomoću korisničkih naredbi te izračunati pripadnu matricu pogleda.
5. Odrediti parametre za izradu perspektivne projekcije te na temelju njih izraditi perspektivnu matricu.
6. Prilikom iscrtavanja, sjenčarima proslijediti svaku matricu zasebno preko `uniform` varijabli.

5.2.1 Zadatak 1

U ovoj inačici programa pod *vjezba5a* sav posao matričnog izračuna prepustit ćete biblioteci *glm*. Za izradu matrica rotacije, pomaka i skaliranja možete koristiti *glm* funkcije `rotate`, `translate` i `scale`. Funkcija `inverse` može poslužiti za izradu inverzne matrice. Iskoristite funkciju `lookAt` za izradu matrice pogleda, te `frustum` za izradu perspektivne projekcije.

Unutar razreda `Renderer` ostvarite agregiranje svih objekata koji se nalaze u sceni. Objekte koji su istog tipa (imaju istu poligonalnu mrežu i iscrtavaju se istim sjenčarima), iscrtajte kao što je to napravljenu u *primjeru 4a* unutar *primjerOpenGL*.

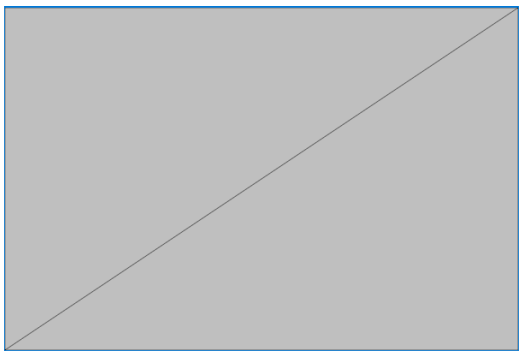
Dodajte razredu `Transform` metodu `getViewMatrix` koja vraća matricu pogleda i metodu `getModelMatrix` koja vraća model matricu.

Napravite razred `Camera` koja će enkapsulirati podatke potrebne za izradu perspektivne projekcije, a nasljeđuje i razred `Transform` kako bi ju mogli pomicati po sceni i izgraditi matricu pogleda. Razredu `Camera` dodajte i metodu `getPerspectiveMatrix` koja na temelju širine i visine prozora izradi perspektivnu matricu.

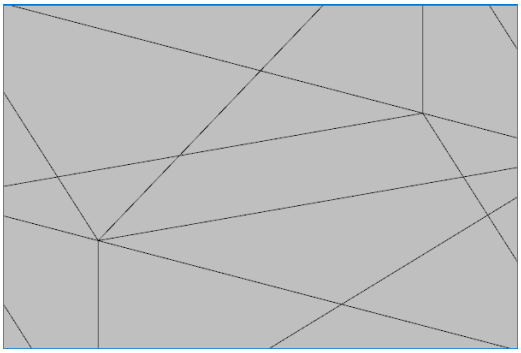
Postepeno u sjenčare uvodite matrice modela, pogleda i projekcije. Držite se poretka množenja matrice s točkom koristeći homogene koordinate. Svaka instanca modela koja se iscrtava posjeduje svoju matricu modela, dok se matrica pogleda i perspektive primjenjuju nad svim objektima jednako. Matrice pogleda i perspektive se izrađuju samo iz informacija vezanih uz kameru. Pogledajte potpoglavlje 5.2.3 za dodatne naputke.

Da podsjetimo, OpenGL osnovnu kameru ne možete micati, nego se privid pomicanja kamere ostvaruje primjenom inverznih operacija nad cijelom scenom preko matrice pogleda.

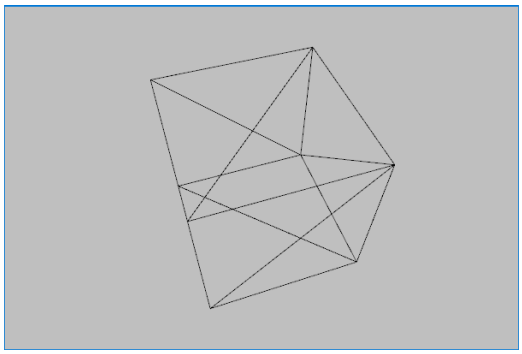
Na slici 5.2c prikazan je rezultat koji biste morali dobiti prikazivanjem normalizirane kocke ako očište smjestite u točku $(3, 4, 1)$, gledište u točku $(0, 0, 0)$, uzmete *view-up* vektor $(0, 1, 0)$, te podesite volumen pogleda tako da se po x i y osima protože ± 0.5 te da je bliža ploha na 1 a dalja na 100.



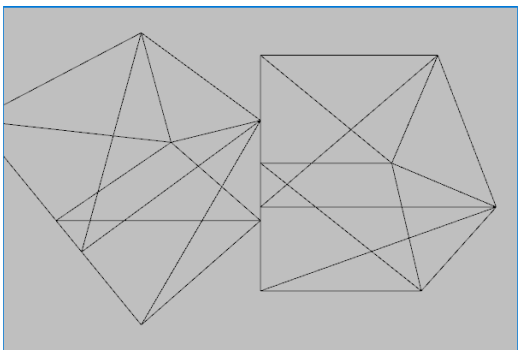
(a) Žičani prikaz normalizirane kocke bez dodatnih matrica.



(b) Prikaz kocke s matricom pogleda uz postavljanje z koordinate u 0.



(c) Kocka prikazana s matricom pogleda i projekcije.



(d) Instance dvije kocke ostvarene pomoću matrica modela.

Slika 5.2: Postepeno dodavanje matrica transformacija

5.2.2 Zadatak 2

U ovoj inačici programa pod *vjezba5b* Vi ćete obaviti kompletan posao izrade matrica modela, pogleda i projekcije. Možete koristiti biblioteku *glm* za osnovne matematičke operacije između vektora i matrica. Možete ju koristiti i za spremanje samih podataka o vektoru i matrici.

Prvi korak

Razredu `Transform` dodajte statičke metode:

- glm::mat4 translate3D(glm::vec3 translateVector);
  Metoda vraća matricu koja odgovara operatoru translacije uz konvenciju množenja matrice s točkom.
- glm::mat4 scale3D(glm::vec3 scaleVector);
  Metoda vraća matricu koja odgovara operatoru skaliranja uz konvenciju množenja matrice s točkom.
- glm::mat4 lookAtMatrix(glm::vec3 eye, glm::vec3 center, glm::vec3 viewUp);
  Metoda vraća matricu koja odgovara transformaciji pogleda koja je zadana očištem, vektorom pogleda te *view-up* vektorom, uz konvenciju množenja matrice s točkom. Prilikom izrade matrice pogleda, vodite se načinom koji je opisan u poglavlju 6.4 knjige. Ravnina u kojoj se stvara slika sadrži očište i njezina normala je kolinearna s vektorom očište-gledište. Ishodište koordinatnog sustava smješteno je u točku očišta, negativna z-os se proteže iz očišta prema zadanom centru. y-os određena je projekcijom *view-up* vektora na ravninu u kojoj se stvara slika. Konačno, sustav je desni čime je do kraja definiran.

Postepeno uvodite svoje funkcije umjesto onih iz biblioteke *glm* koje ste koristili za izgradnju

matrica transformacija. Izračunajte matrice transformacije modela i pogleda mp.

```
glm::mat4 mp = Transform::lookAtMatrix(glm::vec3(3,4,1), glm::vec3(0,0,0), glm::vec3(0,1,0));
```

Uz pretpostavku da ste uzeli koordinate očišta, centra pogleda i *view-up* vektor kako je prethodno zadano, ova bi matrica vrhove normalizirane kocke trebala preslikati u sljedeće točke.

$$
\begin{aligned}
(-1.0, -1.0, -1.0) &\rightarrow (0.632, 0.372, -6.668)\\
(-1.0, -1.0, 1.0) &\rightarrow (-1.265, -0.124, -6.276)\\
(1.0, -1.0, -1.0) &\rightarrow (1.265, -1.116, -5.491)\\
(1.0, -1.0, 1.0) &\rightarrow (-0.632, -1.612, -5.099)\\
(1.0, 1.0, -1.0) &\rightarrow (1.265, 0.124, -3.922)\\
(1.0, 1.0, 1.0) &\rightarrow (-0.632, -0.372, -3.530)\\
(-1.0, 1.0, -1.0) &\rightarrow (0.632, 1.612, -5.099)\\
(-1.0, 1.0, 1.0) &\rightarrow (-1.265, 1.116, -4.707)
\end{aligned}
$$

Rezultat prikaza slike na zaslonu trebao bi izgledati kao što je prikazano na slici 5.2b. Uočite da ovdje još nismo radili perspektivnu projekciju, već smo naprosto za svaku točku odbacili njezinu z-koordinatu (čime smo zapravo napravili ortografsku projekciju). Odbacivanje z koordinate se može izvesti u sjenčaru vrhova, tako što svakoj točki nakon množenja s matricom pogleda z koordinatu postavimo u 0.

Drugi korak

Dodajte sada u razred Transform još jednu statičku metodu koja je opisana u nastavku.

- glm::mat4 frustum(**double** l, **double** r, **double** b, **double** t, **double** n, **double** f);
  Izvod za elemente pripadne matrice možete pogledati u knjizi pod poglavljem 6.5.6

Sada možete modificirati stvaranje matrice mp kako je opisano u nastavku.

```
glm::vec4 mp = Transform::frustum(-0.5, 0.5, -0.5, 0.5, 1, 100);
```

Uz ovako definirane matrice pogleda i perspektivne matrice dobit ćete prikaz koji odgovara onom sa slike 5.2c.

Uz pretpostavku da ste uzeli koordinate očišta, centra i *view-up* vektor kako je prethodno zadano, ova bi matrica (koja je sada umnožak matrice transformacije pogleda i perspektivne projekcije) vrhove normalizirane kocke trebala preslikati u sljedeće točke.

$$
\begin{aligned}
(-1.0, -1.0, -1.0) &\rightarrow (0.190, 0.112, 0.700)\\
(-1.0, -1.0, 1.0) &\rightarrow (-0.403, -0.040, 0.681)\\
(1.0, -1.0, -1.0) &\rightarrow (0.461, -0.407, 0.636)\\
(1.0, -1.0, 1.0) &\rightarrow (-0.248, -0.632, 0.608)\\
(1.0, 1.0, -1.0) &\rightarrow (0.645, 0.063, 0.490)\\
(1.0, 1.0, 1.0) &\rightarrow (-0.358, -0.211, 0.433)\\
(-1.0, 1.0, -1.0) &\rightarrow (0.248, 0.632, 0.608)\\
(-1.0, 1.0, 1.0) &\rightarrow (-0.537, 0.474, 0.575)
\end{aligned}
$$

5.2.3 Korisnički unos i pomicanje kamere

Modificirajte oba programa *vjezba5a* i *vjezba5b* tako da im dodate mogućnost osluškivanja pomaka miša i pritisaka na tipke tipkovnice. Cilj je omogućiti korisniku translatirati i rotirati objekte u sceni koji nasljeđuju razred `Transform`. Program treba podržavati sljedeće operacije koje se mogu primijeniti nad instancom modela razreda koji je naslijedio razred `Transform`:

Pomak miša	**Tražena funkcionalnost**
`gore-dolje`	Rotirati smjer pogleda oko lokalne x osi
`lijevo-desno`	Rotirati smjer pogleda oko globalne y osi

Tipka	**Tražena funkcionalnost**
`w`	Pomaknuti objekt u smjeru pogleda, tj. negativne lokalne z-osi
`s`	Pomaknuti objekt u smjeru lokalne z-osi
`a`	Pomaknuti objekt u smjeru negativne lokalne x-osi
`d`	Pomaknuti objekt u smjeru lokalne x-osi
`e`	Pomaknuti objekt u smjeru negativne lokalne y-osi
`q`	Pomaknuti objekt u smjeru lokalne y-osi

U prozoru vašeg programa možete sakriti pokazivač miša pomoću naredbe `glfwSetInputMode`, a pomoću naredbe `glfwSetCursorPos` ga možete zamrznuti. Slobodno koristite biblioteku *glm*.

U razred `Transform` dodajte metode koje će podržati izvršavanje prethodno navedenih operacija.

Naputci

Pri izradi ove laboratorijske vježbe najviše vremena se obično potroši na greške s krivo orijentiranim matricama, pa da još jednom ponovimo. U sjenčarima se množi $V' = M_p * M_v * M_m * V$. Ako bi u matrici modela M_m imali zapisanu samo translaciju, očekivani poredak elemenata bi bio `{1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, X, Y, Z, 1}`. Ako bibliotekom *glm* izgradite matricu translacije, ona će izraditi matricu za taj isti poredak množenja. Na papiru se može činiti da su matrice krivo orijentirane, ali samo zato što sjenčari podatke o matricama "drže" po stupcima, a ne po recima. Dohvat pojedinačnih elemenata matrice je ostvaren prvo preko stupaca, a potom preko redaka, pa će `matrica[3][0];` zato dohvatiti vrijednost pomaka po X osi.

Ovo je dobar trenutak da podsjetimo i na postojanje programa poput *RenderDoc* koji vam može prikazati podatke koje ste slali grafičkoj kartici, a ako sumnjate na krivo orijentirane matrice, možete vrijednosti lako transponirati naredbom `glm::transpose`.

Matrica modela i matrica pogleda su u inverznom odnosu. Zamislite da su kamera i objekt u centru koordinatnog sustava i oboje "gledaju" u smjeru negativne z-osi. Nema razlike u završnoj slici ako objekt prvo zarotiramo oko x-osi i potom ga pomaknemo u smjeru negativne z-osi ili ako prvo pomaknemo kameru u smjeru pozitivne z-osi, pa ju zarotiramo oko x osi u suprotnom smjeru.

Biblioteka *glfw* prilikom definiranja *callback* funkcija za rad s korisničkim unosom očekuje funkcije definirane u globalnom prostoru, pa ih je malo nespretno enkapsulirati. Jedna od ideja je da te sve funkcije deklarirate u svojem imenovanom prostoru (engl. *namespace*) u posebnoj datoteci.

Dodatni napredniji zadaci za vježbu

- Omogućite korisniku da pritiskom na tipku `shift` ubrza pomicanje kamere.
- Iscrtajte osi globalnog i lokalnog koordinatnog sustava svakog objekta.

- Omogućite da se umjesto kamere možete registrirati bilo koji objekt za pomicanje korisničkim naredbama.