Višeplatformski softverski renderer

Štrekelj, Domagoj

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

VIŠEPLATFORMSKI SOFTVERSKI RENDERER

Diplomski rad

Domagoj Štrekelj

Osijek, 2016.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 22.09.2016.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Domagoj Štrekelj
Studij, smjer:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D 718 R, 14.10.2014.
OIB studenta:	11538063833
Mentor:	Doc.dr.sc. Irena Galić
Sumentor:	Hrvoje Leventić
Predsjednik Povjerenstva:	Doc.dr.sc. Časlav Livada
Član Povjerenstva:	Hrvoje Leventić
Naslov diplomskog rada:	Višeplatformski softverski renderer
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U radu potrebno je opisati teorijske osnove rada softverskog renderera. Za praktični dio potrebno je implementirati softverski renderer koji će se moći pokretati na više platformi, a omogućit će funkcionalnosti učitavanja modela u formatu po izboru i prikaza tog modela na ekranu. Sumentor je Hrvoje Leventić, mag.ing.comp..
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 Postignuti rezultati u odnosu na složenost zadatka: 3 Jasnoća pismenog izražavanja: 3 Razina samostalnosti: 3
Datum prijedloga ocjene mentora:	22.09.2016.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 13.10.2016.

Ime i prezime studenta:	Domagoj Štrekelj
Studij:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D 718 R, 14.10.2014.
Ephorus podudaranje [%]:	2%

Ovom izjavom izjavljujem da je rad pod nazivom**: Višeplatformski softverski renderer**

izrađen pod vodstvom mentora Doc.dr.sc. Irena Galić

i sumentora Hrvoje Leventić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Računala su danas dostupna u širokom spektru oblika i hardverskih konfiguracija, od osobnih računala i pametnih telefona, do minijaturnih računala poput *Raspberry Pi*. Grafičke kartice se u tim konfiguracijama mogu značajno razlikovati, a u nekim slučajevima ni ne postoje kao zasebna jedinica, nego su integrirane na matičnoj ploči računala. Njihova heterogenost se očituje u različitim razinama podrške za standardna sučelja za programiranje grafičkih aplikacija, kao što su *OpenGL* (*Open Graphics Library*), *Microsoft Direct3D*, i *Vulkan*, što može izolirati neke korisnike računala od reproduciranja željenog grafičkog sadržaja.

Google je prepoznao taj problem i 2009. godine razvio *SwiftShader* biblioteku kako bi omogućio podršku *WebGL* (*Web Graphics Library*) funkcionalnosti renderiranja trodimenzionalne geometrije, u internetskim preglednicima korisnika sa slabijim grafičkim karticama. Biblioteka pruža softversku implementaciju *OpenGL ES* (*Open Graphics Library for Embedded Systems*) grafičkog sučelja za programiranje aplikacija (*Application Programming Interface, API*) [1]. Time je omogućeno računalima sa grafičkim karticama bez podrške ili sa ograničenom podrškom za *WebGL* funkcionalnosti da uspješno reproduciraju *WebGL* sadržaj kroz uporabu procesora u postupku softverskog renderiranja, umjesto uobičajenog hardverskog renderiranja.

Ovim se radom želi razmotriti softversko renderiranje kao moguće rješenje za problem rastućih razlika u sposobnostima grafičkih kartica, kroz primjer izrade višeplatformskog softverskog renderera s podrškom za učitavanje i prikazivanje modela na ekranu. U teorijskom dijelu rada se stoga potkrepljuju osnovne definicije, tehnike, i postupci, što se kasnije primjenjuju u implementaciji višeplatformskog softverskog renderera pomoću programskog jezika *Haxe*.

Uvodno poglavlje rada predstavlja teorijske temelje računalne grafike i definira terminologiju koja se koristi kroz rad. Poglavlje 2 opisuje postupak renderiranja u perspektivnoj projekciji, na primjeru trokuta u prostoru. Poglavlje 3 iznosi detalje implementacije višeplatformskog renderiranja.

1.1. Zadatak diplomskog rada

U radu potrebno je opisati teorijske osnove rada softverskog renderera. Za praktični dio potrebno je implementirati softverski renderer koji će se moći pokretati na više platformi, a omogućit će funkcionalnosti učitavanja modela u formatu po izboru i prikaza tog modela na ekranu.

1.2. Terminologija

Renderiranje (engl. *rendering*) je postupak kojim se objekti i slike crtaju na zaslon [2]. Pojam „softversko renderiranje“ (engl. *software rendering*) označava programsku implementaciju tog postupka. Takvo renderiranje nije hardverski potpomognuto od strane grafičke kartice računala – što je danas uobičajena praksa – nego se izvršavanje svih potrebnih koraka renderiranja (izračuna, operacija) prepušta procesoru.

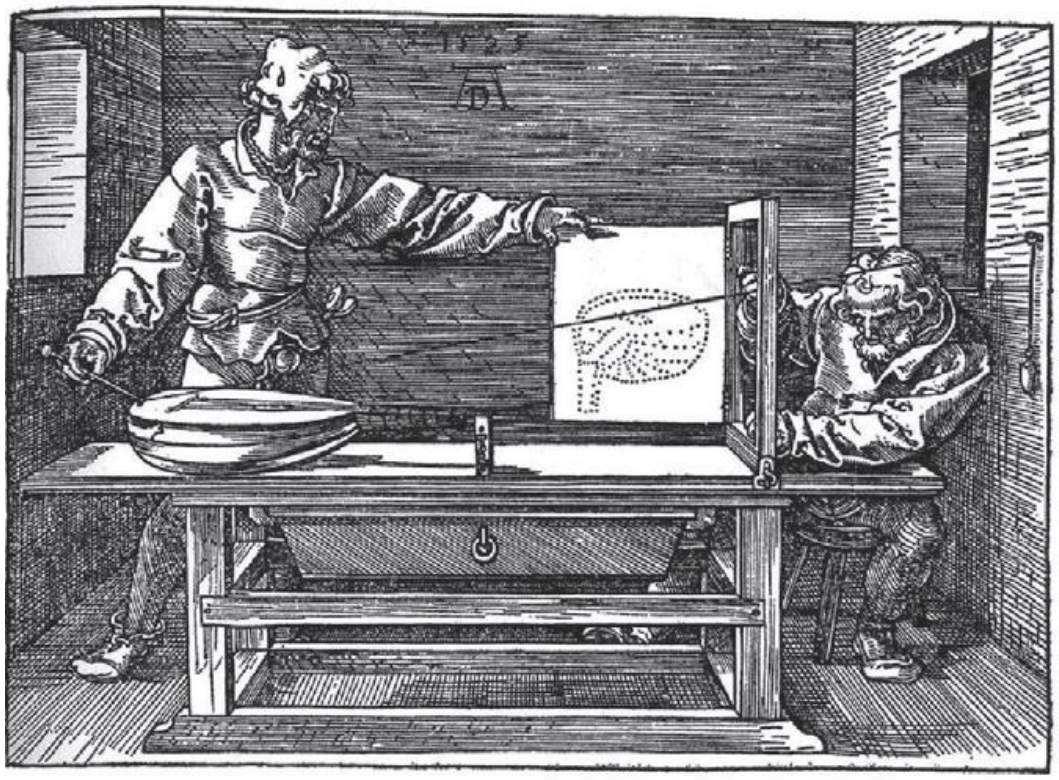
Postupak renderiranja obavlja cjevovod (engl. *pipeline*). Njegov je zadatak da na efikasan način crta projekciju trodimenzionalnih objekata (geometrijskih primitiva, najčešće trokuta) na ravninu (raster) koja predstavlja sliku objekta. Objekti su opisani točkama u trodimenzionalnom prostoru (*vertex*). Više povezanih *vertex*-a čini poligon (mnogokut) [2].

Transformacija trodimenzionalnih položaja *vertex*-a u dvodimenzionalne, koja se događa unutar *pipeline*-a, vrši se u četverodimenzionalnom prostoru. Pritom se *vertex*-i predstavljaju kao vektori sa četiri koordinate *x*, *y*, *z*, i *w*, gdje je *w* homogena koordinata koja služi za perspektivni prikaz [2]. Postupak kulminira dijeljenjem *x*, *y*, i *z* koordinata sa *w* koordinatom – što se naziva perspektivno dijeljenje (engl. *perspective division*) – nakon čega je moguće odrediti položaj točke u dvodimenzionalnom prostoru rastera [3]. Zaključno se provodi rasterizacija (engl. *rasterisation*), postupak kojim se određuje površina poligona koja je vidljiva na rasteru [2].

Produkt rasterizacije su fragmenti – skupine piksela neke slike koje pripadaju rasteriziranom poligonu. Budući da pikseli mogu pripadati fragmentima različitih poligona, potrebno je odrediti koji će od fragmenata biti vidljivi i iscrtani [2]. Vidljivost može ovisiti o više faktora kao što su položaj u prostoru (drugi poligon može zaklanjati poligon koji se trenutno rasterizira) ili način indeksiranja *vertex*-a poligona (u smjeru kazaljke na satu ili suprotno od smjera kazaljke na satu) [2].

1.3. Projekcija trodimenzionalnog objekta na plohu

Njemački umjetnik Albrecht Dürer je 1525. godine izradio drvorez koji demonstrira tehniku crtanja objekta u perspektivi. Drvorez prikazuje dvije osobe u postupku crtanja lutnje pomoću naprave izrađene za tu primjenu. Naprava nalikuje stolu te se sastoji od nekoliko dijelova, kao što je vidljivo na slici 1.1. [2].

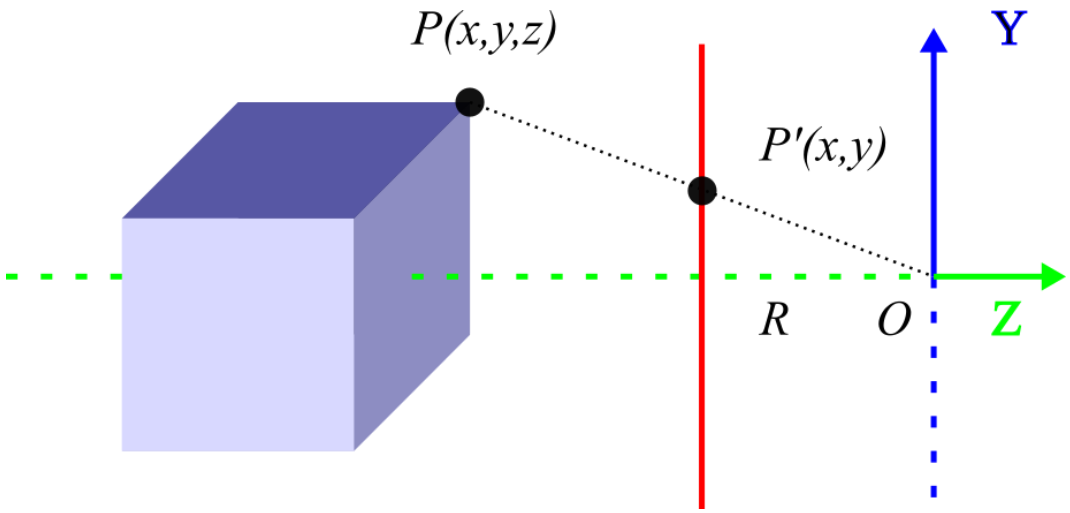


Sl. 1.1. Dürerov drvorez naprave za crtanje objekta u perspektivi. Izvor: [2]

Na jednom kraju stola nalazi se pravokutan okvir za kojeg je – poput prozorskog krila – pričvršćeno platno. Na ovom položaju nalazi se osoba koja vrši ulogu slikara. Na drugom kraju stola, nasuprot okviru, nalazi se promatrani objekt. Iza crtača je na zid pričvršćena karika koja predstavlja gledište, odnosno položaj oka što promatra objekt. Kroz kariku se provlači nit koja se u fizikalnom smislu može smatrati putanjom snopa fotona što se odbijaju od objekta i dolaze do oka (karike) što promatra objekt. Jedan kraj niti prolazi kroz okvir i dolazi u ruke druge osobe što sudjeluje u crtanju objekta. Ta osoba vrši uzorkovanje vidljivih točaka promatranog objekta. Na drugi kraj niti, onaj provučen kroz kariku, pričvršćen je uteg kako bi nit ostala napeta. Sa svakim uzorkovanjem točke objekta, slikar postavlja kist na mjesto gdje nit prolazi kroz okvir. Zatim se nit uklanja i platno zatvara preko okvira kako bi slikar mogao označiti vidljivu točku na platnu. Rezultat ove tehnike crtanja je slika objekta sačinjena od skupa uzorkovanih vidljivih točaka [2].

Promatranjem točaka na platnu kao pojedinačnih piksela digitalne slike, moguće je promatrati Dürerovu tehniku u kontekstu računalne grafike. Položaj karike postaje položaj kamere što promatra virtualni svijet, a platno postaje ravnina na koju se trodimenzionalne točke projiciraju u dvodimenzionalne, s obzirom na trenutni položaj i smjer gledanja kamere.

Kako bi se izveo matematički opis postupka projekcije potrebno je učiniti tri pretpostavke. Prvo, pretpostavlja se kako je u prostoru definiran desni pravokutni koordinatni sustav. Drugo, pretpostavlja se kako kamera promatra prostor iz ishodišta koordinatnog sustava niz *z*-os u negativnom smjeru. Treće, pretpostavlja se kako je projekcijska ploha od kamere udaljena za jednu prostornu jedinicu u negativnom smjeru *z*-osi [3].



Sl. 1.2. Geometrijski prikaz Dürerove tehnike crtanja objekta u perspektivi

Slika 1.2. ilustrira geometrijske osnove Dürerove tehnike, pod uvjetom učinjenih pretpostavki. Problem predstavlja projekcija trodimenzionalne točke *P* objekta u dvodimenzionalnu točku *P'* na projekcijsku ravninu *R*, s obzirom na ishodište koordinatnog sustava *O*.

Problemu se pristupa promatranjem ilustracije u kontekstu dva slična trokuta $\Delta ORP'_Y$ i ΔOP_ZP_Y. Budući da je u pitanju prikaz scene u *yz* ravnini, određivat će se *y* koordinata projicirane točke *P'*. Promatranjem omjera stranica dva slična trokuta izvodi se relacija (1-1) za P'_Y. Analogno tomu izvodi se relacija (1-2) za P'_X.

$$\frac{\overline{OR}}{\overline{OP_z}} = \frac{\overline{DP'_y}}{\overline{EP'_z}} \rightarrow \frac{-1}{P_z} = \frac{P'_y}{P_y} \rightarrow P'_y = \frac{-1}{P_z} \cdot P_y \qquad (1\text{-}1)$$

$$P'_x = \frac{-1}{P_z} \cdot P_x \qquad (1\text{-}2)$$

Iz relacija (1-1) i (1-2) vidi se odnos *x* i *y* koordinata točke *P* sa njenom *z* koordinatom. Točke udaljenije od ishodišta – a samim time i od projekcijske plohe – projiciraju se bliže središtu plohe. Točke bliže ishodištu projiciraju se dalje od središta plohe. Time se ostvaruje središnja točka

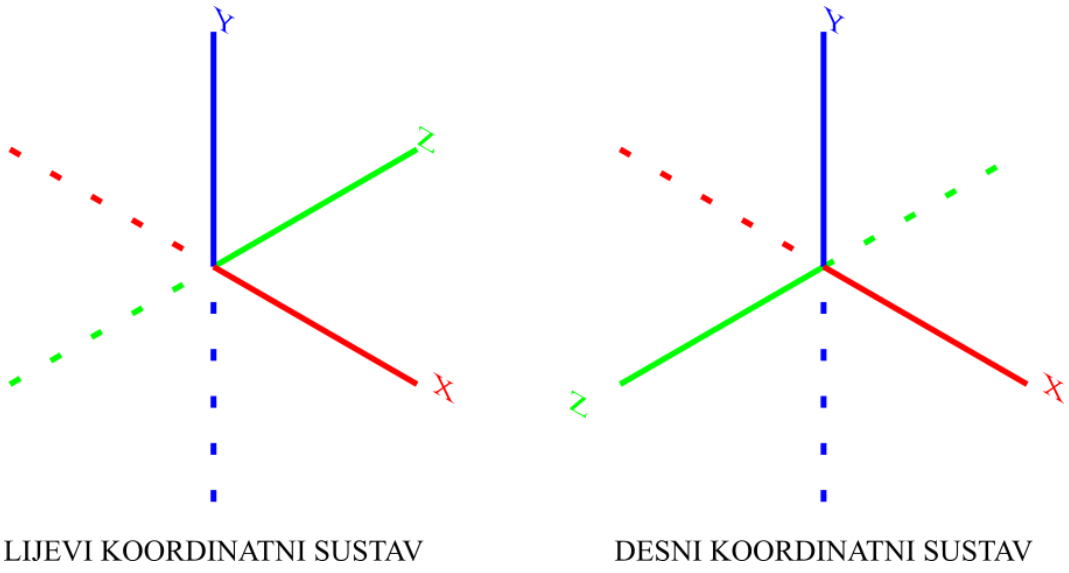
iščezavanja (engl. *vanishing point*) kojom se postiže perspektivna deformacija objekta, odnosno deformacija njegove projekcije [3].

Odabir vidljivih točaka svodi se na definiranje raspona unutar kojeg se vrijednosti *x* i *y* koordinata moraju nalaziti kako bi se točke razmatrale pri rasteriziranju. Odabrane točke prolaze kroz daljnje transformacije kojima se njihove koordinate preslikavaju u koordinate piksela na rasteru. Rezultat je slika objekta sačinjena od točaka, sa stajališta kamere što promatra taj objekt u virtualnom svijetu [3].

Opisani postupak projekcije je mali dio većeg postupka renderiranja, o kojemu se detaljnije piše u drugom poglavlju.

1.4. Matematika trodimenzionalnog prostora

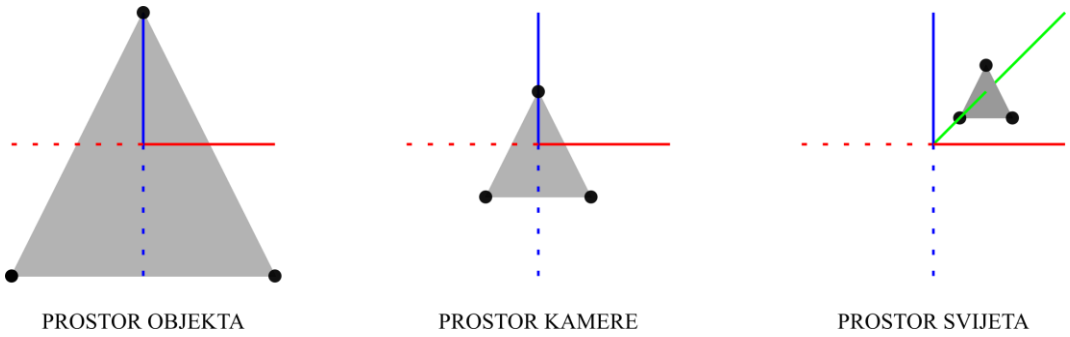
Matematika trodimenzionalnog prostora zasniva se na analitičkom rješavanju geometrijskih problema s obzirom na koordinatni sustav koji je definiran u prostoru [4].



Sl. 1.3. Usporedba lijeve i desne orijentacije trodimenzionalnog koordinatnog sustava

Za definiranje pravokutnog trodimenzionalnog koordinatnog sustava potrebno je definirati ishodište i tri koordinatne osi koje polaze od ishodišta. Koordinatne osi su međusobno okomite i označavaju se sa *x*, *y*, i *z*. Prema orijentaciji koordinatnih osi, sustav može biti definiran kao lijevi ili desni koordinatni sustav, što je prikazano slikom 1.3.. Definiranjem koordinatnog sustava definira se i koordinatni prostor koji služi kao polazište pri određivanju položaja točaka što se u njemu nalaze [4].

Jedan koordinatni prostor može sadržavati više drugih koordinatnih prostora, tvoreći tako hijerarhiju koordinatnih prostora. Matični koordinatni prostor kojemu ostali koordinatni prostori pripadaju, a sam ne posjeduje roditeljski koordinatni prostor, naziva se prostor svijeta [4].



Sl. 1.4. Prikaz položaja točaka objekta u različitim koordinatnim prostorima

Točke nekog objekta se mogu promatrati kroz više koordinatnih prostora, što je prikazano primjerom na slici 1.4.. U prostoru objekta, položaj točaka je opisan u odnosu na neku ishodišnu točku poput lokacije težišta objekta. U prostora oka što promatra objekt, njihov položaj je opisan u odnosu na ishodišnu točku oka. U prostoru svijeta, pak, točke se opisuju u odnosu na ishodište svijeta.

Točke u prostoru su opisane *x*, *y*, i *z* koordinatama koje predstavljaju udaljenosti od *yz*, *xz*, i *xy* ravnina što ih čine osi koordinatnog sustava. Položaj točaka se može izražavati u različitim koordinatnim prostorima kroz uporabu prostornih transformacija. Prostorna transformacija podrazumijeva rotaciju i translaciju koordinatnih osi jednog prostora tako da se preklapaju sa koordinatnim osima drugog prostora u kojem se točke žele izraziti [4]. Bitno je napomenuti kako se pri tome položaj točaka zapravo ne mijenja, nego se mijenja referentna točka u odnosu na koju se položaj točaka opisuje.

U geometrijskom smislu, pod pretpostavkom euklidskog prostora, vektor je usmjerena dužina koju opisuju duljina, smjer, i orijentacija, ali ne i položaj u prostoru. Točka, pak, nema duljinu, smjer, ni orijentaciju, već ju opisuje samo položaj. Iako su u ovom pogledu razlike značajne, matematički su točke i vektori ekvivalentni [4].

Matematički, vektor je *n*-torka realnih brojeva, gdje *n* označava dimenziju vektora. Geometrijski vektor ili točka mogu se matematički prikazati trodimenzionalnim vektorom. Pritom pojedini element vektora sadrži koordinatu s obzirom na jednu od koordinatnih osi prostora [4]. U radu se vektori zapisuju kao vektor redak te su na taj način implementirani u programskom kôdu višeplatformskog softverskog renderera.

Koordinatni prostori pronalaze svoj matematički ekvivalent u matricama. Matrice predstavljaju *n*-torke vektora, stoga se neki *n*-dimenzionalni koordinatni prostor može matrično prikazati vektorima baze tog koordinatnog prostora. Množenjem vektora točke s matricom što sadrži vektore baze koordinatnog prostora provodi se transformacija te točke u koordinatni prostor što koristi navedenu bazu [4].

Transformacije uključuju linearne i afine transformacije [4]. Linearne transformacije održavaju paralelnost postojećih linija u prostoru, te očuvaju svojstva homogenosti i aditivnosti transformiranih vektora. Linearne transformacije, između ostalog, uključuju rotaciju, skaliranje, i ortogonalnu projekciju. Definiraju se trodimenzionalnom kvadratnom matricom [4].

$$\mathbf{M}_{\text{LINEARNA}} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z \\ \mathbf{y}_x & \mathbf{y}_y & \mathbf{y}_z \\ \mathbf{z}_x & \mathbf{z}_y & \mathbf{z}_z \end{bmatrix}$$

Afine transformacije su spoj linearne transformacije i translacije. Svaka se linearna transformacija stoga može postići afinom transformacijom, ali obratno ne vrijedi. Afine transformacije se definiraju kvadratnom matricom sa četiri dimenzije, gdje upravo četvrta dimenzija služi za provođenje translacije [4].

$$\mathbf{M}_{\text{AFINA}} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z & \mathbf{x}_w \\ \mathbf{y}_x & \mathbf{y}_y & \mathbf{y}_z & \mathbf{y}_w \\ \mathbf{z}_x & \mathbf{z}_z & \mathbf{z}_z & \mathbf{z}_w \\ \mathbf{w}_x & \mathbf{w}_y & \mathbf{w}_z & \mathbf{w}_w \end{bmatrix}$$

Zbog četvrte dimenzije se točkama i vektorima koji su prethodno definirani u trodimenzionalnom prostoru treba dodijeliti i četvrta koordinata – homogena koordinata *w* – kako bi se nad njima mogle provoditi afine transformacije. Kako vektori nisu opisani položajem, oni ne mogu biti translatirani, zbog čega njihova *w* koordinata iznosi 0. Točkama – zbog posjedovanja svojstva položaja – *w* koordinata iznosi 1 [4].

S obzirom na navedeno, *pipeline* renderiranja spomenut u poglavlju 1.3. može se promatrati kao niz transformacija točaka iz jednog koordinatnog prostora u drugi. Pri tome zadnji koordinatni prostor prije rasterizacije uvijek bude koordinatni prostor rastera. U tom prostoru koordinate točaka odgovaraju položajima piksela u rasteru, što olakšava postupak rasterizacije.

Koraci *pipeline*-a se stoga mogu sažeto prikazati sa pet točaka [5]:

1. Transformacija *vertex*-a iz prostora svijeta u prostor kamere što gleda na svijet.

2. Transformacija *vertex*-a u prostor projekcije (i filtriranje vidljive geometrije);
3. Transformacija *vertex*-a u prostor normaliziranih koordinata;
4. Trasnformacija *vertex*-a u prostor rastera;
5. Rasterizacija.

Navedeni koraci implementirani su u višeplatformskom softverskom rendereru te se podrobnije opisuju u poglavlju 3. Konkretnija primjena matematike trodimenzionalnog prostora demonstrirana je u poglavlju 2.

2. POSTUPAK RENDERIRANJA

U ovome se poglavlju opisuje postupak renderiranja na temelju kojeg se implementirao višeplatformski softverski renderer, te se demonstrira primjena matematike trodimenzionalnog prostora kroz određivanje jednadžbi projiciranih točaka, transformacijskih, i projekcijskih matrica.

2.1. Definiranje vidljivog obujma prostora

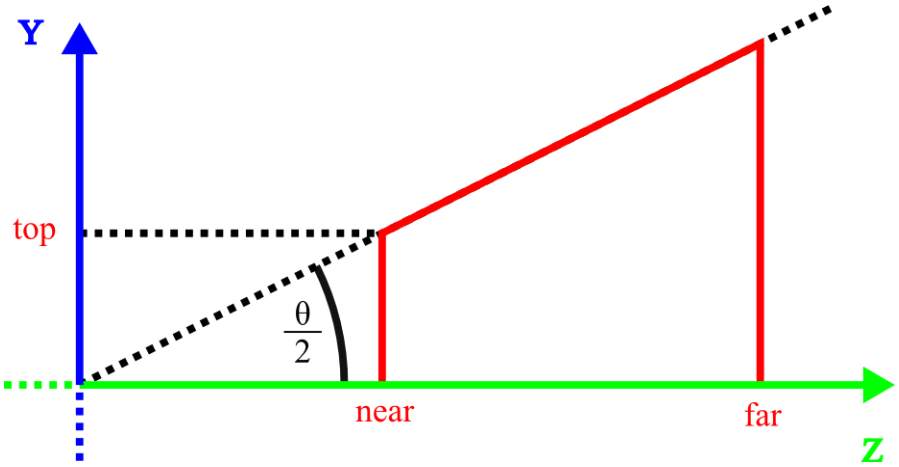
Za početak se pretpostavlja kako promatrač promatra prostor u kojemu je definiran desni pravokutni koordinatni sustav. Gledište promatrača nalazi se u ishodištu koordinatnog sustava i usmjereno je u negativnom smjeru niz *z*-os. Time je postavljen uvjet da se objekti ispred promatrača moraju nalaziti na negativnoj strani *z*-osi, odnosno da moraju imati negativnu *z* koordinatu.

Promatrač vidi jedan konačan i ograničen dio prostora što se nalazi ispred njega. Taj dio prostora je njegov vidljivi obujam (engl. *view volume*) [2].

Vidljivi obujam se opisuje sa tri para ravnina: bližom (engl. *near*) i daljom (engl. *far*); lijevom (engl. *left*) i desnom (engl. *right*); donjom (engl. *bottom*) i gornjom (engl. *top*). Način na koji se ravnine određuju ovisi o načinu na koji se točke prostora projiciraju na projekcijsku ravninu [2].

Dvije česte projekcijske metode su perspektivna, i ortogonalna projekcija. Ključna razlika između navedenih metoda je što ortogonalna projekcija očuva postojeće paralelne linije, dok perspektivna projekcija uvodi točku iščezavanja u kojoj se sve linije eventualno sijeku. Vidljivi obujam kod ortogonalne projekcije stoga poprima oblik kocke ili kvadra, dok kod perspektivne projekcije on predstavlja krnju piramidu [2].

U nastavku se razmatra perspektivna projekcija, budući da je ona implementirana u višeplatformskom softverskom rendereru. Izvod i objašnjenje ortogonalne projekcije dostupno je u prilogu 1.



Sl. 2.1. Određivanje parametara vidljivog obujma

Krnja piramida vidljivog obujma opisana je bližom, daljom, gornjom, donjom, lijevom, i desnom stranicom, koje su redom označene *near, far, top, bottom, left,* i *right* [2]. Sagleda li se vidljivi obujam na *yz* ravnini prema slici 2.1., može se uočiti veza između stranica *near* i *top* te (vertikalnog) kuta gledanja θ.

$$top = \tan\frac{\theta}{2} \cdot near \tag{2-1}$$

$$bottom = -top \tag{2-2}$$

Položaj stranica *left* i *right* ovisi o stranicama *bottom* i *top* te omjeru širine i visine slike koja se rasterizira (*aspect ratio*) [2].

$$right = aspect\ ratio \cdot top \tag{2-3}$$

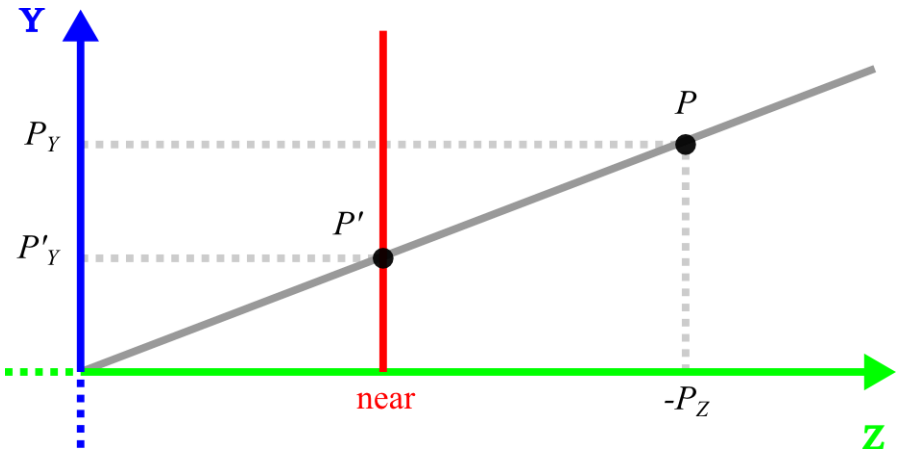
$$left = -right \tag{2-4}$$

Relacije (2-1), (2-2), (2-3), i (2-4) čine parametre vidljivog obujma perspektivne projekcije. Potrebno je napomenuti kako je vidljivi obujam radi intuitivnosti parametara postavljen na pozitivnu stranu *z*-osi koordinatnog prostora. Budući da se objekti nalaze na negativnoj strani, *z* koordinate točaka objekata će se morati negirati pri projekcijskoj transformaciji.

2.2. Projekcija s obzirom na vidljivi obujam

U poglavlju 1.3. ustanovljeno je kako se projekcija provodi relacijama (1-1) i (1-2). Međutim, navedene su relacije izvedene pod pretpostavkom da je projekcijska ravnina od gledišta uvijek udaljena jednu prostornu jedinicu. Uvođenjem vidljivog obujma u postupak projekcije ta

pretpostavka više nije valjana. Razlog tomu je što se položaj projekcijske ravnine sada poistovjećuje sa položajem *near* stranice obujma [2].



Sl. 2.2. Projekcija točke na projekcijsku ravninu vidljivog obujma

I dalje je potrebno negirati *z*-koordinatu promatranih točaka kako bi se one mogle pronaći unutar vidljivog obujma. Ali, uloga *near* ravnine kao projekcijske ravnine sada uvodi novu varijablu, kao što je prikazano na slici 2.2..

$$P'_X = near \cdot \frac{P_X}{-P_Z} \tag{2-5}$$

$$P'_Y = near \cdot \frac{P_Y}{-P_Z} \tag{2-6}$$

Točke koje se kvalificiraju za rasterizaciju moraju biti unutar vidljivog obujma, što znači da moraju biti omeđene ravninama što ga opisuju. U slučaju *x* koordinate točke, ona se mora nalaziti između *left* i *right* ravnina [2].

$$left \leq P_X \leq right$$

Pogodno je u ovom koraku definirati raspon u kojem će se nalaziti vrijednost *x* koordinate projicirane točke P'. Koordinate u ovom rasponu predstavljat će normalizirane koordinate točke, odnosno njen položaj u prostoru normaliziranih koordinata [2]. Koordinate *x* i *y* normalizirat će se u raspon $[-1,1]$, a koordinata *z* u raspon $[0,1]$.

$$0 \leq P_X - left \leq right - left$$

$$0 \leq \frac{2 \cdot (P_X - left)}{right - left} \leq 2$$

$$-1 \leq \frac{2 \cdot P_X}{right - left} - \frac{right + left}{right - left} \leq 1 \tag{2-7}$$

Uvođenjem relacije (2-5) u relaciju (2-7) dobiva se jednadžba (2-8) za projiciranu *x* koordinatu točke. Analogno se dobiva i jednadžba (2-8) za projiciranu *y* koordinatu točke.

$$P'_X \cdot (-P_Z) = \frac{2 \cdot near}{right - left} \cdot P_X - \frac{right + left}{right - left} \cdot (-P_Z) \tag{2-8}$$

$$P'_Y \cdot (-P_Z) = \frac{2 \cdot near}{top - bottom} \cdot P_Y - \frac{top + bottom}{top - bottom} \cdot (-P_Z) \tag{2-9}$$

Kod određivanja jednadžba projicirane *z* koordinate (2-10) polazi se od graničnih slučajeva za koje su rezultati poznati. Pokušava se dobiti jednadžba oblikom slična jednadžbama (2-7) i (2-8).

$$0 \leq P'_Z \leq 1$$

$$P'_Z \cdot (-P_Z) = A \cdot P_Z + B$$

$$-P_Z = near \rightarrow P'_Z = 0 \rightarrow B = A \cdot near$$

$$-P_Z = far \rightarrow P'_Z = 1 \rightarrow A = \frac{far}{near - far}$$

$$P'_Z \cdot (-P_Z) = \frac{far}{near - far} \cdot P_Z + \frac{near \cdot far}{near - far} \tag{2-10}$$

Jednadžbe (2-8), (2-9), i (2-10) prikazuju linearnu ovisnost o varijabli $-P_Z$. Ona se pridodaje homogenoj koordinati, kojom se naposljetku dijele *x*, *y*, i *z* koordinate točke u postupku perspektivnog dijeljenja, i dobiju njihove vrijednosti u prostoru normaliziranih koordinata. Rezultat se može prikazati matrično u obliku transformacijske matrice perspektivne projekcije (2-11) [2].

$$\mathbf{M}_{\mathbf{PERSPEKTIVNO}} = \begin{bmatrix} \frac{2 \cdot near}{right - left} & 0 & 0 & 0 \\ 0 & \frac{2 \cdot near}{top - bottom} & 0 & 0 \\ -\frac{right + left}{right - left} & -\frac{top + bottom}{top - bottom} & \frac{far}{near - far} & -1 \\ 0 & 0 & \frac{near \cdot far}{near - far} & 0 \end{bmatrix} \tag{2-11}$$

2.3. Primjer projekcije trokuta

U promatranom prostoru nalazi se trokut opisan točkama *A*(-1, -2, -8), *B*(1, -2, -4), i *C*(1, 2, 5). Njihov vektorski zapis u trodimenzionalnom prostoru glasi:

$$A(-1,-2,-8) \rightarrow \mathbf{a} = [-1 \quad -2 \quad -8]$$

$$B(1,-2,-4) \rightarrow \mathbf{b} = [1 \quad -2 \quad -4]$$

$$C(1,2,5) \rightarrow \mathbf{c} = [1 \quad 2 \quad -5]$$

Promatrač vrši perspektivnu projekciju trokuta s obzirom na vertikalni kut gledanja od 60°. Definira raster širine 800 px i visine 600 px, što daje omjer širine i visine rastera 4:3. Promatrač odabire razmatrati točke udaljene minimalno 1 prostornu jedinicu, a maksimalno 100 prostornih jedinica od njegovog gledišta. Temeljem zadanih parametara i jednadžbi (2-1), (2-2), (2-3), i (2-4) dobivaju se ravnine vidljivog obujma.

$$top = 0{,}5774$$

$$bottom = -0.5774$$

$$right = 0{,}7698$$

$$left = -0{,}7698$$

Uvrštavanjem dobivenih parametara vidljivog obujma u projekcijsku matricu (2-11) dobivaju se vrijednosti transformacijske matrice.

$$\mathbf{M_{PERSPEKTIVNO}} = \begin{bmatrix} 1{,}299 & 0 & 0 & 0 \\ 0 & 1{,}7319 & 0 & 0 \\ 0 & 0 & -1.0101 & -1 \\ 0 & 0 & -1.0101 & 0 \end{bmatrix}$$

Prije vršenja perspektivne transformacije, točkama se dodjeljuje homogena (*w*) koordinata s iznosom 1. Potom se može vršiti transformacija, koja točke prebacuje iz prostora kamere u prostor projekcije.

$$\mathbf{a}' = \mathbf{a} \cdot \mathbf{M_{PERSPEKTIVNO}} = [-1{,}299 \quad -3{,}4638 \quad 7{,}0707 \quad 8]$$

$$\mathbf{b}' = \mathbf{b} \cdot \mathbf{M_{PERSPEKTIVNO}} = [1{,}299 \quad -3{,}4638 \quad 3{,}0303 \quad 4]$$

$$\mathbf{c}' = \mathbf{c} \cdot \mathbf{M_{PERSPEKTIVNO}} = [1{,}299 \quad 3{,}4638 \quad 4{,}0404 \quad 5]$$

Prebacivanje točaka iz prostora projekcije u prostor normaliziranih koordinata vrši se perspektivnim dijeljenjem, odnosno dijeljenjem *x*, *y*, i *z* koordinata sa *w* koordinatom. Na mjestu *w* koordinate pohranjuje se njena recipročna vrijednost te se ona koristi dalje u postupku rasterizacije.

$$\mathbf{a}' = \left[\frac{-1,299}{8} \quad \frac{-3,4638}{8} \quad \frac{7,0707}{8} \quad \frac{1}{8}\right]$$

$$\mathbf{a}' = [-0,1624 \quad -0,4329 \quad 0,8838 \quad 0,125]$$

$$\mathbf{b}' = \left[\frac{1,299}{4} \quad \frac{-3,4638}{4} \quad \frac{3,0303}{4} \quad \frac{1}{4}\right]$$

$$\mathbf{b}' = [0,3248 \quad -0,866 \quad 0,7576 \quad 0,25]$$

$$\mathbf{c}' = \left[\frac{1,299}{5} \quad \frac{3,4638}{5} \quad \frac{4,0404}{5} \quad \frac{1}{5}\right]$$

$$\mathbf{c}' = [0,2598 \quad 0,6928 \quad 0,8081 \quad 0,2]$$

U ovom se trenutku – ukoliko postoje – dijele i atributi koji su vezani uz transformiranu točku. To su skalari, primjerice boja, koordinate uzorka teksture, koordinate vektora normale, i slično. Pri projekciji posjeduju istu ovisnost o -P_Z koja se javlja u relacijama (2-8), (2-9), i (2-10) koju je potrebno korigirati perspektivnim dijeljenjem [6].

Koordinate projiciranih točaka *x* i *y* nalaze se u rasponu $[-1,1]$. U prostoru normaliziranih koordinata, one se nalaze oko ishodišta koordinatnog sustava s obzirom na ravninu *xy*. Stoga se skaliranjem i translatiranjem točaka prema dimenzijama rastera one mogu transformirati u prostor rastera [2]. Matrica (2-12) predstavlja transformacijsku matricu raster prostora (2-12).

$$\mathbf{M_{RASTER}} = \begin{bmatrix} \frac{širina}{2} & 0 & 0 & 0 \\ 0 & \frac{visina}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{širina}{2} & \frac{visina}{2} & 0 & 1 \end{bmatrix} \tag{2-12}$$

Međutim, s obzirom na postupak renderiranja predstavljen u ovom primjeru, primjena navedene transformacijske matrice neće dati očekivane rezultate.

U prethodnom koraku se tokom perspektivnog dijeljenja u *w* koordinatu pohranjuje njena recipročna vrijednost. Iako je ta vrijednost potrebna za postupak interpolacije atributa tokom rasterizacije, ona u pravilu nije točna. Perspektivno dijeljenje se provodi dijeljenjem svih

koordinata točke sa *w* koordinatom – uključujući i samu *w* koordinatu – iz čega proizlazi da *w* nakon perspektivnog dijeljenja ima vrijednost 1. U tom slučaju će transformacijska matrica dati očekivane rezultate jer će se translacija provesti prema predviđenim pravilima.

Privremena promjena vrijednosti *w* koordinate je implementacijski detalj koji je bilo potrebno posebno natuknuti u ovom poglavlju rada. Stoga se u kontekstu ovog primjera *w* koordinata u vektorima **a'**, **b'**, i **c'** mora privremeno postaviti na 1 kako bi transformacijska matrica (2-12) pravilno djelovala. Vrijednost *w* se nakon transformacije vraća na prethodnu radi uporabe pri rasterizaciji.

$$A'_X = \frac{\check{s}irina}{2} \cdot (\mathbf{a}'_X + 1) = 335{,}04\ px \approx 335\ px$$

$$A'_Y = \frac{visina}{2} \cdot (\mathbf{a}'_Y + 1) = 170{,}13\ px \approx 170\ px$$

$$B'_X = \frac{\check{s}irina}{2} \cdot (\mathbf{b}'_X + 1) = 529{,}92\ px \approx 529\ px$$

$$B'_Y = \frac{visina}{2} \cdot (\mathbf{b}'_Y + 1) = 40{,}2\ px \approx 40\ px$$

$$C'_X = \frac{\check{s}irina}{2} \cdot (\mathbf{c}'_X + 1) = 503{,}92\ px \approx 503\ px$$

$$C'_Y = \frac{visina}{2} \cdot (\mathbf{c}'_Y + 1) = 507{,}84\ px \approx 507\ px$$

Za rezultat se dobiju koordinate projiciranih točaka *A'*, *B'*, i *C'* na samom rasteru, odnosno lokacije njihovih piksela na slici. Sljedeći korak čini rasterizacija objekta temeljem položaja njegovih točaka u prostoru rastera.

2.4. Rasterizacija

Proces rasterizacije obuhvaća sljedeće korake [2]:

1. Određivanje kvadratne površine rastera koja obuhvaća projekciju trokuta.
2. Obrađivanje svakog piksela navedene površine u potrazi za fragmentom trokuta.
3. Interpolacija atributa dodijeljenih točkama trokuta, na položaju fragmenta.
4. Određivanje konačnog utjecaja fragmenta na piksel rastera.

Određivanje površine rastera koja omeđuje projicirani trokut svodi se na pronalazak minimalnih i maksimalnih vrijednosti *x* i *y* koordinata među svim točkama trokuta. Vrijednosti se pri tome ograničavaju unutar dimenzija rastera, ukoliko su ih one tokom transformacije u prostor rastera premašile.

Pripadnost nekog piksela rastera trokutu određuje se na osnovu baricentričnih koordinata tog piksela u odnosu na projekciju trokuta. Piksel sa vrhovima projiciranog trokuta zatvara tri manja trokuta. Izračunom mješovitog produkta vektorskih zapisa točaka što čine pojedini trokut, dobiva se obujam paralelopipeda što ga vektori zatvaraju. Predznak obujma ukazuje na položaj piksela u odnosu na stranicu trokuta koju čine druge dvije točke [7].

$$A'(x,y) \rightarrow \mathbf{a} = [A'_X \quad A'_Y \quad 1]$$

$$B'(x,y) \rightarrow \mathbf{b} = [B'_X \quad B'_Y \quad 1]$$

$$C'(x,y) \rightarrow \mathbf{c} = [C'_X \quad C'_Y \quad 1]$$

$$(\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c} = \begin{vmatrix} \mathbf{a}_X & \mathbf{a}_Y & 1 \\ \mathbf{b}_X & \mathbf{b}_Y & 1 \\ \mathbf{c}_X & \mathbf{c}_Y & 1 \end{vmatrix} \tag{2-13}$$

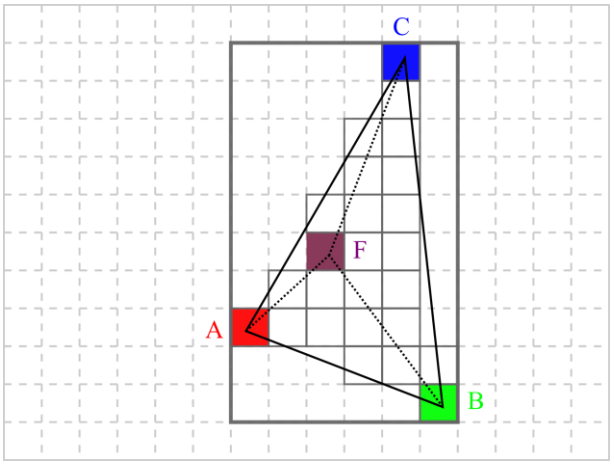
Rješavanjem relacije (2-13) s obzirom na vektor **c** dobiva se relacija (2-14). Ako vektori **a**, **b**, i **c** predstavljaju vektorske zapise točaka *A*, *B*, i *C* trokuta u prostoru rastera, onda se iz rezultata korištenja relacije (2-14) može utvrditi položaj točke *C* u odnosu na stranicu trokuta koju tvore točke *A* i *B*.

$$Produkt(\mathbf{a},\mathbf{b},\mathbf{c}) = \mathbf{c}_X \cdot (\mathbf{a}_Y - \mathbf{b}_Y) - \mathbf{c}_Y \cdot (\mathbf{a}_X - \mathbf{b}_X) + (\mathbf{a}_X \cdot \mathbf{b}_Y - \mathbf{a}_Y \cdot \mathbf{b}_X) \tag{2-14}$$

Takav izračun se provodi za svaki piksel unutar kvadratne površine nad kojom se provodi rasterizacija, u odnosu na sve tri stranice trokuta. Ovisno o redoslijedu kojim se obilaze točke novostvorenih trokuta, svi obujmi moraju biti pozitivni, odnosno negativni, kako bi piksel bio fragment trokuta kojeg se rasterizira. U tom slučaju se oni dijele sa obujmom paralelopipeda dobivenog uvrštavanjem točaka projiciranog trokuta u relaciju (2-14). Rezultat je udjel svake točke projiciranog trokuta u promatranom pikselu rastera, odnosno fragmentu trokuta [7].

Temeljem udjela točaka trokuta u promatranom fragmentu vrši se interpolacija atributa tih točaka do položaja fragmenta. Atributi su prethodno podlegli postupku perspektivnog dijeljenja zbog linearne ovisnosti o homogenoj koordinati (*w*) kao rezultat projiciranja iz trodimenzionalnog u dvodimenzionalni prostor. Njihova interpolacija u dvodimenzionalnom prostoru se stoga mora

vršiti s obzirom na interpoliranu *w* koordinatu, koja je u vektorskom prikazu točke u prostoru normaliziranih koordinata pohranjena kao $1/w$ [6].



Sl. 2.3. Ilustracija rasterizacije trokuta

Rasterizacija trokuta iz primjera ilustrirana je slikom 2.3., uz malu izmjenu dodavanja točkama *A*, *B*, i *C* atribut boje crvene, plave, i zelene radi boljeg predočavanja rezultata rasterizacije.

Interpolacija *w* koordinate prema (2-14) glasi:

$$P_{ABC} = Produkt(\mathbf{a}, \mathbf{b}, \mathbf{c})$$

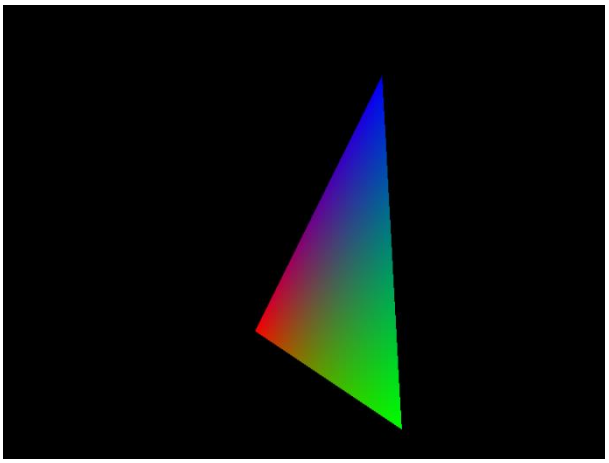
$$P_{ABF} = Produkt(\mathbf{a}, \mathbf{b}, \mathbf{f})/P_{ABC}$$

$$P_{BCF} = Produkt(\mathbf{b}, \mathbf{c}, \mathbf{f})/P_{ABC}$$

$$P_{CAF} = Produkt(\mathbf{c}, \mathbf{a}, \mathbf{f})/P_{ABC}$$

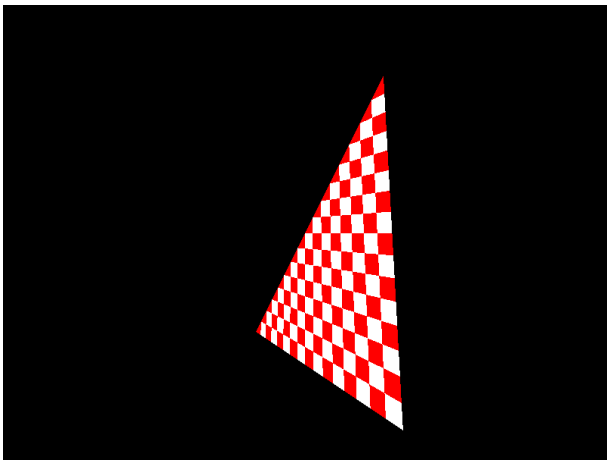
$$F_W = A_W \cdot P_{BCF} + B_W \cdot P_{CAF} + C_W \cdot P_{ABF} \qquad (2\text{-}15)$$

Analogno (2-15) vrše se interpolacije *z* koordinate i atributa točke. Za dobivanje točne vrijednosti atributa fragmenta potrebno je interpoliranu vrijednost atributa pomnožiti sa $1/F_W$, jer su atributi bili dijeljeni sa *w* koordinatom tokom perspektivnog dijeljenja.



Sl. 2.4. Rasterizirani trokut s definiranim atributom boje na točkama

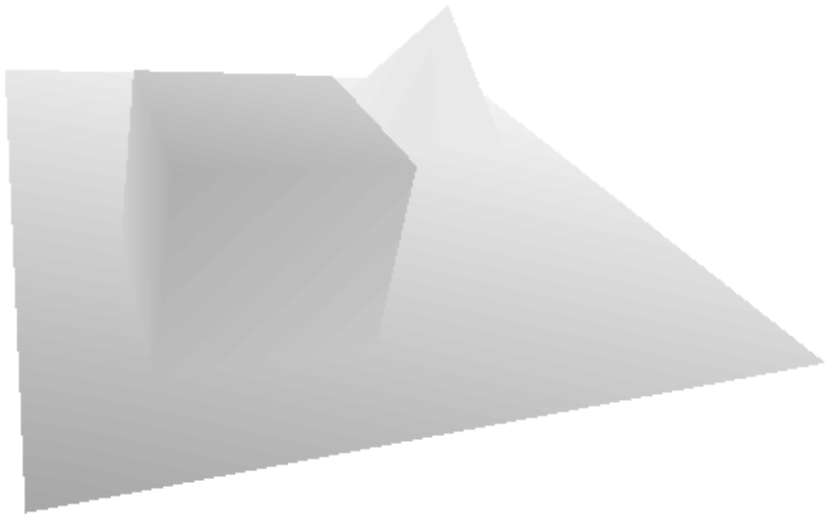
Rezultat rasterizacije trokuta iz primjera prikazana je slikom 2.4.. Može se uočiti kako točke udaljenije od gledišta svojim atributima manje utječu na konačni izgled fragmenta, što je utjecaj perspektivne projekcije. Na slici 2.5. prikazana je rasterizacija istog trokuta, ali s definiranim atributom koordinata uzorka teksture. Iluzija perspektive se jasno očituje u veličini kvadrata teksture u odnosu na položaj točke po z osi.



Sl. 2.5. Rasterizirani trokut s definiranim atributom koordinata uzorka teksture na točkama

Zapisivanje fragmenta u raster ovisi prvenstveno o njegovom položaju u prostoru. Fragmenti koji su bliži gledištu imaju prioritet nad onima koji su više udaljeni od gledišta. Prije odlučivanja hoće li se fragment zapisati u raster vrši se provjera odnosa njegove z koordinate sa z koordinatom fragmenta koji je na tom mjestu u rasteru već zapisan. Budući da su z koordinate normalizirane u raspon [0,1], onaj fragment sa z koordinatom bližoj 0 bit će zapisan u raster. Boja piksela rastera pri tome ovisi o atributima fragmenta [6].

Slika 2.6. prikazuje rezultat sjenčanja fragmenta rasterizacije sa vrijednosti njegove *z* koordinate. Vrijednost *z* bliža gledištu daje tamniju nijansu sive, dok ona udaljenija od gledišta daje svjetliju nijansu sive. Dugim riječima, raspon $[0,1]$ se preslikava u nijanse sive od crne do bijele. Također je vidljivo kako se na kraju postupka rasterizacije u raster upisuju samo oni fragmenti koji su na temelju svoje interpolirane *z* koordinate pozicionirane ispred ostalih, odnosno bliže gledištu.



Sl. 2.6. Rasterizacija scene kroz sjenčanje fragmenata na osnovu z koordinate

3. VIŠEPLATFORMSKI SOFTVERSKI RENDERER

U ovom poglavlju opisana je implementacija višeplatformskog softverskog renderera. Opisuje se implementacijsko okruženje, dizajn implementacije, te specifičnosti oko podržavanja više platformi.

3.1. Implementacijsko okruženje

Za implementacijsko okruženje odabran je programski jezik *Haxe*. *Haxe* je skup alata za višeplatformski razvoj koji objedinjuje programski jezik, prevoditelj, i standardnu biblioteku sa višeplatformskom podrškom. Višeplatformskom razvoju pristupa prevođenjem izvornog kôda napisanog u *Haxe* jeziku, u izvorni kôd drugih (podržanih) programskih jezika [8].

Jezik je statično tipiziran te dozvoljava implicitno i eksplicitno navođenje podatkovnih tipova u programskom kôdu. Podržava više programskih paradigmi radi prilagodbe jezicima u koje se prevodi, ali je prvenstveno objektno-orijentiran. Sintaksa je inspirirana *ECMAScript* standardom [9].

Prevoditelj je optimirajuće vrste. Vrši analizu programskog kôda pri prevođenju te vrši preinake u generiranom kôdu kako bi pospješio njegovo izvođenje. Između ostalog, pruža podršku za metapodatke te – uz pomoć *macro* sustava – metaporgramiranje u trenutku prevođenja [10]. Od inačice 3.3, *Haxe* prevoditelj omogućuje prevođenje u *ActionScript 3, C#, C++, Java, JavaScript, Lua, PHP*, i *Python* izvorni kôd, te *Flash* i *Neko bytecode* [11].

Standardna biblioteka dijeli se na biblioteku opće uporabe i biblioteku za sistemsko programiranje. Biblioteka opće uporabe je podržana za sve jezike u koje se *Haxe* prevodi, dok je biblioteka za sistemsko programiranje podržana samo za one jezike koji imaju pristup podatkovnom sustavu. Uz to, svaki podržani programski jezik posjeduje svoju biblioteku koja sadrži sučelja za pristup specifičnim funkcionalnostima tog jezika [12].

Postupak renderiranja građen je na matematičkoj osnovi. S izuzetkom grafičkog prikaza rezultata, on ne posjeduje elemente koji bi uvjetovali korištenje određenog programskog jezika ili platforme. *Haxe* pruža mogućnost iskorištavanja potencijala prenosivosti programskog rješenja renderera, te nudi potrebne alate za rukovanje specifičnostima grafičkog prikaza rezultata renderiranja.

3.2. Detalji implementacije

Softverski renderer je implementiran u obliku *Haxe* biblioteke. Biblioteka je koncipirana na modularan način – logika renderiranja i logika grafičkog prikaza odvojene su u zasebne biblioteke. Svaka platforma definira svoju biblioteku za grafički prikaz, koja se u projekt uključuje po potrebi.

Biblioteka za grafički prikaz mora biblioteci za renderiranje pružati implementaciju konteksta za crtanje i implementaciju podatkovne strukture za pohranu piksela rastera, odnosno *framebuffer*.

Kontekst predstavlja prozor u kojem se vrši grafički prikaz. Pri inicijalizaciji konteksta se navode dimenzije rastera. Pri pokretanju konteksta se navode referenca na funkciju koja obnavlja predmet prikazivanja u kontekstu, i referenca na funkciju za crtanje u raster konteksta. Na raster konteksta se utječe kroz *framebuffer*, čije sučelje definira metode za čitanje i zapisivanje pojedinog piksela, te brisanje svih piksela rastera.

Biblioteka za renderiranje definira podatkovne strukture, *pipeline* korake, pomoćne klase, i ostale potrepštine za obavljanje postupka renderiranja. Logika renderiranja je građena na nekoliko temeljnih pretpostavki:

- u prostoru je definiran desni koordinatni sustav;
- kamera (promatrač) što promatra scenu u prostoru polazi od ishodišta i gleda u negativnom smjeru niz *z*-os;
- ishodište prostora rastera nalazi se u donjem lijevom kutu;
- temeljni geometrijski primitiv u postupku rasterizacije je trokut;
- vidljivi su samo oni trokuti čiji su indeksi točaka navedeni u smjeru obrnutom od smjera kretanja kazaljke na satu (sa stajališta kamere).

Osnovni element renderiranja čini geometrija, opisana klasom *Geometry*. Geometrija definira položaje točaka u prostoru (*vertex*), njihove atribute (boja, koordinate uzoraka teksture), te indekse što obilježavaju redoslijed kojim definirane točke tvore trokute. Položaj točaka opisan je u odnosu na svoj lokalni koordinatni prostor.

Geometriju koristi model, opisan klasom *Mesh*. Model geometriji dodjeljuje transformacijsku matricu, definirajući tako točke geometrije u odnosu na prostor modela. Uz to, model može posjedovati i teksturu koja se uzorkuje kroz atribute geometrije.

Kamera je opisana klasom *Camera*. Ona je svojevrsni objekt u prostoru koji se može translatirati i rotirati. Izmjena tih svojstava kamere se pohranjuje u njenoj transformacijskoj matrici, ali na suprotan način. Primjerice, translacijom kamere u desno transformacijska matrica zapisuje izvršenje pomaka u lijevo. Svi se transformirani modeli stoga translatiraju u lijevo, a kamera ostavlja dojam pomaka u prostoru, iako ostaje na istom položaju gdje se i prije nalazila. Time je stvorena transformacijska matrica prostora kamere. Uz to, kamera definira projekcijsku matricu jer je koncept projekcije usko vezan uz koncept kamere.

Za postupak renderiranja potrebno je definirati scenu i program za sjenčanje fragmenata. Scena je opisana klasom *Scene* i čine ju kamera i promatrani modeli koji se žele renderirati. Program za sjenčanje fragmenata (engl. *shader program* ili *shader*) opisan je klasom *Shader* te sadrži funkcije kojima se utječe na postupak obrade *vertex*-a i fragmenata.

3.1. *Shader*

Shader klasa pruža pristup parametrima iz različitih stadija postupka renderiranja. Definira dvije funkcije koje se tim parametrima mogu služiti pri obradi ulaznih podataka. To su *vertex shader* funkcija, koja služi za obradu točaka, i fragment *shader* funkcija, koja služi za obradu fragmenata.

Vertex shader zaprima položaj točke u prostoru i vraća transformirani položaj točke. Namijenjen je za izvršavanje transformacija točke iz jednog koordinatnog prostora u drugi. Najčešće je to niz transformacija *model-view-projection* koje točke transformiraju iz lokalnog prostora u prostor modela, zatim u prostor kamere, te na kraju u prostor projekcije [6].

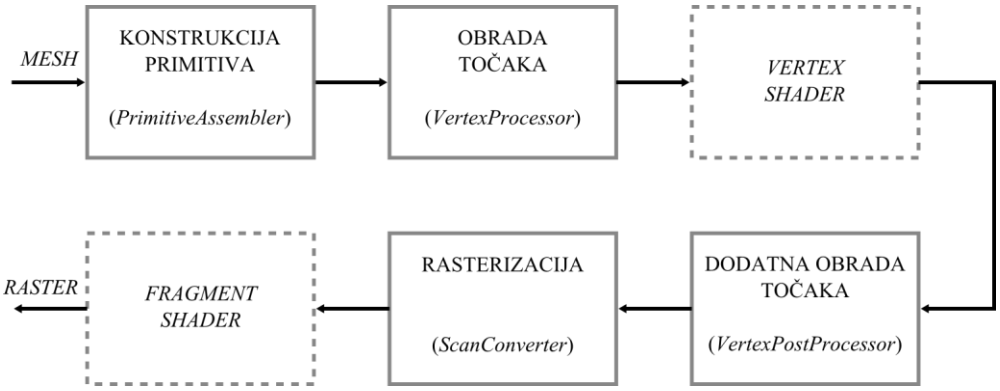
Fragment shader zaprima koordinate fragmenta i koordinate piksela rastera, i vraća vrijednost istine (*true*) ili ne (*false*) ukoliko se fragment zapisuje ili odbacuje pri rasterizaciji. Namijenjen je za sjenčanje fragmenta, odnosno određivanje njegove boje pri zapisu u raster [6].

Shader klasa je namijenjena kao klasa roditelj za *shader* programe koje korisnik biblioteke želi koristiti, jer ona sama po sebi ne vrši nikakve transformacije točaka niti sjenča fragmente.

3.2. *Pipeline*

Pipeline se inicijalizira sa dimenzijama rastera. Pri tome se priprema transformacijska matrica koja služi za transformiranje projiciranih točaka u prostor rastera. Također se kreira i *depth buffer* (*z-buffer*), podatkovna struktura unutar koje se pohranjuju vrijednosti *z* koordinate fragmenata zapisanih u raster kako bi se odredili fragmenti s većim prioritetom pri rasterizaciji [6].

Prilikom izvršavanja, *depth buffer* se poništava postavljanjem svih vrijednosti na $+\infty$ kako bi se omogućilo bilježenje *z* koordinata fragmenata, koje zbog transformacija imaju pozitivan predznak. Zatim se prolazi kroz sve modele u sceni i prosljeđuje ih u *pipeline* radi renderiranja, pri čemu se na određenim mjestima obavlja konfiguracija *shader* objekta kroz postavljanje vrijednosti transformacijskih matrica, teksture modela, i sličnih parametara.



Sl. 3.1. Prikaz implementiranog *pipeline*-a

Prikaz implementiranog *pipeline*-a dan je slikom 3.1., gdje okviri ocrtani punom linijom prikazuju korake na koje se ne može utjecati, a oni isprekidanom linijom korake koji su programibilni. Na ulazu u *pipeline* nalaze se modeli (objekti klase *Mesh*) u sceni, koji se zatim obrađuju, i na kraju budu ili ne budu rasterizirani i prikazani rasterom.

Prvi korak renderiranja je izrada geometrijskih primitiva (trokuta) na temelju geometrije modela. Ovaj korak obavlja klasa *PrimitiveAssembler*. Za ulaz prima indekse trokuta iz geometrije, kao i samu geometriju na temelju koje će konstruirati trokut te točkama dodijeliti atribute.

Drugi korak čini obrada točaka i obavlja ga klasa *VertexProcessor*. Za ulaz prima trokut i *shader* objekt. U ovom koraku se provodi transformacija točaka trokuta, najčešće uporabom transformacijskih matrica, na način koji je definiran u *vertex shader* funkciji *shader* objekta.

U trećem koraku provodi se filtriranje trokuta na način da se označavaju oni trokuti koji su vidljivi i kvalificiraju se za postupak rasterizacije. Obrada se vrši u *VertexPostProcessor* klasi. Vidljivima se smatraju samo oni trokuti koji se nalaze unutar vidljivog obujma prostora. U svrhu određivanja vidljivih trokuta koristi se koncept predstavljen u Cohen-Sutherland algoritmu za obrezivanje linija [5].

Vrši se provjera položaja točaka u odnosu na granice vidljivog obujma. Zbog ovisnosti koordinata točaka poslije projekcijske transformacije o homogenoj koordinati, granice se mogu definirati na sljedeći način:

$$-P_W \leq P_X \leq P_W$$

$$-P_W \leq P_Y \leq P_W$$

$$-P_W \leq P_Z \leq P_W$$

$$P_W > 0$$

Ako točke trokuta ne ispunjavaju sve navedene uvjete, trokut se neće rasterizirati. Ako točke trokuta ispunjavaju sve ili neke od navedenih uvjete, trokut se kvalificira za rasterizaciju i dalje se obrađuje.

Nakon prethodnog filtriranja točaka vrši se perspektivno dijeljenje te jedna dodatna provjera vidljivosti – računanje vektora normale trokuta. Vektor normale mora biti okrenut prema gledištu kamere kako bi trokut bio vidljiv, odnosno njegova *z* koordinata mora biti negativna jer se nakon perspektivne transformacije točke prebacuju na pozitivnu stranu *z*-osi koordinatnog sustava.

Ukoliko točke trokuta ispunjavaju sve od navedenih uvjeta, transformiraju se u prostor rastera. Trokut tako postaje spreman za postupak rasterizacije.

Četvrti i posljednji korak obuhvaća rasteriziranje trokuta vidljivih kameri. Rasterizaciju obavlja klasa *ScanConverter*. Odabir fragmenta koji se upisuju u *framebuffer* ovisi o *fragment shader* funkciji definiranoj u *shader* objektu, kao i boja piksela, odnosno fragmenta što se upisuje u raster.

3.3. Podržane platforme

Biblioteka za grafički prikaz implementirana je na *Android*, *Java*, i *HTML5* platformama, stoga se sotfverski renderer može koristiti u mobilnim, *desktop*, i *web* okruženjima.

Na *HTML5* (*HyperText Markup Language 5)* platformi, kontekst je implementiran kao *canvas* element. *Framebuffer* predstavlja niz od 8-bitnih cijelih brojeva bez predznaka. Jedan piksel rastera se tako pohranjuje kao niz od četiri broja koji predstavljaju vrijednosti *RGBA* (*Red Green Blue Alpha*) kanala piksela. Korištenje softverskog renderera na ovoj platformi kreira *JavaScript* datoteku sa skriptom za renderiranje scene

Na *Java* platformi, kontekst je implementiran kao prozor pomoću *Swing* biblioteke što je uključena u *Java Development Kit* (*JDK*) razvojni alat. *Framebuffer* predstavlja niz cijelih brojeva. Jedan piksel rastera se tako pohranjuje kao jedan broj u nizu, unutar kojeg su pohranjene vrijednosti *RGB* (*Red Green Blue*) kanala piksela. Korištenje softverskog renderera na ovoj platformi kreira *Java* izvorni kôd za renderiranje scene i *JAR* (*Java Archive*) datoteku s kojom se renderer može pokrenuti.

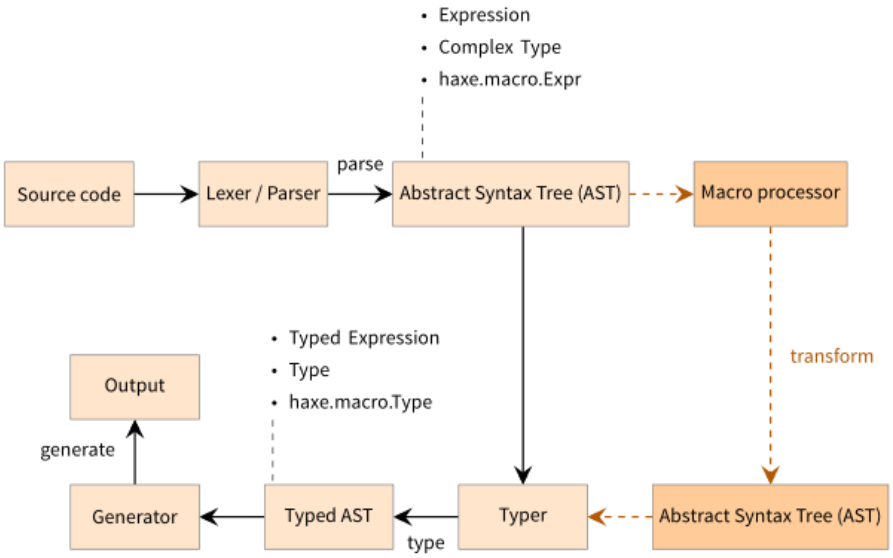
Na *Android* platformi, kontekst je implementiran kao poseban *Activity* i *SurfaceView* pomoću *Android Software Development Kit* (*Android SDK*) razvojnog alata. *Framebuffer* predstavlja objekt *Bitmap* klase iz *Android SDK*. Jedan piksel rastera se tako pohranjuje u *Bitmap* objekt kao cijeli broj, unutar kojeg su pohranjene vrijednosti *ARGB* (*Alpha Red Green Blue*) kanala piksela. Korištenje softverskog renderera na ovoj platformi kreira projekt za *Android Studio* razvojno okruženje unutar kojeg se nalazi izvorni kôd za renderiranje scene. Korisniku je prepušteno da si prema potrebi konfigurira datoteku projekta unutar razvojnog okruženja i samostalno generira aplikacijsku datoteku renderera.

Kontekst se na svakoj platformi pokušava osvježiti 60 puta u sekundi. Zbog različitih formata zapisa boja piksela, sa strane biblioteke za rendereriranje on je standardiziran isključivim korištenjem *RGB* formata.

3.1. Prikaz modela

Zbog specifičnosti platformi, učitavanju modela pristupa se s posebnom pažnjom. Sve platforme nemaju mogućnost izravnog pristupa podatkovnom sustavu, stoga nije moguće generalizirati postupak učitavanja modela kroz oslanjanje na podatkovni sustav.

Međutim, zahvaljujući *macro* kontekstu izvršavanja *Haxe* prevoditelja – koji se odvija pri prevođenju izvornog kôda – moguće je ostvariti pristup podatkovnom sustavu, obraditi modele, i efektivno ih unijeti u izvorni kôd kao definiranu geometriju.

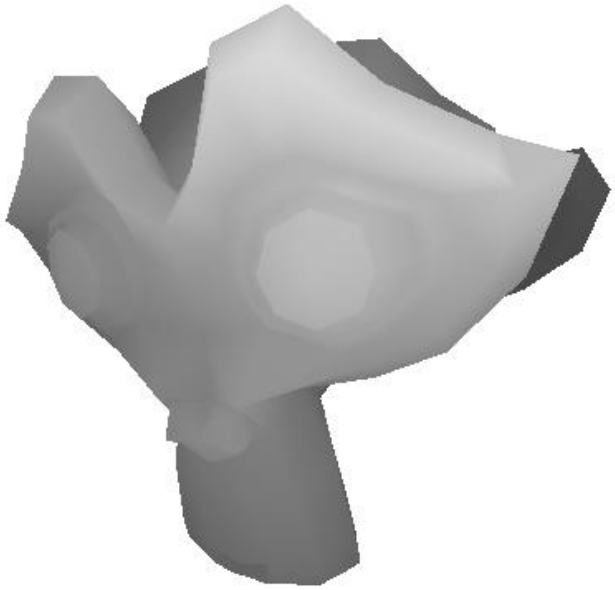


Sl. 3.2. Prikaz proces prevođenja *Haxe* izvornog kôda. Izvor: [13]

Na slici 3.2. prikazan je proces prevođenja *Haxe* izvornog kôda. Na slici se vidi kako se u *macro* kontekstu vrši transformacija generiranog apstraktnog sintaksnog stabla. Transformacije su sintaksne – mogu se promatrati kao dodavanje izvornog kôda gdje ga prije nije bilo, što naravno mijenja rezultat prevođenja *Haxe* izvornog kôda u izvorni kôd drugih jezika [13].

U sklopu biblioteke za renderiranje, učitavanje modela podrazumijeva čitanje podataka o geometriji modela, njihovu obradu, te kreiranje novih statičnih klasa koje bi korisniku biblioteke posluživale obrađene podatke kao varijable statične klase. Prevoditelj na kraju generira izvorni kôd ekvivalentan ručnom unosu svih podataka o geometriji.

Podržan je samo primitivan način učitavanja *ASCII* (*American Standard Code for Information Interchange*) formatiranih *Wavefront* datoteka (ekstenzija: *OBJ)* čiji je standard zapisa opisan u [14]. Model pohranjen u takvu datoteku mora biti trianguliran zato što se implementacija softverskog renderera služi isključivo trokutima kao geometrijskim primitivima. Iz datoteke se čitaju samo osnovni podaci potrebni za konstrukciju trokuta, a to su položaji točaka i njihovi indeksi.



Slika 3.3 Prikaz renderiranja modela zapisanog u *OBJ* formatu

Slikom 3.3. dan je prikaz renderiranja modela zapisanog u *OBJ* formatu. Potrebno je napomenuti kako na *Android* i *Java* platformama postoji ograničenje na veličinu modela što se upisuje u programski kôd. Java izvorni kôd pretvara se u bytecode kojeg izvršava virtualni stroj – *Java Virtual Machine* (*JVM*). *JVM* prema svom dizajnu postavlja granice na veličinu *bytecode*-a funkcija i varijabli neke klase. Konkretno, ne smije premašiti veličinu od 65535 bajta, odnosno skoro 64 kilobajta [15]. Ukoliko se učitava model čiji *bytecode* oblik bude veći ili jednak 64 kB, *JVM* ga neće moći obraditi i javit će grešku.

4. ZAKLJUČAK

Svrha ovog diplomskog rada je predstaviti softversko renderiranje i mogućnost njegove višeplatformske implementacije. U sklopu rada predstavljene su teorijske osnove iza postupka renderiranja, točnije transformacije točaka što definiraju objekte i rasterizaciju objekata, kao i višeplatformska programska implementacija tog postupka u programskom jeziku *Haxe*.

Implementiranje softverskog renderera u *Haxe* programskom jeziku pokazalo je mnogo prednosti i nedostataka. Korištenje jednog programskog jezika standardizira pristup rada na više platformi, što je olakšalo posao implementacije. Modularni dizajn biblioteke omogućuje lako kreiranje rješenja za druge platforme i programske jezike, pod uvjetom da su podržane od strane *Haxe* prevoditelja.

Međutim, razina apstrakcije koju *Haxe* pruža može stvoriti probleme pri testiranju, pronalasku i uklanjanju grešaka na pojedinim platformama, te optimizaciji programskih rješenja. Svaka platforma krije specifičnosti prevođenja generiranog izvornog kôda u izvršni kôd na koje se kroz *Haxe* ne može izravno utjecati. U takvim slučajevima je potrebno razmotriti opciju implementacije u nativnom programskom jeziku platforme, s kojom bi *Haxe* komunicirao kroz definirana komunikacijska sučelja.

Rezultat je rudimentaran softverski renderer skromnih mogućnosti prikaza modela, realiziran u obliku biblioteke za *Haxe* programski jezik, s podrškom za *Android*, *HTML5*, i *Java* platforme.

Prostor za napredak svakako postoji. Općenito, potrebno je evaluirati iskorištavanje višenitnosti u raznim stadijima procesa renderiranja, s ciljem povećanja brzine izvođenja na uređajima i računalima s višenitnim procesorima. S obzirom na renderiranje, moguće je razmotriti uporabu podatkovnih struktura za ubrzanje obrade geometrije, poput grafova scene. S druge strane, preostaje implementirati podršku za svjetla, sjene, i slične elemente što pospješuju vizualni dojam renderirane scene. Mišljenje je autora kako korištenje softverskog renderiranja uslijed hardverskih nedostataka vrijedi podrobnije evaluirati.

LITERATURA

[1] Chromium Blog, Universal rendering with SwiftShader, now open source, (http://blog.chromium.org/2016/06/universal-rendering-with-swiftshader.html), pristup ostvaren 17.09.2016

[2] J. F. Hughes, A. Van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, K. Akeley, Computer Graphics – Principles and Practice, 3rd Edition, Addison-Wesley, Upper Saddle River, NJ, USA, 2014.

[3] Scratchapixel, Where Do I Start? A Very Gentle Introduction to Computer Graphics Programming, (http://www.scratchapixel.com/lessons/3d-basic-rendering/get-started), pristup ostvaren 28.06.2016.

[4] F. Dunn, I. Parberry, 3D Math Primer for Graphics and Game Development, 1st Edition, Wordware Publishing, Inc., Plano, TX, USA, 2002.

[5] Scratchapixel, About the Projection Matrix, the GPU Rendering Pipeline and Clipping, (http://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/projection-matrix-GPU-rendering-pipeline-clipping), pristup ostvaren 28.06.2016.

[6] T. Akenine-Mőller, E. Haines, N. Hoffman, Real-Time Rendering, 3rd Edition, A K Peters, Ltd., Wellesley, MA, USA, 2008.

[7] Scratchapixel, The Rasterization Stage, (http://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/rasterization-stage), pristup ostvaren 28.06.2016.

[8] Haxe Foundation, Introduction to the Haxe Toolkit, (http://haxe.org/documentation/introduction/toolkit-introduction.html), pristup ostvaren 17.09.2016.

[9] Haxe Foundation, Introduction to the Haxe Language, (http://haxe.org/documentation/introduction/language-introduction.html), pristup ostvaren 17.09.2016.

[10] Haxe Foundation, Compiler Features, (http://haxe.org/manual/cr-features.html), pristup ostvaren 17.09.2016.

[11] Haxe Foundation, Compiler Targets, (http://haxe.org/documentation/introduction/compiler-targets.html), pristup ostvaren 17.09.2016.

[12] Haxe Foundation, Introduction to the Haxe Standard Library, (http://haxe.org/documentation/introduction/stdlib-introduction.html), pristup ostvaren 17.09.2016.

[13] Haxe Foundation, Macros, (http://haxe.org/manual/macro.html), pristup ostvaren 28.06.2016.

[14] Paul Bourke, Object Files (.obj), (http://paulbourke.net/dataformats/obj/), pristup ostvaren 28.06.2016.

[15] Oracle, Java Virtual Machine Specification, (http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.11), pristup ostvaren 17.09.2016.

SAŽETAK

Cilj ovog rada je razmotriti softversko renderiranje kao moguće rješenje za problem rastućih razlika u sposobnostima grafičkog kartica dostupnih u osobnim računalima, minijaturnim računalima, pametnim telefonima, i sličnim uređajima. Ovaj rad pruža pregled osnova računalne grafike, s naglaskom na postupak renderiranja, posebice projekciju i rasterizaciju trokuta. U radu je demonstrirana implementacija višeplatformskog softverskog renderera izrađena u *Haxe* skupu alata otvorenog kôda namijenjenom za višeplatformski razvoj, koji prevodi izvorni kôd napisan u *Haxe* programskom jeziku u izvorni kôd podržanih programskih jezika. Rezultat je softverski renderer što podržava učitavanje modela u *Wavefront OBJ* formatu, dostupan za korištenje u okruženjima osobnih računala, pametnih telefona, i internetskih preglednika kroz *Java, Android*, i *HTML5* platforme.

Ključne riječi: višeplatformski razvoj, rasterizacija, softversko renderiranje, računalna grafika, *Haxe*

ABSTRACT

CROSS-PLATFORM SOFTWARE RENDERER

The aim of this paper is to consider software rendering as a possible solution to the growing differences in capabilities of graphics hardware present in personal computers, single-board computers, smartphones, and other devices. The paper provides an overview of computer graphics fundamentals with a focus on the rendering process, particularly projection and triangle rasterization. The paper demonstrates a cross-platform software rendering solution implemented in Haxe, an open source toolkit for cross-platform development which translates source code written in the Haxe programming language to source code of supported programming languages. The result is a software rendering solution with support for model assets stored in Wavefront OBJ format, available for use on desktop, mobile, and web environments through Java, Android, and HTML5 platforms.

Keywords: cross-platform, rasterization, software rendering, computer graphics, Haxe

ŽIVOTOPIS

Domagoj Štrekelj rođen je 18. prosinca 1992. godine, u Osijeku. Osnovno obrazovanje započinje 1999. godine upisom u osječku osnovnu školu Retfala. Po završetku osnovne škole, 2007. godine nastavlja obrazovanje u III. gimnaziji Osijek. 2011. godine upisuje preddiplomski studij računarstva na Elektrotehničkom fakultetu Osijek. Preddiplomski studij završava 2014. godine te upisuje diplomski studij računarstva na istom fakultetu. Istraživanje provedeno tijekom izrade završnog rada na preddiplomskom studiju prezentira na konferenciji *Science in Practice* 2014. godine u Osijeku te objavljuje znanstveni rad u časopisu *IJECES* (*International Journal of Electrical and Computer Engineering Systems*).

PRILOG 1 – TRANSFORMACIJSKA MATRICA ORTOGONALNE PROJEKCIJE

Ortogonalna projekcija je vrsta projekcije s namjerom očuvanja postojećih paralelnih linija. Vidljiv obujam prostora, sa stajališta ortogonalne projekcije, ima oblik kvadra. Objekti, odnosno točke koje se nalaze unutar vidljivog obujma postaju kandidati za rasterizaciju [2].

Pretpostavlja se vidljivi obujam prostora definiran stranicama *right*, *left*, *top*, *bottom*, *near*, i *far* koje čine desnu, lijevu, gornju, donju, bližu, i dalju stranicu kvadra. Provjerava se položaj točke *P* u odnosu na vidljivi obujam. Vidljivi obujam normalizira vrijednosti u raspon [-1, 1] za *x* i *y* koordinate te [0, 1] za *z* koordinatu.

$$left \leq P_X \leq right$$

$$0 \leq P_X - left \leq right - left$$

$$0 \leq \frac{2 \cdot (P_X - left)}{right - left} \leq 2$$

$$-1 \leq \frac{2}{right - left} \cdot P_X - \frac{right + left}{right - left} \leq 1$$

$$P'_X = \frac{2}{right - left} \cdot P_X - \frac{right + left}{right - left} \qquad \text{(P1-1)}$$

Analogno (P1-1) dobiva se izraz za *y*-koordinatu projekcije točke P.

$$P'_Y = \frac{2}{top - bottom} \cdot P_Y - \frac{top + bottom}{top - bottom} \qquad \text{(P1-2)}$$

Izraz za *z* koordinatu projekcije točke dobiva se iz slučajeva kada se točka nalazi na bližoj ili daljoj stranici kvadra.

$$near \leq -P_Z \leq far$$

$$P'_Z = a \cdot P_Z + b$$

$$-P_Z = near \rightarrow P'_Z = 0 \rightarrow a = \frac{b}{near}$$

$$-P_Z = F \rightarrow P'_Z = 1 \rightarrow b = \frac{near}{near - far}$$

$$P'_Z = \frac{1}{near - far} \cdot P_Z + \frac{near}{near - far} \qquad \text{(P1-3)}$$

Uvrštavanjem jednadžbi (P1-1), (P1-2), i (P1-3) u matricu dobiva se transformacijska matrica ortogonalne projekcije (P1-4).

$$\mathbf{M}_{\mathbf{ORTOGONALNO}} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & \frac{1}{near-far} & 0 \\ -\frac{right+left}{right-left} & -\frac{top+bottom}{top-bottom} & \frac{near}{near-far} & 1 \end{bmatrix} \qquad \text{(P1-4)}$$

Iz matrice (P1-4) može se uočiti kako ortogonalna projekcija objedinjuje dvije transformacije – skaliranje i translaciju – kojima se transformirane točke pozicioniraju unutar vidljivog obujma. Perspektivno dijeljenje nije potrebno provoditi jer *w* koordinata ostaje jednaka 1.

PRILOG 2 – PRIMJERI

Na CD-u se – uz izvorni kôd biblioteka softverskog renderera – nalaze mape sa primjerima scena. Za svaku scenu dan je izvorni kôd te rezultat prevođenja za svaku od platforma što ih softverski renderer podržava. Primjeri demonstriraju:

1. Rasterizaciju trokuta bez sjenčanja.
2. Rasterizaciju trokuta sa sjenčanjem.
3. Teksturiranje trokuta.
4. Teksturiranje ravnine.
5. Sjenčanje pomoću *z* koordinate.
6. Rasterizaciju *OBJ* modela.