SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1200

PROGRAMI ZA SJENČANJE

Bruno Mikuš

Zagreb, lipanj 2010.

Sadržaj

Uvod

Cilj ovog rada je proučiti vrste i načine sjenčanja, koje je bitan dio programirljivog grafičkog cjevovoda. Prvo je dan pregled gdje se i kako obavlja sjenčanje vrhova i fragmenata te je opisan predviđeni budući razvoj ovog djela računalne grafike.

Slijedi podjela programa za sjenčanje s obzirom na primjenu te je napravljen kratak opis svih poznatih jezika za pisanje programa za sjenčanje.

Implementacijom jednostavnih programa za sjenčanje vrhova i fragmenata dan je uvod u razvoj programa za sjenčanje kao i u jezik za sjenčanje HLSL. Razvijena je i programska potpora koja pokazuje kako u stvarnim aplikacijama te programe i primijeniti.

U konačnici je dana analiza prednosti i nedostataka razvoja i korištenja programa za sjenčanje te su razmatrani utjecaji različitih parametara na rad i ponašanje sjenčanja.

Uz sjenčanje, na početku samog rada je i kratak pregled matematičkih modela i principa koje koristi grafički procesor odnosno koji se u pozadini odvijaju u programima za sjenčanje.

Pregled korištenih alata i resursa dan je u Dodatku A.

Pregled područja

Postojeći radovi usko vezani uz temu programa za sjenčanje koriste OpenGL aplikacijsko programsko sučelje te GLSL odnosno Cg jezike za razvoj programa za sjenčanje.

U radu [1] dan je opsežniji pregled cjelokupnog područja grafičkog cjevovoda i procesa unutar istog, a naglasak je na programirljivosti grafičkog sklopovlja. Obrađena je sintaksa jezika Cg (HLSL) i GLSL te prikazano na nekoliko programskih primjera korištenje tih jezika. Također je dan pregled mnoštva alata vezanih uz rad na ovom području.

Blizak je rad [2] koji se doduše ne bavi cjevovodom, ali obrađuje sintaksu jezika GLSL i orijentira se na prikaz korištenja programa za sjenčanje kako bi se primijenilo teksturiranje i osvjetljavanje na brz i jednostavan način.

Osim nekoliko znanstvenih radova na Hrvatskom sveučilištu, postoji mnoštvo znanstvenih radova na stranim sveučilištima i još više privatno razvijanih programskih primjera čiji su izvorni kodovi dostupni na Internetu.

1 Matematika računalne grafike

Kako bi postupak računanja koji se odvija prilikom sjenčanja bio jasniji, potrebno je napraviti kratak pregled osnovnih matematičkih operacija korištenih u računalnoj grafici. Riječ je prvenstveno operacijama nad vektorima i matricama, matricama transformacija, interpolacijama koordinata, računanju i operacijama nad bojama te računanju i jednostavnom modelu svjetlosti.

Kako opis matematičkih operacija nije prvenstveni cilj ovog rada, one su ovdje navedene kao podsjetnik.

1.1. Vektorski i matrični račun

Matrice su nizovi brojeva. Definira ih broj redova m i broj stupaca n. Primjer matrice $m \times n$ dimenzija je:

$$M = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix}$$

Matrice je moguće množiti sa konstantom k tako da se svaki element matrice pomnoži s tom konstantom:

$$kM = \begin{bmatrix} kx_{1,1} & \cdots & kx_{1,n} \\ \vdots & \ddots & \vdots \\ kx_{m,1} & \cdots & kx_{m,n} \end{bmatrix}$$

Umnožak dvije matrice moguć je samo kada se dimenzije matrica poklapaju na način da su dimenzije prve matrice $m \times n$, a dimenzije druge matrice $n \times l$. Dimenzije rezultantne matrice su $m \times l$, a elementi matrice računaju se množenjem redova prve matrice sa stupcima druge matrice. Ilustracije radi, primjer s manjim matricama:

$$R = M_1 \times M_2 = \begin{bmatrix} 1 & 5 \\ -2 & 4 \end{bmatrix} \times \begin{bmatrix} 0 & 3 \\ 2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 0 + 5 \cdot 2 & 1 \cdot 3 + 5 \cdot -1 \\ -2 \cdot 0 + 4 \cdot 2 & -2 \cdot 3 + 4 \cdot -1 \end{bmatrix} = \begin{bmatrix} 10 & -2 \\ 8 & -10 \end{bmatrix}$$

Vektor je matrica sa jednim redom ili jednim stupcem. U računalnoj grafici često se radi o vektorima sa tri ili četiri dimenzije.

$\vec{v} = [x \quad y \quad z]$ odnosno $\vec{v} = [x \quad y \quad z \quad w]$

Duljina vektora je definirana kao:

$$\|\vec{v}\| = \sqrt{x^2 + x^2 + x^2}$$

Vektor je moguće skalirati kao i običnu matricu, množenjem s konstantom, a moguće ih je i međusobno pomnožiti na dva načina, skalarno ili vektorski.

Skalarni umnožak, geometrijski predstavlja površinu koju dva vektora razapinju:

$$\vec{v}_1 \cdot \vec{v}_2 = \|\vec{v}_1\|\|\vec{v}_2\| \cos \Theta$$

gdje je Θ kut između vektora. Jednostavniji način računanja skalarnog umnoška je:

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

Vektorski produkt, predstavlja vektor okomit na $\vec{v}_1$ i $\vec{v}_2$ usmjeren prema pravilu desne ruke:

$$\vec{v}_1 \times \vec{v}_2 = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix} = (y_1 z_2 - z_1 y_2)\vec{i} + (z_1 x_2 - x_1 z_2)\vec{j} + (x_1 y_2 - y_1 x_2)\vec{k}$$

Normalizirani vektor je onaj čija je duljina 1, a dobiva se skaliranjem s recipročnom duljinom:

$$\hat{v} = \frac{\vec{v}}{\|\vec{v}\|}$$

Kosinus kuta između dva normalizirana vektora je jednostavno izračunati preko skalarnog umnoška.

1.2. Transformacije objekata, pogleda i projekcije

Objekti su u računalnoj grafici predstavljeni skupom vrhova koji određuju vanjsku ljusku objekta. Vrh se opisuje vektorom sa 4 komponente, x, y i z koordinatom te homogenom komponentom:

$$vrh = [x \quad y \quad z \quad h]$$

Transformacijom svake od točaka koje čine jedan objekt izvršava se transformacija čitavog objekta u prostoru. Transformacija se izvršava jednostavno, množenjem vektora točke s matricom transformacije. Korisno je znati tri transformacije: translacija, rotacija i skaliranje.

Translacija po pojedinačnim osima:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{bmatrix}$$

Rotacija oko pojedine osi u smjeru suprotnom smjeru kazaljke na satu:

$$R_{OsX} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{OsY} = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{OsZ} = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Skaliranje po osima:

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformacija pogleda ima kao cilj pretvoriti koordinate objekata iz sustava scene u sustav oka, kako bi svi objekti bili definirani u odnosu na mjesto iz kojeg gledamo na svijet, tj. kako bi očište bilo u ishodištu koordinatnog sustava. Dodatno, transformacijom pogleda poravnava se os aplikata (z-os) s vektorom iz očišta prema gledištu, kako bi bilo jednostavnije određivati koji su objekti ispred, a koji iza.

Perspektivna projekcija služi kako bi se objekti prikazali kako ih ljudsko oko u stvarnosti percipira. Riječ je o skaliranju x i y koordinata točke faktorom $\frac{H}{z}$ gdje je H udaljenost gledišta od očišta.

Potpuni postupak s računom moguće je pronaći u službenim materijalima predmeta Interaktivna računalna grafika [3][4].

1.3. Interpolacija

Interpolacija u prostoru je parametarsko određivanje koordinata na temelju definiranih parametara nad nekim točkama. Kako bi bio jasniji koncept, opisane su linearna i bilinearna interpolacija.

Linearna interpolacija je definirana nad dvije točke. Parametrom t određujemo omjer u kojem će točke V_0 i V_1 imati utjecaja na rezultantnu točku. Jednadžba ima sljedeći izgled:

$$V(t) = (1-t)V_0 + tV_1$$

Bilinearna interpolacija je parametarska jednadžba definirana nad četiri točke. Parametrima u i v određuje se udio pojedine točke prilikom izračuna konačne točke koja se interpolira. Jednadžba ima sljedeći izgled:

$$V(u,v) = (1-u)(1-v)V_{0,0} + u(1-v)V_{1,0} + (1-u)vV_{0,1} + uvV_{1,1}$$

1.4. Računanje boja i svjetlosti

Boja se u računalnoj grafici najčešće rastavlja na tri aditivne komponente, a to su crvena, zelena i plava. Predstavljene su svaka sa po 8 bitova, tj. ukupno sa 256 mogućih razina za svaku boju, ali se zbog lakšeg računa, komponente boje zapisuju u decimalnom obliku u rasponu $[1..0]$. U obzir se uzima i α komponenta koja predstavlja koeficijent prozirnosti boje. Ove četiri komponente čine vektor:

$$boja = [red \quad green \quad blue \quad alpha]$$

Ovako zapisanu boju moguće je transformirati pomoću matrica, što omogućuje ostvarivanje vrlo jednostavnih efekata.

Svjetlost je fizikalno iznimno komplicirana za opisati, pošto sadrži mnogo komponenti i parametara. Grafika aproksimira svjetlost na nekoliko komponenti, a za potrebe rada navedene su i korištene ambijentalna i difuzna komponenta.

Ambijentalna svjetlost nastaje općom refleksijom u nekom prostoru te se pojednostavljeno uzima da djeluje na sve površine jednako:

$$I = I_a k_a$$

I_a je intenzitet boja ambijentalnog svjetla, a k_a koeficijent utjecaja ambijentalnog svjetla.

Difuzna svjetlost nastaje utjecajem točkastog izvora svjetlosti:

$$I = I_d k_d (\hat{L} \cdot \hat{N})$$

I_d je intenzitet boja ambijentalnog svjetla, k_d je koeficijent utjecaja ambijentalnog svjetla, $\hat{L}$ je normalizirani vektor od točke koja je osvijetljena do izvora, a $\hat{N}$ je normalizirani vektor normale u promatranoj točci.

Ukupna svjetlost u ovom modelu je:

$$I_u = I_a k_a + I_d k_d (\hat{L} \cdot \hat{N})$$

2 Sjenčanje

Računalna grafika, od svojih prvih dana, nastoji što vjernije i što kvalitetnije imitirati i prikazati vizualnu stvarnost percipiranu okom. Kroz godine ti su se postupci i tehnike konstantno proširivali i poboljšavali nadograđujući se na postojeće sustave čineći u konačnici grafički cjevovod.

Jačanje grafičkih procesnih jedinica te povećanje dostupne grafičke memorije glavni su indikatori razvoja grafičkih kartica. U taj razvoj uključen je i razvoj kompleksnog sustava paralelnog rada te specijaliziranih jedinica - procesora. Raslojavanjem cjevovoda postalo je očigledno kako se neki poslovi mogu međusobno neovisno izvršavati – upravo na specijaliziranim jedinicama.

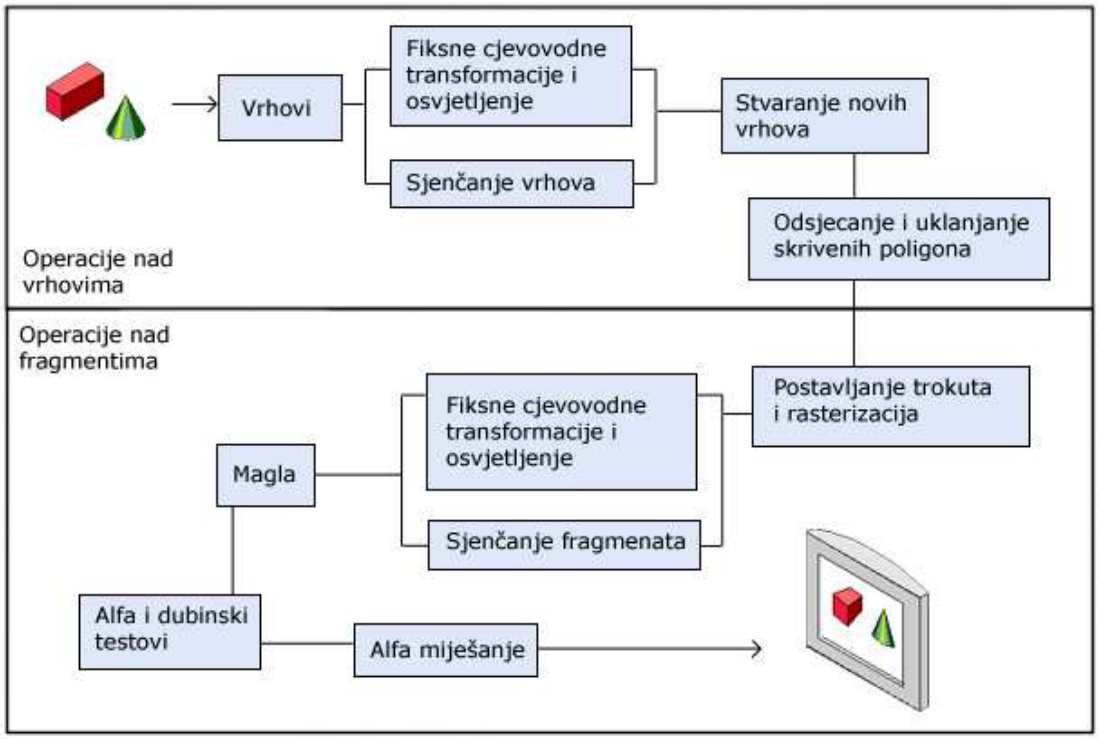
Na ovaj način ne samo da je ubrzan rad računalne grafike, već je ubrzan i njen razvoj, jer se programeri i dizajneri ne moraju toliko puno zamarati s nužnim dijelovima grafičkih programa, čija je implementacija postala jednostavnija.

U nastavku poglavlja opisano je raslojavanje cjevovoda, grupiranje poslova na grafičkoj kartici te pojedinačni opis jedinica za sjenčanje.

2.1. Slojevi grafičkog cjevovoda

Dolaskom 3D grafike pojavilo se mnoštvo novih zadataka koje je grafička kartica trebala obavljati u određenom redoslijedu. Ti poslovi danas predstavljaju nužne komponente u programiranju računalne grafike, a zajedno čine grafički cjevovod, sastavljen od dijelova prikazanih na slici 2.1.

Riječ je dakle o sljedećim zadatcima: transformacija objekata (točaka i normala) u prostor pogleda, računanje koordinata tekstura, stvaranje novih točaka u prostoru, projekcija, odsijecanje, postavljanje trokuta i rasterizacija, uklanjanje nevidljivih trokuta, teksturiranje, računanje boja i osvjetljenja.



Slika 2.1 Grafički cjevovod

Kao što je ranije rečeno, pojedini su poslovi grupirani unutar cjevovoda i njihovo izvođenje prepušteno je elementima paralelnog sustava, zasebnim procesnim jedinicama pa tako na modernim grafičkim karticama razlikujemo procesne jedinice za:

- sjenčanje vrhova
- sjenčanje geometrije
- sjenčanje fragmenata
- preslikavanje teksture
- generiranje izlaza – cjevovod rasterskih operacija

Svaka od tih jedinica zadužena je za obavljanje nekih od poslova, a one koje su od posebnog interesa su dakako jedinice za sjenčanje.

Sjenčanje je nastalo kao model sjenčanja 1.0 kroz DirectX 8.0 aplikacijsko programsko sučelje i tada je uvedeno sjenčanje vrhova i fragmenata. Sa inačicom DirectX 9.0 pojavio se model sjenčanja 2.0 koji je kasnije podržao još dvije inačice sjenčanja fragmenata (2.0a i 2.0b) te još jednu inačicu sjenčanja vrhova (2.0a). Model sjenčanja 3.0 je uveden sa DirectX 9.0c sučeljem. Značajna promjena došla je sa DirectX 10.0 inačicom, a to je model sjenčanja 4.0, koji je proširen sa sjenčanjem geometrije. Ovaj model je osim sa DirectX sučeljem podržan i u OpenGL 3.2 sustavu.

2.2. Sjenčanje vrhova

Kao što je naznačeno na slici 2.1, sjenčanje vrhova nalazi se na samom početku cjevovoda. Svaki vrh svakog objekta prolazi kroz ovu jedinicu. Ulazni podatci su prostorni podatci ulazne točke poput pozicije, normale, tangente i koordinata teksture.

Poslovi koji se obavljaju prilikom sjenčanja vrhova su:

- projekcijska transformacija - ovaj posao je jedini obavezan za ispravan rad cjevovoda
- promjena pozicije točke
- računanje i promjena koordinata tekstura
- računanje vektora prema svjetlu i prema oku

Najbitniji izlazni podatak je pozicija na ekranu koja nastaje kao rezultat projekcije i vrijednost dubine točke koja se koristi kasnije u Z-spremniku. Ostali izlazni podaci uključuju normalu, vektore smjera pogleda i svjetla , konačne koordinate tekstura i slično.

2.3. Sjenčanje geometrije

Nakon sjenčanja vrhova, jedinica za sjenčanje geometrije može generirati nove grafičke primitive, kao što su točka, linija i trokut. Ulazni podatci ove jedinice su primitivi obrađeni u sjenčanju vrhova i podatci o susjedstvu za taj primitiv.

Poslovi koji se obavljaju prilikom sjenčanja geometrije su:

- generiranje novih točaka
- povećanje kompleksnosti geometrije

Izlazni podatci su novi primitivi, a prosljeđuju se na rasterizaciju.

2.4. Sjenčanje fragmenata

Sjenčanje fragmenata obavlja se prije konačnog crtanja slike na ekran. Nakon sjenčanja vrhova dolazi do rasterizacije objekata i nastaju fragmenti kojima je taj objekt predstavljen na ekranu. Svaki od izračunatih fragmenata mora proći kroz jedinicu za sjenčanje fragmenata kako bi se odredila njegova konačna boja, prozirna komponenta i dubina. Ulazni podatci su oni potrebni za računanje boje, a to uključuje koordinate teksture, prethodno interpolirana boja interpolirana, vektori prema svjetlu i prema oku, normala i slično.

Poslovi koji se obavljaju prilikom sjenčanja fragmenata su:

- računanje boje u nekom obliku – minimalni mogući posao je računanje uvijek iste boje za svaki fragment
- dohvat boje s teksture za dani fragment
- računanje osvjetljenja
- kompleksniji efekti poput preslikavanja izbočina, računanja sjena, uporaba prozirnosti i slično
- kontrola i promjena dubine fragmenta

Izlazni podatak je uvijek boja, a moguće je mijenjati i dubinu pojedinog fragmenta. Za pojedinu točku na ekranu moguće je imati više objekata koji bi mogli određivati boju te točke. Sjenčanje fragmenata računa boju za svaki objekt, a na temelju podataka o dubini i prozirnosti, određuje se konačna boja i ona se prosljeđuje dalje na izlaznu jedinicu.

2.5. Budućnost

Tri jedinice za sjenčanje (vrhova, geometrije i fragmenata) dio su modela sjenčanja 4.0 i čine unificirani model sjenčanja, a posebne su po tome što koriste isti set instrukcija i imaju slične mogućnosti.

Na modernim grafičkim karticama koriste iste jedinice za sva tri sjenčanja, a postoji ih čak nekoliko stotina. Uz velik broj ostalih jedinica, visoke radne taktove jezgre i memorije te široke sabirnice, najnovije grafičke kartice su sposobne računati bilijune operacija sa posmačnim zarezom u sekundi.

Osim povećanja broja jedinica, radnih taktova i općenito nekih fizikalnih svojstava, za očekivati je da će razvoj jedinica i jezika za sjenčanje malo usporiti. Uzevši u obzir unificiranost sustava, ovo područje djeluje poprilično dorađeno te djeluje kao da prostora za razvoj ima samo kod jedinica za geometrijsko sjenčanje.

Ali ako rast nastavi prema dobro poznatom Mooreovom zakonu, u bliskoj bi budućnosti mogli očekivati grafiku najmodernijih filmskih animacija izvođenu u realnom vremenu na našim kućnim računalima i to upravo zahvaljujući mnogobrojnim jedinicama za sjenčanje.

3 Jezici za sjenčanje

Smisao nastanka jedinica za sjenčanje i odvajanje pojedinih postupaka u grafičkom cjevovodu je upravo bilo postizanje veće kontrole nad onime što se događa prije konačnog prikaza slike. A veća kontrola znači pisanje vlastitih programa kako bi se ili postigla već postojeća funkcionalnost ili još bitnije, kako bi se već postojeća funkcionalnost nadmašila.

Potrebe računalne grafike za sjenčanjem kretale su se u dva smjera:

- Iscrtavanje posrednim režimom – slike iznimno visoke kvalitete, gdje vrijeme računanja pojedine slike nije ograničavajući faktor
- Iscrtavanje u stvarnom vremenu – slike se računaju puno puta u sekundi, uz ostvarenu umjerena pa čak i visoku kvalitetu

Prvi smjer je bio zahtjevan od strane filmske industrije koja nas iz godine u godinu zapanjuje vizualnom kvalitetom i sve većim fotorealizmom mnogih filmova.

Drugi smjer je bio zahtjevan od strane proizvođača igara kojima je cilj, između ostalog, impresionirati igrača vizualnim dostignućima igre i povećati zadovoljstvo igranja.

3.1. Iscrtavanje posrednim režimom

Jezici razvijeni za iscrtavanje posrednim režimom teže računanju i prikazu fotorealističnih slika i scena. Kako vrijeme nije faktor ovim jezicima, oni nisu ograničeni na procesnu moć grafičke kartice, već se programi pišu s ciljem izvođenja na centralnoj procesnoj jedinici po mogućnosti više računala odjednom. Ovakav način iscrtavanja pripada programski temeljenom iscrtavanju upravo zato što koristi CPU, a ne dodatno grafičko sklopovlje.

Najpoznatiji jezik iz ove domene je RenderMan, a osim njega postoje i jezici Houdini VEX i Gelato koji se temelje na njemu, tako da o njima neće biti posebno riječi u ovom radu.

3.1.1. RenderMan

Jezik za sjenčanje RenderMan je jedan od prvih jezika za sjenčanje razvijen od strane Pixar Animation Studios u sklopu razvoja RenderMan programskog sučelja (RISpec). Sintaksa jezika je slična sintaksi jezika C, a podržava tipove podataka potrebne za rad s grafikom.

Postoji 6 različitih vrsta sjenčanja definiranih ovim jezikom: premještanje, deformacija, volumen, izvori svjetlosti, površine, crtanje.

U filmu Toy Story (Pixar, 1995.) na slici 3.1, je prvi puta u komercijalnoj filmskoj industriji korišten RenderMan za izradu cijelog djela.

Kako je programsko sučelje ovog jezika otvoreno, moguće je korištenje RenderMan-a i u vlastitim projektima, a dobar primjer za to je film Elephants Dream [5] koji je objavljen kao prvi otvoreni film i dostupan je svakome.

Slika 3.1 Scena iz filma Toy Story

3.2. Iscrtavanje u stvarnom vremenu

Jezici za iscrtavanje u stvarnom vremenu nastali su kao što je već rečeno kako bi se postigla veća kontrola nad radom grafičkog cjevovoda u grafičkim karticama. Ovi jezici omogućuju korištenje kompleksnijih tipova podataka i specifičnih funkcija grafičke kartice.

3.1.2. ARB – GPU asembler

Razvojem prvih grafičkih kartica razvijali su se mnogi različiti asemblerski jezici za grafički procesor. S vremenom je došlo do standardizacije tih jezika od strane OpenGL odbora za pregled arhitekture (OpenGL ARB) i tvrtke Nvidia. Godine 2002. razvijen je standard za asemblerski jezik za sjenčanje vrhova i fragmenata.

Varijable ARB asemblera su vektori sastavljeni od 4 broja s posmačnim zarezom. Postoji podrška za matrice stanja poput matrice pogleda i projekcije.

Instrukcije su slične onima asemblera za CPU poput zbrajanja (ADD) i množenja (MUL), ali ima mnogo onih karakterističnih za matematiku računalne grafike poput skalarnog umnoška (DP3 i DP4) ili vektorskog umnoška (XPD). Zanimljivo je to što ARB nema naredbe skoka i grananja, što ga čini malo težim za uporabu.

3.1.3. GLSL

OpenGL Shading Language (OpenGL jezik za sjenčanje) je razvio ranije spomenuti OpenGL ARB odbor, a uključen je u OpenGL od inačice 2.0. Kako je OpenGL otvoreni standard, glavna prednost GLSL jezika je mogućnost korištenja na više operacijskih sustava. Proizvođači kartica uključuju prevoditelj za ovaj jezik u svoje upravljačke programe kako bi strojni kod uvijek bio optimiran za korištenu grafičku karticu.

Sintaksa jezika slična je sintaksi jezika C odnosno C++ pa su tako podržane petlje i grananja (što nije bilo moguće sa ARB-om), pisanje vlastitih funkcija, preopterećivanje funkcija i još mnogo toga.

Programi za sjenčanje definiraju se glavnom funkcijom (main), moguće im je proslijediti ulazne podatke iz toka podataka grafičkog cjevovoda, a moguće je prenijeti i vanjske parametre iz OpenGL okruženja.

3.1.4. Cg

C for Graphics (C za grafiku), kao što samo ime govori je programski jezik sličan programskom jeziku C, kojim se pišu grafički programi za sjenčanje. Razvila ga je tvrtka Nvidia u suradnji s Microsoftom, te je stoga ovaj jezik iznimno sličan Microsoftovom HLSL-u. Programe za sjenčanje pisane u Cg-u moguće je koristiti iz DirectX i OpenGL okruženja, a prevoditi se mogu pri izvođenju, kako bi bili optimirani za korištenu grafičku karticu.

Sintaksa je slična jeziku C, podržane su naredbe za kontrolu toka programa, uz standardne tipove podataka postoje i vektori i matrice te operatori za rad s tim strukturama, a definirane su i mnoge funkcije potrebne u programima za sjenčanje poput uzorkovanja tekstura i slično.

Kao i GLSL, moguće je upravljati i definirati podatke grafičkog cjevovoda koji će biti ulaz i izlaz iz programa za sjenčanje te prenijeti parametre iz DirectX tj. OpenGL okruženja.

3.1.5. HLSL

High Level Shading Language (jezik sjenčanja visoke razine) je kao što je rečeno ranije, razvio Microsoft u suradnji sa Nvidiom dok je razvijala Cg, tako da su ta dva jezika iznimno slična. HLSL programe za sjenčanje moguće je koristiti jedino iz DirectX okruženja koje pruža Windows platforma i igrača konzola Xbox.

O sintaksi, toku podataka i prenošenju parametara će biti riječi detaljno u sljedećem poglavlju, pošto je ovaj jezik odabran za izradu programskog primjera.

4 Razvoj programa za sjenčanje

Razvoj programa za sjenčanje je zahvaljujući odličnim dostupnim alatima poprilično jednostavan posao. Jezici za sjenčanje (pogotovo oni viši) uz poznavanje principa rada grafičkog cjevovoda i matematike računalne grafike omogućavaju odvajanje posla dizajniranja grafike i grafičke aplikacije.

Ovo poglavlje prikazuje kako napraviti program za sjenčanje, bez potrebe za programiranjem dodatne programske potpore u obliku grafičkog programa koji komunicira s grafičkim sučeljem.

Biti će prikazano i korištenje više prolaza programa za sjenčanje. To je sposobnost grafičkog cjevovoda da koristi različite programe u različitom vremenu za različite objekte. Nužno je koristiti više prolaza kako bi se ostvario neki složeniji efekt. Svaki prolaz može imati definiran program za sjenčanje vrhova i fragmenata koji će se primjenjivati nad objektima koji im se proslijede.

4.1. Korišteni jezik - HLSL

Razlog odabira jezika HLSL za izradu programa za sjenčanje, a i DirectX sučelja za razvoj programske potpore je želja autora rada da proširi spektar znanja računalne grafike u smjeru suprotnom od OpenGL sučelja s kojim se već susreo.

U ovom djelu biti će opisana sintaksa jezika HLSL, odnosno spomenuti oni najbitniji dijelovi. Razvijen je na temelju jezika C, te s njim dijeli puno ključnih riječi, tipova podataka i operatora, ali je naravno proširen za potrebe opisivanja i rada s grafičkim cjevovodom.

4.1.1. Tipovi podataka

HLSL podržava mnoge tipove podataka koji postoje u jeziku C, ali i uvodi neka proširenja.

Skalarni tipovi podataka

Koriste se tipovi podataka nužni za rad s brojevima, a navedeni su u tablici 4.1. Tipovi poput znaka i pokazivača nisu potrebni u programima za sjenčanje. U slučaju da grafička kartica ne podržava ove tipove, prevodioc će zamijeniti nepodržane tipove s tipom *float*, što može izazvati neočekivano ponašanje.

Tablica 4.1 Skalarni tipovi podataka

Tip	Opis
bool	logički tip
int	32-bitni predznačeni cijeli broj
half	16-bitni broj sa posmačnim zarezom
float	32-bitni broj sa posmačnim zarezom
double	64-bitni broj sa posmačnim zarezom

Matrični tipovi podataka

Ovi tipovi podataka su semantičko maskiranje niza elemenata s ciljem bolje apstrakcije pisanih programa. Koriste se vektori i matrice, a predstavljaju jednodimenzionalni, odnosno dvodimenzionalni niz brojeva. U tablici 4.2 nalazi se popis ključnih riječi za deklaracije vektorskih i matričnih tipova.

Tablica 4.2 Matrični tipovi podataka

Tip	Opis
vector	vektor sastavljen od 4 *float* broja
vector< *tip*, *veličina*>	vektor duljine *veličina*, sa brojevima tipa *tip*
*float*N	vektor sastavljen od N *float* brojeva, N je između 2 i 4, na isti je način moguće deklarirati i vektore tipa *int* i *half*
matrix	matrica od 4 reda i 4 stupca *float* brojeva
matrix < *tip*, *redova*, *stupaca*>	matrica dimenzija *redova* x *stupaca*, sa brojevima tipa *tip*
*float*RxS	matrica dimenzija R x S *float* broja, R i S su između 2 i 4, na isti je način moguće deklarirati i vektore tipa *int* i *half*

Pristup matričnim podatcima obavlja se kao i pristup običnom nizu, indeksiranjem u uglatim zagradama. Elementima vektora moguće je pristupati kao da se pristupa elementima strukture, npr. *vektor[1]* se može napisati i kao *vektor.y* ili *vektor.g*. Oznake korištene sa točkom su *xyzw*, odnosno *rgba*, a nastale su iz oznaka vektora položaja i boje. Moguće je istovremeno označiti više elemenata na taj način, tako će npr *vektor.xzy* predstavljati vektor od 3 elementa u kojima se nalaze redom vrijednosti *vektor.x*, *vektor.z* i *vektor.y*.

Strukture

Strukture se deklariraju ključnom riječi *struct*. Oblik deklaracije je:

`struct` *Naziv* { članovi strukture `};`

Za jednostavniji i jasniji prikaz ulaznih i izlaznih podataka programa za sjenčanje koriste se strukture.

Uzorci

Uzorcima se predstavljaju teksturni objekti, a označuju se ključnom riječi *sample.* Postoje dodatne ključne riječi za različite tipove tekstura poput 1D, 2D, 3D ili kocke, ali ih nije potrebno koristiti.

4.1.2. Programske strukture

U programske strukture pripadaju varijable, deklaracije, naredbe, izrazi i izjave.

Varijable

Varijable se mogu deklarirati kao i u jeziku C, a deklaracija je obavezna ako se varijabla namjerava koristiti. Mogu se koristiti prefiksi *static*, *uniform*, *volatile* i *const*, a značenjem su istovjetni kao i u drugim jezicima.

Sintaksno proširenje dolazi u obliku navođenja semantičkih oznaka. Kako bi se olakšao i bolje naznačio izvor pojedinih podataka, neke je varijable potrebno označiti semantičkim dodatkom, koji govori što neka varijabla predstavlja. U isječku koda 4.1 u deklaraciji glavnog programa za sjenčanje vrhova definirali smo ulaznu varijablu *texCoo* tipa *float2*, a semantike *TEXCOORD0*. Rezultat funkcije je tipa *float4*, a semantike *COLOR0*. Ovaj program kao ulazni podatak prima koordinate teksture, dva decimalna broja, a vraća kao rezultat boju sa 4 komponente izračunatu s ulazne teksture *tekstura*.

```
sampler tekstura;

float4 ps_main( float2 texCoo : TEXCOORD0 ) : COLOR0
{
        return tex2D( tekstura, texCoo );
}
```

Isječak koda 4.1 Program za sjenčanje fragmenta koji primjenjuje teksturu.

Semantičke oznake biti će opisane kroz primjere u nastavku.

Izjave

U izjave pripadaju operacije i naredbe za kontrolu toka programa, a dane su u tablici 4.3.

Tablica 4.3 Izjave za kontrolu toka programa

Izjava	Niz naredbi
blok	{ *niz_izjava* }
naredba	*izraz*;
povrat vrijednosti	`return` *izraz*;
if-else	`if` (*uvjet*) *blok* `else` *blok*
for petlja	`for`(*izraz/deklaracija*; *izraz*; *izraz*) *blok*
while petlja	`while` (*uvjet*) *blok*

Izrazi

Izrazi su gradivne jedinice programa koje obavljaju nekakav posao, a čine ih matematičke, logičke i funkcijske operacije. Lista nekih operatora korištenih u HLSLu je u tablici 4.4.

Tablica 4.4 Operatori korišteni u izrazima

Tip operatora u izrazu	Operator
matematički operatori	+, -, *, /, %, ++, --
operatori usporedbe	<, >, <=, >=, ==, !=
logički operatori	!, &&, ||
operatori pridruživanja	=, +=, -=, *=, /=, %=
dohvat elementa	[], . (kad vektor ima do 4 elementa)

Ugrađene funkcije

Postoji mnoštvo ugrađenih funkcija koje ubrzavaju razvoj programa za sjenčanje jer pružaju gotova rješenja za probleme poput umnožaka vektora, uzorkovanja tekstura, interpolacije i slično. U tablici 4.5 je popis češće korištenih funkcija.

Tablica 4.5 Neke od ugrađenih funkcija jezika HLSL

Funkcija	Opis povratne vrijednosti
`abs(x)`	apsolutna vrijednost brojeva u vektoru x
`sin, cos, tan, asin, acos, atan (x)`	rezultat trigonometrijske funkcije
`ceil, floor (x)`	rezultat je prvi viši tj. prvi niži cijeli broj u odnosu na x
`clamp(x, min, max)`	vrijednosti vektora x se postavljaju na min ili max ako prelaze te granice
`cross(a, b)`	vektorski umnožak vektora a i b
`dot(a, b)`	skalarni umnožak vektora a i b
`exp, exp2 (x)`	vraća vektor elemenata potencija broja e , tj. broja 2 na vrijednost elementa ulaznog vektora x
`len, length (x)`	duljina vektora x
`lerp(s, a, b)`	interpolacija između vektora a i b koristeći omjer s, rezultat je računat prema formuli $(1-s)\vec{a}+s\vec{b}$
`log, log2, log10 (x)`	rezultat operacije logaritma u bazi e, 2 ili 10 nad vektorom x
`mul(a, b)`	matrični umnožak matrica a i b, dimenzije se moraju podudarati
`normalize(x)`	normalizirani vektor x
`pow(x, a)`	vektor x čije su komponente potencirane na a
`sqrt(x)`	drugi korijen od x
`tex1D, tex2D, tex3D (s, x)`	sa ulazne teksture s uzima boju određenu vektorom x, vektor x mora odgovarati broju dimenzija teksture

4.1.3. Prevođenje

Gotovi kod programa za sjenčanje se u DirectX-u prevodi prilikom izvođenja programa. Na taj način će se dobiti kod optimiran za grafičku karticu koja postoji na računalu. U sljedećem poglavlju biti će pokazano s kojim se to naredbama radi.

Problem ovog pristupa je naravno u potrebi korištenja izvornog koda programa za sjenčanje svaki put, ali taj problem ne ulazi u opseg ovog rada.

Prevodi se u asemblerski, tj. strojni kod postojeće grafičke kartice pa je za očekivati da će izlazni kod biti drugačiji od platforme do platforme. Razlike u izlaznom kodu koji radi isti posao moguće je primijetiti i korištenjem različitih naredbi. U isječku koda 4.2 prikazani su programi koji rade isti posao, a daju različiti izlazni asemblerski kod. Oba programa za sjenčanje fragmenata kao izlaz daju crno-bijelu boju na temelju boje s teksture.

Izvorni kod	Prevedeni kod

`sampler tekstura;`
 	
`float4 ps_main(float2 texCoord:TEXCOORD0)`
`  : COLOR0`
`{`
`  float4 boja=tex2D(tekstura,texCoord);`
 	
`  float intenzitet= 0.299 * boja.r +`
`  0.587 * boja.g +`
`  0.184 * boja.b;`
 	
`  return float4(intenzitet.xxx,boja.a);`
`}`
	`ps_2_0`
	`def c0, 0.5870, 0.29899, 0.184, 0`
	`dcl t0.xy`
	`dcl_2d s0`
	`texld r0, t0, s0`
	`mul r1.w, r0.y, c0.x`
	`mad r1.x, r0.x, c0.y, r1.w`
	`mad r0.xyz, r0.z, c0.z, r1.x`
	`mov oC0, r0`
	
	`5 instrukcijskih mjesta`

`sampler tekstura;`
 	
`float4 ps_main(float2 texCoord:TEXCOORD0)`
`  : COLOR0`
`{`
`  float4 boja=tex2D(tekstura,texCoord);`
 	
`  float intenzitet=dot(`
`  boja,`
`  float4(0.299, 0.587, 0.184, 0) );`
 	
`  return float4(intenzitet.xxx,boja.a);`
`}`
	`ps_2_0`
	`def c0, 0.29899, 0.5870, 0.184, 0`
	`dcl t0.xy`
	`dcl_2d s0`
	`texld r0, t0, s0`
	`dp3 r0.xyz, r0, c0`
	`mov oC0, r0`
	
	`3 instrukcijska mjesta`

Isječak koda 4.2. Primjer prevođenja programa za sjenčanje fragmenata

Razlika u broju instrukcija nastaje zato što prvi primjer množi zasebno komponente vektora boje s koeficijentima za crno-bijelo. Drugi primjer koristi skalarni umnožak dva vektora, što je podržano sklopovski pa se štedi na dvije operacije množenja.

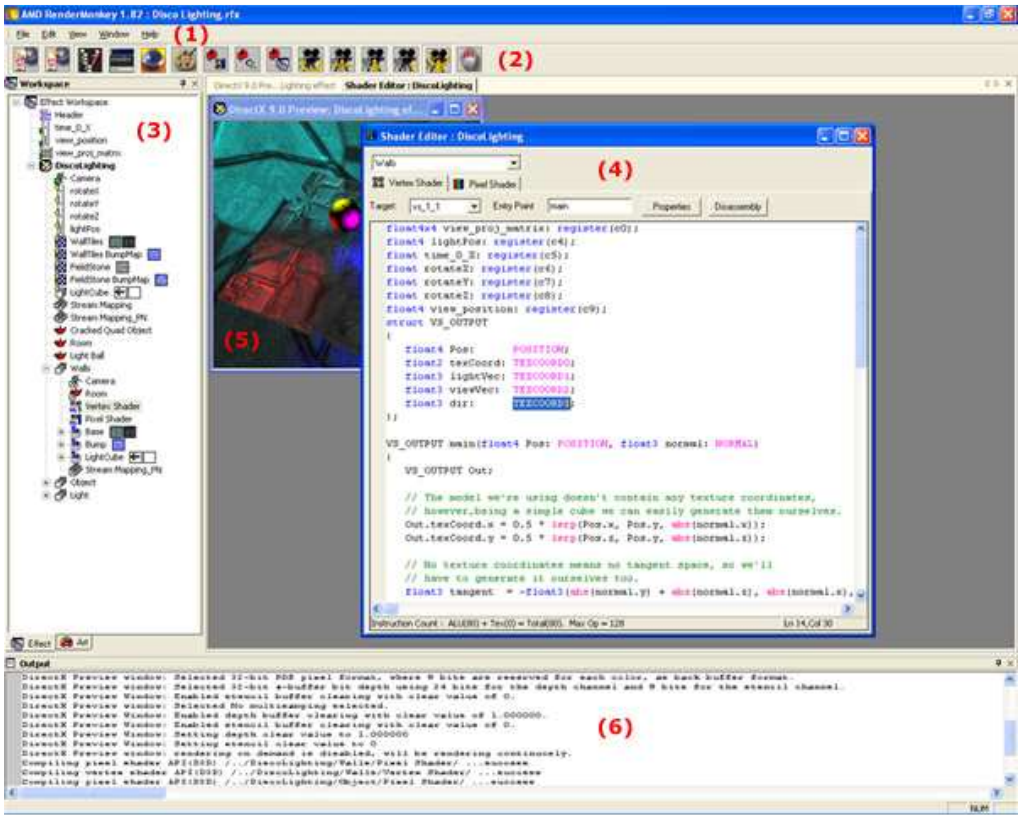
4.2. Alati

Dizajneri u ovom području bez pravih alata imaju problem velike zavisnosti od aplikacija koje koriste sučelja poput OpenGLa ili DirectXa, jer tek pokretanjem grafičkih aplikacija mogu vidjeti plodove svog rada na programima za sjenčanje. To znači da ili sami moraju naučiti koristiti neki viši programski jezik i jedno od sučelja ili moraju čekati da programeri naprave svoj posao, prije nego što mogu početi s dizajniranjem efekata.

Srećom, razvijeni su alati koji olakšavaju i ubrzavaju postavljanje scene nad kojom se mogu izrađivati efekti, a jedan od njih je RenderMonkey, koji je korišten i prilikom izrade ovog rada.

RenderMonkey

Izgled sučelja ovog alata prikazan je na slici 4.1. Sučelje je sastavljeno od glavnih izbornika (1), kontrolnih gumba (2), radnog prostora u kojem se izrađuje efekt (3), prikaza izvornog koda (4), prikaza scene (5) i tekstualnog izlaza prevodioca (6).



Slika 4.1 Izgled sučelja programa Render Monkey

Na službenim stranicama [6] je moguće preuzeti program i pripadnu dokumentaciju.

4.3. Koraci razvoja

Korišteno je DirectX 9.0c sučelje, koje podržava model sjenčanja 3.0, ali ovaj je program moguće prevesti i unutar modela sjenčanja 2.0 jer nema pretjerano kompliciranih računica.

Najjednostavnije scene u programima koji crtaju grafiku na ekran, rade perspektivno ispravan prikaz objekata, teksturiranje objekata, korištenje jednostavnog osvjetljenja (ambijentalno i difuzno) i animaciju. Sav taj posao je moguće smjestiti u efekt – program za sjenčanje. Iako na prvi pogled djeluje kao da je to malo previše posla odjednom, poznavajući matematiku grafike i jezik HLSL, riječ je o svega nekoliko desetaka linija koda.

4.3.1. Sjenčanje vrhova

U programu za sjenčanje vrhova potrebno je za ovaj primjer napraviti sljedeće stvari:

- Transformacija svijeta, pogleda i projekcije
- Animiranje teksture
- Računanje vektora svjetlosti

Za početak je potrebno definirati koji podatci će ulaziti u cjevovod sjenčanja vrhova i što će biti rezultat ovog koraka. Za transformaciju nam je potrebna pozicija točke, za animiranje teksture trebaju početne koordinate za danu točku, a za svjetlost nam treba normala. Izlaz će sadržavati promijenjenu poziciju točke, nove koordinate teksture i koeficijent za računanje difuzne svjetlosti. Taj dio koda je prikazan u isječku 4.3.

```
struct VS_OUTPUT
{
	float4 Pos: POSITION0;
	float2 Txr: TEXCOORD0;
	float  koefDifuse: TEXCOORD1;
};

VS_OUTPUT vs_main( float4 inPos: POSITION0,
                   float2 inTxr: TEXCOORD0,
                   float3 normal: NORMAL0  )
{ ... }
```

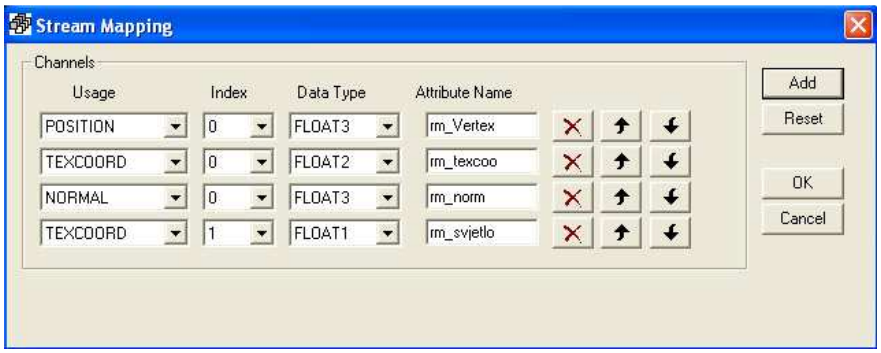
Isječak koda 4.3 Definicija ulaznih i izlaznih podataka

Izlazna struktura VS_OUTPUT sadrži one podatke koji su ranije navedeni. Ona će biti rezultat posla sjenčanja vrhova koji se obavlja u vs_main. Ulazni parametri su također svi oni ranije navedeni.

Korištenjem semantičkih oznaka su ovdje naznačene vrijednosti koje se prenose po kanalima cjevovoda. Tako varijabla `Pos` predstavlja poziciju točke iz kanala `POSITION0`,

`inTxr` predstavlja početne koordinate iz kanala `TEXCOORD0` itd. Broj na kraju oznake je indeks kanala nekog tipa, a koristi se kako se ne bi miješale različite vrijednosti. Tako se u ovom slučaju koordinate teksture prenose preko varijabli `Txr` i `inTxr` po kanalu `TEXCOORD0`, a `koefDifuse` se prenosi kanalom `TEXCOORD1`. Ta zadnja varijabla će biti objašnjena kasnije, njen kanal je odabran proizvoljno.

Ukupan izgled kanala je moguće definirati u RenderMonkey-u i izgleda kao na slici 4.2.



Slika 4.2 Mapa kanala u cjevovodu

Kada je definiran kanal podataka, moramo ubaciti vlastite podatke koje ćemo koristiti u sjenčanju. Potrebno je dodati matricu transformacije svijeta, matricu pogleda i projekcije, nekakve varijabilne vrijednosti za animaciju teksture i poziciju svjetla kao što je prikazano na isječku koda 4.4.

```
float4x4 matViewProjection;
float4x4 matWorld;
float fCosTime0_X;
float fSinTime0_X;
float3 light_position;
```

Isječak koda 4.4 Globalne varijable

Ove varijable moguće je definirati pomoću RenderMonkey-a. Prve četiri varijable, matrice svijeta, pogleda i projekcije, kosinus vremena i sinus vremena alat postavlja automatski. One ovise o poziciji kamere u prostoru i o vremenu. Zadnju varijablu postavljamo proizvoljno, a ona će predstavljati poziciju svjetlosti u prostoru.

Sad treba napisati kod programa koji će napraviti sve željene operacije. Izlaznu varijablu Out definiramo preko ranije navedene strukture. Transformacija svijeta svodi se na množenje točke u prostoru inPos s prethodno definiranom matricom. Koordinate teksture su transformiranje koristeći kosinus i sinus vremena. Ove operacije su prikazane u isječku 4.5.

```
VS_OUTPUT Out;
Out.Txr = Txr1;
Out.Txr.y = mul( abs(fCosTime0_X)+0.5, Txr1.y);
Out.Txr.x  = mul( abs(fSinTime0_X)+0.5, Txr1.x);
Out.Pos = mul( inPos, matWorld );
```

Isječak koda 4.5 Transformacija pogleda i projekcije te animacija teksture

Slijedi računanje koeficijenta za difuzno svjetlo. Prema formuli za svjetlo navedenoj u poglavlju 1.4, svaka od tri komponente boje svjetla se za difuznu svjetlost množi sa skalarnim umnoškom normaliziranih vektora normale i vektora prema svjetlu.

$$I = I_d k_d (\hat{L} \cdot \hat{N})$$

Umnožak ta dva vektora je konstantan za sve komponente boje pa ga možemo pohraniti u varijablu `koefDifuse` i iskoristiti kod sjenčanja fragmenata. Svjetlo se računa na temelju pozicije točke u svijetu stoga treba paziti kako se pozicija točke ne bi transformirala sa matricom pogleda i projekcije prije računanja vektora prema svjetlu. Nakon računanja smjera svjetlosti, pozicija točke se može konačno transformirati.

Problem bi mogla predstavljati normala koja je povezana sa netransformiranom točkom. Rješenje leži u transformaciji normale matricom svijeta i računanja gornje formule sa tom normalom.

Spoj svih ovih operacija rezultira kodom u isječku 4.6.

```
float3 lightDirection = light_position - Out.Pos;
lightDirection = normalize(lightDirection);
Out.Pos = mul( Out.Pos, matViewProjection );

float4 normalCpy = mul( float4( normal.xyz, 0 ), matWorld );
float3 transNormal = normalize( normalCpy.xyz );
Out.koefDifuse = saturate( dot( lightDirection, transNormal ) );
```

Isječak koda 4.6 Računanje skalarnog umnoška normale i vektora prema svjetlu

Konačan kod nastaje spajanjem ovih dijelova i prikazan je u isječku 4.7.

```
float4x4 matViewProjection;
float4x4 matWorld;
float fCosTime0_X;
float fSinTime0_X;
float3 light_position;

struct VS_OUTPUT
{
   float4 Pos: POSITION0;
```

```
    float2 Txr: TEXCOORD0;
    float  koefDifuse: TEXCOORD1;
};

VS_OUTPUT vs_main( float4 inPos: POSITION0,
                   float2 inTxr: TEXCOORD0,
                   float3 normal: NORMAL0
                   )
{
    VS_OUTPUT Out;

    Out.Txr = inTxr;
    Out.Txr.y = mul( abs(fCosTime0_X)+0.5, inTxr.y);
    Out.Txr.x  = mul( abs(fSinTime0_X)+0.5, inTxr.x);

    Out.Pos = mul( inPos, matWorld );
    float3 lightDirection = light_position - Out.Pos;
    lightDirection = normalize(lightDirection);
    Out.Pos = mul( Out.Pos, matViewProjection );

    float4 normalCpy = mul( float4( normal.xyz, 0 ), matWorld );
    float3 transNormal = normalize( normalCpy.xyz );
    Out.koefDifuse = saturate( dot(lightDirection, transNormal) );

    return Out;
}
```

Isječak koda 4.7 Konačni program za sjenčanje vrhova

4.3.2. Sjenčanje fragmenata

U programu za sjenčanje fragmenata potrebno je napraviti sljedeće stvari:

- Uzorkovati teksturu
- Izračunati osvjetljenje

Kao i kod programa za sjenčanje vrhova potrebno je definirati koji se kanali koriste za ulaz, a koji za izlaz podataka te definirati koje se globalne varijable koriste.

Od ulaznih podataka koriste se koordinate tekstura i skalarni umnožak za difuznu komponentu svjetla izračunati prilikom sjenčanja vrhova. Izlazni podatak je boja pa nije potrebno posebno navoditi izlaznu strukturu podataka. Boja nije navedena ni na prikazu kanala na slici 4.2 jer je sjenčanje fragmenata prije kraja cjevovoda i podrazumijeva se da će biti na izlazu.

Globalni podatci koji se ovdje koriste su tekstura koja se uzorkuje i svi elementi potrebni za formulu svjetlosti, a to su koeficijent ambijentalne i difuzne svjetlosti te intenziteti tih svjetlosti.

Uzorkovanje teksture radi se korištenjem ugrađene funkcije koja prima koordinate i uzorak teksture. Povratna vrijednost je boja.

Svjetlost se računa prema formuli navedenoj u poglavlju 1.4. Konačni rezultat svjetlosti se primjenjuje na boju tako da se pojedine komponente pomnože.

Konačni rezultat je program za sjenčanje fragmenata dan u isječku 4.8.

```
sampler Texture0;
float kAmbient;
float kDifuse;
float3 iSource;
float3 iAmbient;

float4 ps_main( float2 Txr: TEXCOORD0,
                float  koefDifuse: TEXCOORD1  ): COLOR0
{
   float4 baseColor = tex2D( Texture0, Txr );

   float3 light = kAmbient * iAmbient + kDifuse * koefDifuse * iSource;
   float3 baseColLight = baseColor.rgb * light.rgb;

   return float4( baseColLight.rgb, baseColor.a );
}
```

Isječak koda 4.8 Konačni program za sjenčanje fragmenata

4.4. Rezultati

Koristeći dostupne alate i znanje računalne grafike, moguće je napraviti ovakav jednostavan program za sjenčanje u vrlo kratkom vremenskom periodu. U startu su rezultati sasvim solidni, a u više vremena i rada, mogu se postići iznimno lijepi efekti.

Implementiranom programu dodan je još jedan prolaz kako bi se prikazala mogućnost stvaranja različitih efekata i njihova primjena u istoj sceni.

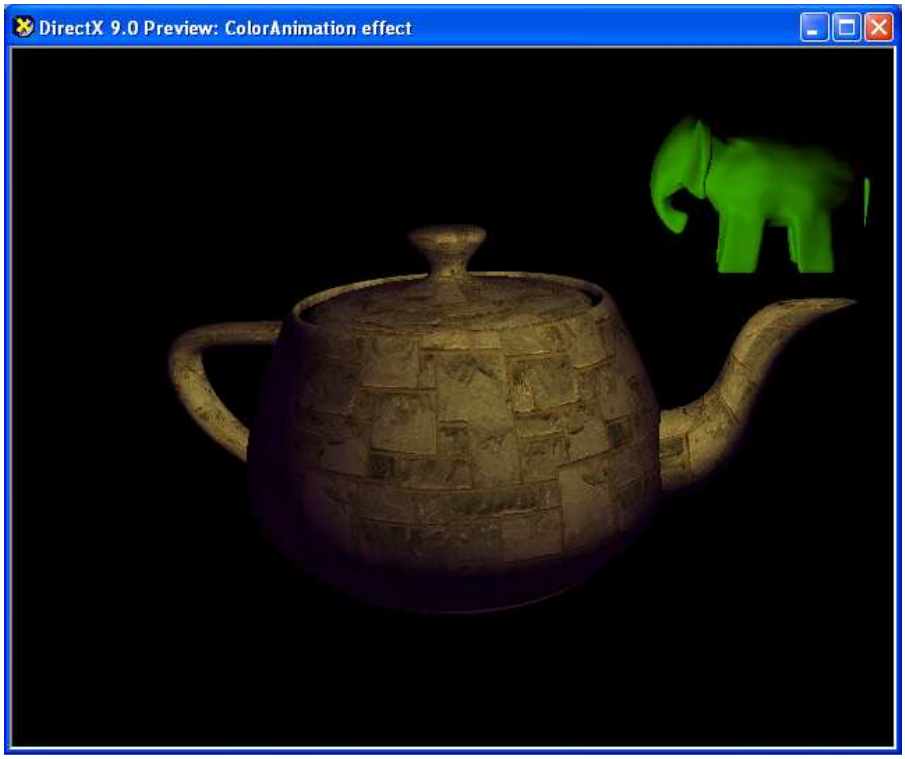
Drugi prolaz istovjetan je prvom samo što ne uzorkuje teksturu, već koristi kosinus i sinus vremena kako bi se odredila boja.

Jedina bitna razlika između prolaza je u sjenčanju fragmenata, gdje se boja računa po formuli kao što je prikazno u isječku 4.9.

```
float4 baseColor = float4(
                    abs( fCosTime0_X ),
                    abs( fSinTime0_X*fSinTime0_X ),
                    fCosTime0_X * fSinTime0_X + abs( fCosTime0_X )/2.0f,
                    1.0f);
```

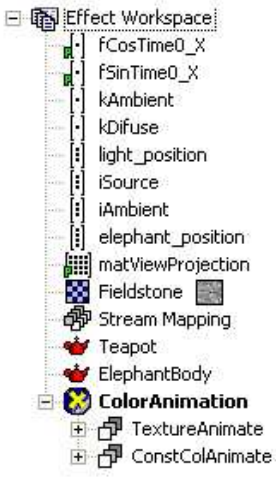
Isječak koda 4.9 Računanje boje pomoću kosinusa i sinusa vremena

U RenderMokey-u je moguće svaki prolaz koristiti na samo jednom objektu pa je tako prvi prolaz korišten na čajniku, a drugi prolaz na slonu i konačni rezultat je prikazan na slici 4.3.

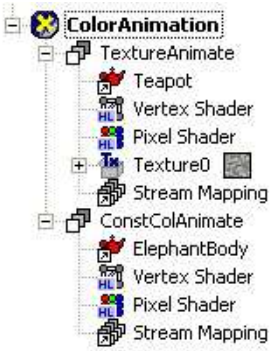


Slika 4.3 ColorAnimation efekt

U RenderMonkey radnom prostoru nalaze se sve globalne varijable koje smo koristili (slika 4.4.), a efekt nazvan ColorAnimation ima dva prolaza, svaki sa svojim modelom i programima za sjenčanje (slika 4.5.).



Slika 4.4 Globalne varijable efekta



Slika 4.5 Struktura efekta

5 Primjena programa za sjenčanje

Razvijeni program za sjenčanje potrebno je uklopiti u programsku potporu, tj. grafičku aplikaciju. Pretpostavlja se da je programer odradio dobar posao i da je program postavljen i jedva čeka početi sjenčati. Detalji oko postavljanja prozora, inicijalizacije DirectX sučelja i reagiranja na kontrole nisu opisani jer ne ulaze u temu rada.

5.1. Odabir okruženja

Programsko rješenje napravljeno je koristeći jezik C++ i aplikacijsko programsko sučelje DirectX [7].

Korišteni alat je Visual Studio 2010, dostupan studentima preko MSDNAA servisa i servisa Dreamspark [8].

5.2. Uklapanje programa za sjenčanje

Datoteka efekta koju generira RenderMonkey može biti malo zakomplicirana i puna redundantnih deklaracija pa ju je za početak poželjno preurediti. Najjednostavniji format je popis globalnih varijabli na početku, a zatim definicije pojedinih programa za sjenčanje. Na kraju se nalazi definicija tehnika, što su u stvari grupacije prolaza koje omogućuju definicije različitih efekata u istoj datoteci ili ostvarivanje različito prevedenih programa za sjenčanje kako bi se postigla kompatibilnost sa starijim sustavima. Primjer definiranja tehnike je u isječku 5.1.

```
technique ColorAnimation
{
   pass TextureAnimate
   {
      VertexShader = compile vs_2_0 TextureAnimate_vs_main();
      PixelShader = compile ps_2_0 TextureAnimate_ps_main();
   }
   pass ConstColAnimate
   {
      VertexShader = compile vs_2_0 ConstColAnimate_vs_main();
      PixelShader = compile ps_2_0 ConstColAnimate_ps_main();
   }
}
```

Isječak koda 5.1 Tehnika razvijenog efekta

Inicijalizacija efekta

Nakon inicijalizacije DirectX-a i postavljanja prozora, treba ubaciti kod koji inicijalizira efekt. Potreban kod izgleda kao u isječku 5.2.

```
// deklaracija efekta - globalno
LPD3DXEFFECT shaderEffect = NULL;
...
LPD3DXBUFFER pBufferErrors = NULL;     // spremnik za greške
// naredba za učitavanje efekta
D3DXCreateEffectFromFile( d3dDevice, "Za_rad_2.fx", NULL, NULL, 0, NULL,
                          &shaderEffect, &pBufferErrors );
```

Isječak koda 5.2 Inicijalizacija efekta

Varijablu efekta treba deklarirati globalno, kako bi se mogla koristiti svugdje u programu. Naredbom `D3DXCreateEffectFromFile` se otvara datoteka predana kao drugi argument i koristeći direct3D uređaj `d3dDevice` se u varijablu `shaderEffect` inicijaliziraju potrebni podatci za rad efekta. Varijabla `pBufferErrors` služi kako bi se mogle pratiti i dojaviti greške pri prevođenju efekta.

Prije korištenja efekta treba postaviti njegove globalne varijable, a to su sve one navedene u prethodnom poglavlju: `matViewProjection`, `matWorld`, `fCosTime0_X`, `fSinTime0_X`, `light_position`, `kAmbient`, `kDifuse`, `iSource`, `iAmbient`. Te varijable treba postaviti svaki put prije pozivanja efekta, tako da je potrebno taj posao odvojiti u zasebnu metodu. Potreban kod je prikazan u isječku 5.3.

```
void setTechniqueVariables( void )
{
      // postavljanje matrice pogleda i projekcije
      setShaderWorldView( 1.0f, 0.0f, 5.5f, D3DXToRadian(fSpinX),
                                    D3DXToRadian(fSpinY), 0.0f, 1.0f
      // postavljanje defaultne teksture
      shaderEffect->SetTexture( "inTexture", defaultTexture );
      // postavljanje kosinusa i sinusa vremena
      shaderEffect->SetFloat( "fCosTime0_X", cos( timeAccum ));
      shaderEffect->SetFloat( "fSinTime0_X", sin( timeAccum ));
      // postavljanje svjetla
      shaderEffect->SetFloatArray( "light_position", light_position, 3 );
      shaderEffect->SetFloat("kAmbient", kAmbient);
      shaderEffect->SetFloat("kDifuse", kDifuse);
      shaderEffect->SetFloatArray( "iSource", iSource, 3 );
      shaderEffect->SetFloatArray( "iAmbient", iAmbient, 3 );
      shaderEffect->CommitChanges();
}
```

Isječak koda 5.3. Postavljanje varijabli efekta

Podrazumijeva se da su sve varijable koje se prenose u efekt globalno definirane s nekim vrijednostima. Jedini varijabilni parametri su kosinus i sinus vremena, koje je potrebno računati tako da se u svakom crtanju poveća `timeAccum`, spremnik proteklog vremena, i iskoristi za računanje matematičkih funkcija.

Sve funkcije za postavljanje parametara pozivaju se nad efektom, a za pojedine tipove podataka potrebno je pozivati pripadajuće metode. Tako se teksture postavljaju sa `SetTexture`, jednodimenzijske varijable sa `SetFloat`, nizovi sa `SetFloatArray`, a matrice sa `SetMatrix`. Postavljanje matrica svijeta te pogleda i projekcije odvojeno je u zasebnu metodu, koja računa transformacije svijeta i projekcije te postavlja rezultat u efekt. Kada su postavljene sve varijable potrebno je pozvati funkciju `CommitChanges` da bi se te varijable i primijenile u efektu.

Učitavanje objekata

Efekt predstavlja elemente grafičkog cjevovoda i njemu je potrebno poslati neke podatke. To su u stvari grafički objekti, njihove normale, koordinate tekstura i slično. Dakle treba neke objekte prvo učitati. Korištena su dva objekta, model stolca i sfere, tekstura stolca i jedna nevezana tekstura. Kod učitavanja svih objekata je u isječku 5.4.

```
      // deklaracija mreže sfere i potrebnih varijabli
LPD3DXMESH sphere_mesh = NULL;
LPD3DXBUFFER sphere_material = NULL; DWORD sphere_material_num = 0;
      // deklaracija mreže stolca i potrebnih varijabli
LPD3DXMESH chair_mesh = NULL;
LPD3DXBUFFER chair_material      = NULL; DWORD chair_material_num = 0;
      // deklaracija tekstura
LPDIRECT3DTEXTURE9 defaultTexture = NULL;
LPDIRECT3DTEXTURE9 chair_tex = NULL;
...

      // Učitavanje sfere
D3DXLoadMeshFromX("modeli\\sphere.x", D3DXMESH_SYSTEMMEM, d3dDevice, NULL,
            &sphere_material,NULL, &sphere_material_num, &sphere_mesh);
      // Učitavanje stolca
D3DXLoadMeshFromX("modeli\\chair.x", D3DXMESH_SYSTEMMEM, d3dDevice, NULL,
            &chair_material, NULL, &chair_material_num, &chair_mesh );
      // Učitavanje nezavisne teksture
D3DXCreateTextureFromFile( d3dDevice, "tekstura2.bmp", &defaultTexture );

      // učitavanje teksture stolca iz podataka o materijalima
      // stolac ima samo jedan materijal
D3DXMATERIAL* materijal = (D3DXMATERIAL*)chair_material->GetBufferPointer();
LPSTR filename = concat("modeli\\",materijal[0].pTextureFilename);
D3DXCreateTextureFromFile( d3dDevice, filename, &chair_tex );
```

Isječak koda 5.4 Učitavanje objekata i tekstura

Svi objekti i teksture deklarirani su globalno. Funkcija `D3DloadMeshFromX` puni varijable objekta s potrebnim podatcima, koristi lokaciju datoteke objekta, a podatke o točkama, susjedstvu i normalama pohranjuje u varijablu `sphere_mesh` i `chair_mesh`. Podatci o materijalima, pohranjuju se u zasebne spremničke varijable `sphere_material` i `chair_material`.

Teksture se učitavaju funkcijom `D3DXCreateTextureFromFile` kojoj se predaju lokacija teksture i varijabla teksture koja će se napuniti podatcima. Lokacija teksture stolca je pohranjena u podatcima o materijalima, tako da prvo treba spremnik sphere_material pretvoriti u pokazivač na varijablu materijala (`D3DXMATERIAL* materijal`) iz koje se lako dolazi do imena teksture.

Pokretanje scene

U djelu koda koji direct3D uređaju signalizira da započinje scena treba smjestiti pokretanje efekta. Efektu se prvo postavi tehnika koju će primijeniti, zatim se postavljaju varijable kao što je ranije opisano i na kraju se prolazi kroz prolaze. Unutar prolaza se postavljaju teksture i matrice za svaki objekt, a zatim se prosljeđuju mreže objekata. Konačni kod scene je prikazan na isječku 5.5.

```
d3dDevice->BeginScene();

shaderEffect->SetTechnique( "ColorAnimation" );
setTechniqueVariables();

UINT uPasses;
shaderEffect->Begin( &uPasses, 0 );

for( UINT uPass = 0; uPass < uPasses; ++uPass )
{
	shaderEffect->BeginPass( uPass );
	if(uPass == 0){ 		// TextureAnimate
		// crtanje velikog stolca
		setShaderWorldView( -1.0f, -3.5f, 10.0f, -D3DXToRadian(fSpinX),
					D3DXToRadian(fSpinY), 0.0f, 1.0f);
		shaderEffect->SetTexture( "inTexture", chair_tex );
		shaderEffect->CommitChanges();
		chair_mesh->DrawSubset(0);

		// crtanje velike sfere
		setShaderWorldView( 2.0f, -2.5f, 7.0f, D3DXToRadian(fSpinX),
					-D3DXToRadian(fSpinY), 0.0f, 0.08f);
		shaderEffect->SetTexture( "inTexture", defaultTexture );
		shaderEffect->CommitChanges();
		sphere_mesh->DrawSubset(0);
	}
```

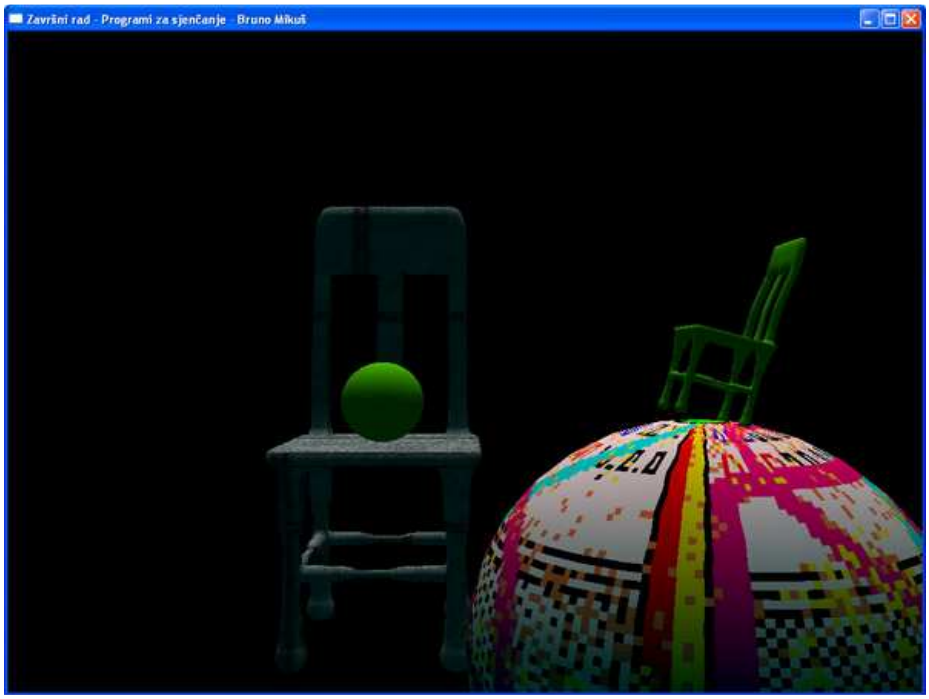
```
        else{                               //ConstColAnimate
                // crtanje male sfere
                setShaderWorldView( -1.0f, -0.5f, 10.0f, D3DXToRadian(fSpinX),
                                            -D3DXToRadian(fSpinY), 0.0f, 0.02f);
                shaderEffect->CommitChanges();
                sphere_mesh->DrawSubset(0);

                // crtanje malog stolca
                setShaderWorldView( 2.0f, -0.5f, 7.0f, -D3DXToRadian(fSpinX),
                                            D3DXToRadian(fSpinY), 0.0f, 0.3f);
                shaderEffect->CommitChanges();
                chair_mesh->DrawSubset(0);
        }
        shaderEffect->EndPass();
}
shaderEffect->End();
d3dDevice->EndScene();
```

Isječak koda 5.5. Konačni kod scene koji koristi sjenčanje

Pri započinjanju scene, metoda `Begin` postavlja varijablu `uPasses` s brojem prolaza. U `for` petlji se iterira kroz sve prolaze koji postoje korištenjem funkcije `BeginPass`. Na taj je način moguće različitim prolazima proslijediti različite objekte. Tako je u ovom primjeru prolazu koji animira teksturu proslijeđen veliki stolac i velika sfera, s tim da je prije crtanja mreže potrebno postaviti matricu transformacije i teksturu koje želimo koristiti za taj objekt. Kako bi se zadane promjene vidjele u efektu treba obavezno pozvati funkciju `CommitChanges`. U drugom prolazu prosljeđujemo manju sferu i manji stolac. Njima je potrebno samo postaviti matrice, jer prolaz za animiranje boje ne koristi teksture.

Zanimljivo je primijetiti kako se za objekte poziva funkcija `DrawSubset(0)`. 3D objekti mogu se sastojati od više poddijelova, a svaki dio može imati definiran svoj materijal. To znači da se složeniji objekti mogu po dijelovima crtati s različitim prolazima.

Na kraju će biti nacrtana scena kao ona prikazana na slici 5.1, s tim da je izvor svjetlosti iznad objekata na sceni.



Slika 5.1 Pokrenuta grafička aplikacija

Na ovaj način bilo bi moguće crtati mnogo kompleksnije scene, s mnogo kompleksnijim efektima i s mnogo boljim efektima. Ali princip bi se sveo na ovaj koji je pokazan ovim radom.

6 Konačni rezultati

U ovom poglavlju biti će ukratko riječ o postignutim rezultatima, performansama, utjecaju različitih parametara i eventualnom daljnjem razvoju implementacije.

6.1. Analiza rezultata i performansi

Iako na prvi pogled rezultati ne djeluju pretjerano impresivno, kada se usporedi količina potrebnog koda i vremena utrošenog na izradu ovog rada i sličnih projekata gdje se ne koriste efekti, mislim da programi za sjenčanje uvelike pridonose odvajanju funkcionalnosti, ostvarivanju složenosti i povećanju razumljivosti koda.

Projekt bez programa za sjenčanje zahtjeva više računanja, kao što je računanje tekstura, svjetlosti i transformacija, upravo onih poslova koje programi za sjenčanje odrade u par linija koda.

Korišteni su objekti stolac i sfera, čiji su podatci o vrhovima i poligonima navedeni u tablici 6.1.

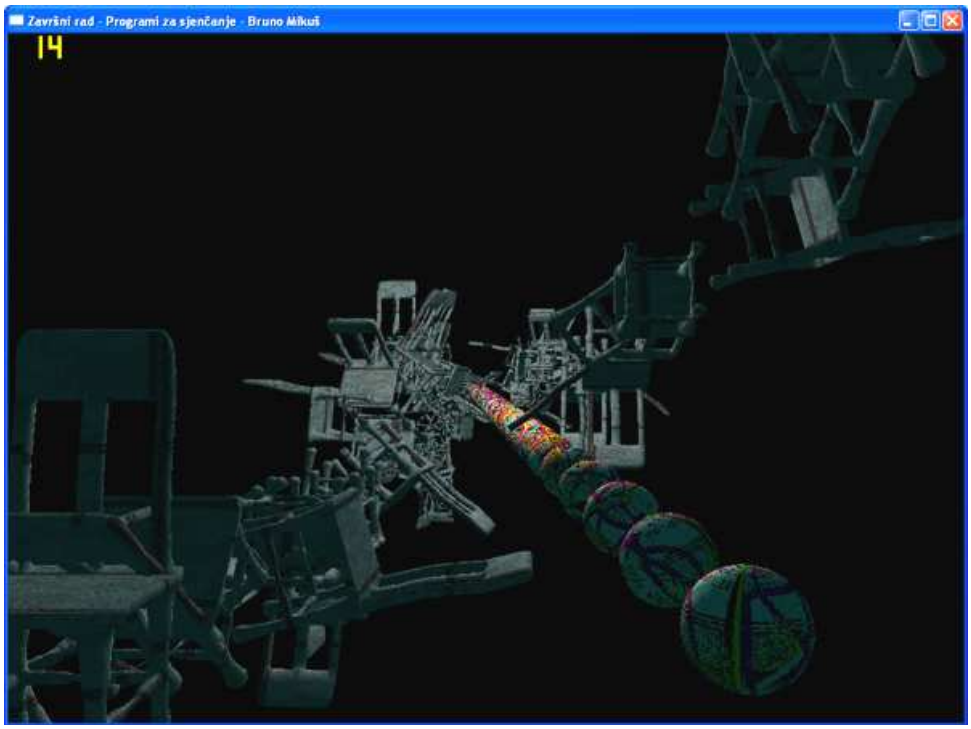
Tablica 6.1 Broj vrhova i poligona za korištene objekte

Objekt	Broj vrhova	Broj poligona
Stolac	3258	4304
Sfera	561	960

Performanse možemo otprilike procijeniti crtanjem većeg broja objekata. To je jednostavno za implementirati, samo treba dio koda koji crta pojedini objekt ubaciti u petlju. Na slici 6.1 prikazana je scena sa 300 objekata. Razlike u brzini crtanja između 300 i 400 objekata su gotovo neprimjetne, ali se u oba slučaja lako uočava da se grafički procesor muči. Kakav je odnos broja slika u sekundi i broja obrađenih vrhova kod različitog broja objekata na sceni pokazuje tablica 6.2.

Tablica 6.2 Broj slika u sekundi, te broja obrađenih vrhova

Broj objekata			Broj slika u sekundi	Broj vrhova
Stolaca	Sfera	Ukupno		
2	2	4	570	7,638
10	10	20	230	38,190
40	40	80	65	152,760
100	50	150	28	353,850
200	50	250	17	679,650
250	50	300	14	842,550
300	100	400	11	1,033,500

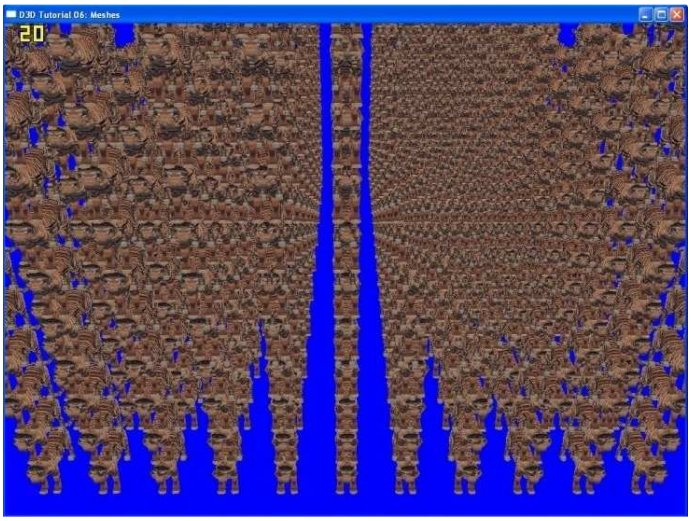


Slika 6.1 Kompleksna scena

Broj slika u sekundi se prema broju vrhova ponaša prema funkciji – tj. za manji broj vrhova, male promjene će stvarati veću razliku u broju slika u sekundi, a za veći broj vrhova, male razlike će stvarati manju razliku u broju slika u sekundi. To je najlakše za primijetiti u tablici da kada se broj vrhova poveća sa 8 tisuća na 40 tisuća (za 32 tisuće) broj slika u sekundi padne za 240. S druge strane kada se broj vrhova poveća sa 850 tisuća na 1 milijun (za 150 tisuća), broj slika u sekundi padne za 3.

Potrebno je još napraviti i usporedbu sa programom koji crta bez korištenja programa za sjenčanje, koristeći mogućnosti DirectX sučelja. Na slici 6.2 je prikazana takva scena sa 1210 tigrova, a korišten je malo izmijenjen primjer koda koji dolazi uz DirectX razvojni alat.

Svaki tigar napravljen je od 303 točke, što znači da se procesuira 366 tisuća točaka uz održavanje broja sličica na 20 u sekundi. Usporedba sa sličnim slučajem iz tablice 6.2 govori kako korištenje programa za sjenčanje omogućuje bolje rezultate uz isti broj obrađenih vrhova. Najzanimljivije kod ove usporedbe je to, što program za sjenčanje još animira teksturu i boju te računa svjetlost, dok program bez sjenčanja, samo transformira objekte i crta teksture na njih.



Slika 6.2 Scena bez programa za sjenčanje

Brzina crtanja i broj slika u sekundi će naravno ovisiti o korištenom računalu i njegovim mogućnostima, te o kompleksnosti efekta koji se primjenjuje.

6.2. Utjecaj različitih parametara

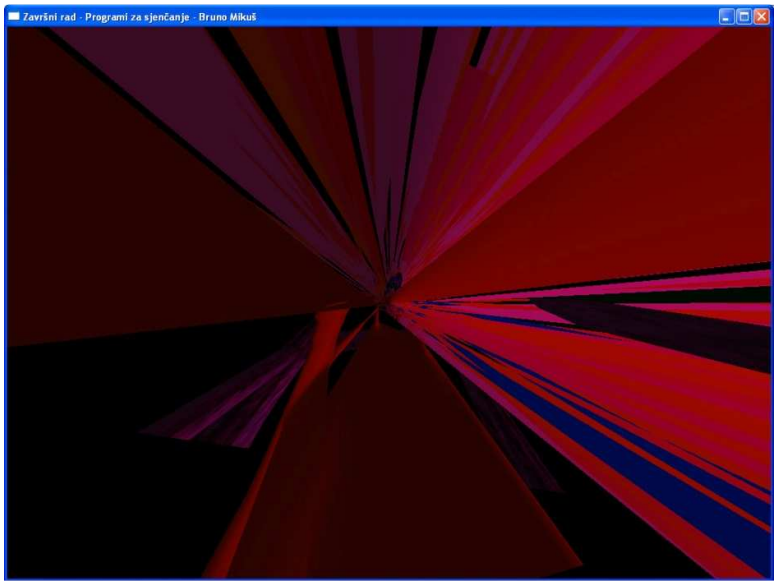
Postoji mnogo varijabli koje ulaze u program za sjenčanje, a mogu imati utjecaja na rezultat.

Mijenjanjem `kAmbient` i `kDifuse` varijabli mijenjati će se osvjetljenje scene, a mijenjanjem `iSource` te `iAmbient`, mijenjati će se boja osvjetljenja. Korištenjem tih parametara postiže se nekakav ugođaj pa tako jače ambijentalno svjetlo ostavlja dojam dnevnog svjetla, dok slabije ambijentalno svjetlo stvara dojam mračnog i opasnog prostora.

Varijablom `light_position` je vrlo jednostavno mijenjati položaj svjetla, te je svega nekoliko linija koda dovoljno kako bi se koristeći miš to svjetlo micalo.

Ako se varijable kosinus i sinus vremena postavljaju uvijek na istu vrijednost, neće biti animacije. Isto tako, ako nije dobro napravljeno računanje vremena animacija teksture može biti sporija ili brža, što možda i nije željeni efekt. Naravno, ako je cilj imati brzu animaciju teksture, to se jednostavno napravi skaliranjem vremenske varijable s kojom se računa kosinus i sinus.

Unosom neispravne matrice pogleda i projekcije vrlo vjerojatno neće biti ništa nacrtano na ekranu tako da je potrebno paziti na orijentiranost koordinatnog sustava i na pravilan redoslijed primjena matrica. Na slici 6.2 je prikazana scena koja nastaje transponiranjem matrice pogleda i projekcije, a izgleda kao da je svijet eksplodirao.



Slika 6.2 Pogreška sa matricom pogleda i projekcije

Problem isto može nastati kada se koriste objekti koji nemaju ispravne normale. Tada neće biti utjecaja difuznog svjetla već samo ambijentalnog i objekt će biti mračan. Isto tako, ako objekt nema ispravne teksturne koordinate, mogući su čudni rezultati. Općenito, inicijalizirani objekt bi trebao biti ispravan i sadržavati sve potrebne informacije.

6.3. Daljnji razvoj

Problem ove implementacije je taj što se transformacija objekta u svijetu događa u programu za sjenčanje, što možda nije prikladno za ostvarivanje nekih drugih mogućnosti poput detekcije kolizije i interakcije objekata. Rješenje bi bilo u transformaciji objekata na prije prosljeđivanja jedinici za sjenčanje, koja bi onda trebala samo odraditi transformaciju pogleda i projekcije.

Ova implementacija crta samo 2 tipa objekta u velikim količinama, tako da bi očigledno proširenje bilo implementacija funkcija i struktura koje bi omogućile lakše korištenje većeg broja različitih objekata.

Što se tiče programa za sjenčanje, tu je ograničenje mašta dizajnera. Prvi korak bi recimo bilo dodavanje zrcalnog osvjetljenja, kako bi model svjetla bio potpun. Zatim bi bilo zanimljivo dodati prolaz za računanje okruženja, kako crtani objekti ne bi bili u mračnom prostoru.

Poželjno bi bilo napraviti kontrolu mogućnosti grafičke kartice platforme na kojoj se program pokreće. To uključuje stvaranje tehnika koje prevode program za sjenčanje u višoj ili nižoj inačici, stvaranje tehnika sa više ili manje kompleksnim programima za sjenčanje i slično.

Zaključak

Programi za sjenčanje odličan su mehanizam kontrole grafičkog cjevovoda, njegove optimizacije i najboljeg iskorištavanja. Programiranjem najniže razine, one vezane uz vrhove i fragmente moguće je ostvariti lijepe efekte, uz manje posla programera, dizajnera, a i računala koje grafičku aplikaciju pokreće. Potrebe industrije zabave iz dana u dan su veće, a ovakvim će tehnologijama biti moguć njihov daljnji rast i razvoj. Samo je pitanje vremena kada će igre izgledati poput najmodernijih filmskih uradaka, a filmovi biti stvarniji od stvarnosti.

Jezici za sjenčanje nisu komplicirani ako se uče uz dobre temelje i uz dobre alate. Kako su sintaksom slični mnogim modernim programskim jezicima, prosječnom programeru su potrebna samo znanja o računalnoj grafici. Svi viši jezici za sjenčanje pružaju slične mogućnosti, a bitno je jedino razmatrati za kakvu platformu se program razvija. Prijašnji radovi na ovu temu orijentirani su na otvoreniji pristup sa OpenGL sučeljem i GLSL jezikom, tako da ovaj rad predstavlja mali iskorak u drugom smjeru koristeći okolinu ovisnu o Microsoft operacijskim sustavima.

Rezultati dobiveni ovim radom pokazuju kako se programima za sjenčanje postižu veće brzine crtanja na ekran u odnosu na korištenje ne-programirljivog cjevovoda, a da se pritom postižu i bolji efekti. Potrebno je obratiti pozornost na redoslijed računanja pojedinih dijelova efekta i na parametre koji se prenose kako ne bi nastajali neželjeni učinci. Prednost korištenja platformski ovisnog sučelja DirectX su dostupnost mnogih gotovih funkcija i struktura koje ubrzavaju razvoj i olakšavaju razumijevanje programskog koda.

Dodatak A – alati i resursi

Alati

U tablici 1. nalazi se popis korištenih alata te je uz kratak opis pojedinog alata navedena i dostupnost, tj. uvjet korištenja pojedinog alata.

Tablica A.1. Popis korištenih alata za izradu završnog rada

Korišteni alat	Opis	Dostupnost	Referenca
HLSL	Programski jezik za sjenčanje	--	--
C++	Programski jezik programske potpore	--	--
DirectX SDK February 2010	Aplikacijsko programsko sučelje (API)	Besplatan	[7]
Microsoft Visual Studio 2010	Razvojna okolina programske potpore	Besplatan studentima preko MSDNAA	[8]
RenderMonkey 1.82	Razvojna okolina programa za sjenčanje	Besplatan	[6]

Kao alternativa Visual Studiu 2010 mogući su Dev-Cpp, Eclipse i slične razvojne okoline koje podržavaju razvoj C++ aplikacija.

Resursi – računalo

Tablica A.2. Svojstva korištenog računala

Operacijski sustav	Windows XP SP3
Procesor	AMD Athlon 64bit 2.2Ghz
Radna memorija	1 GB DDR

Resursi – grafička kartica

Tablcia A.3 Svojstva korištene grafičke kartice

Naziv	ATI Radeon X1300
Radna memorija	256 MB
Sučelje	PCIe x16
Podržana verzija DirectXa	9.0c
Jedinica za sjenčanje vrhova	2
Jedinica za sjenčanje fragmenata	4

Literatura

[1] Jozić K., Programirljivo grafičko sklopovlje, diplomski rad, Fakultet elektrotehnike i računarstva, srpanj 2006., http://bib.irb.hr/prikazi-rad?&rad=252356, svibanj 2010.

[2] Piljek L., Postupci teksturiranja upotrebom grafičkog procesora, završni rad, Fakultet elektrotehnike i računarstva, lipanj 2008., http://www.zemris.fer.hr/predmeti/irg/Zavrsni/08Piljek/index.html, svibanj 2010.

[3] Mihajlović Ž., Laboratorijska vježba 5: transformacija pogleda i perspektivna projekcija, http://www.zemris.fer.hr/predmeti/irg/labosi/vj5.pdf, svibanj 2010.

[4] Mihajlović Ž., Grafičke primitive i transformacije, http://www.zemris.fer.hr/predmeti/irg/predavanja/3_primitive.pdf, svibanj 2010.

[5] Roosendaal T., Kurdali B., Elephant's dream, http://www.elephantsdream.org/, lipanj 2010.

[6] Render Monkey, http://developer.amd.com/gpu/rendermonkey/Pages/default.aspx, travanj 2010.

[7] Microsoft, DirectX SDK, http://msdn.microsoft.com/en-us/directx/default.aspx, travanj 2010.

[8] Microsoft, Dreamspark, https://www.dreamspark.com/Default.aspx, travanj 2010.

[9] Salford university lecture notes, http://activehelix.com/programming4.html. svibanj 2010.

[10] St-Laurent S., Shaders for game programmers and artists, prvo izdanje, SAD, Thomson Course Technology, 2004.

Sažetak

Programi za sjenčanje

Ovim radom dan je pregled rada grafičkog cjevovoda s posebnim naglaskom na sjenčanje i jedinice za sjenčanje. Opisani su dostupni načini sjenčanja i programski jezici za njihovo ostvarenje. Napravljen je uvod u jezik HLSL te je u njemu implementiran jednostavan program za sjenčanje koji animira teksturu i primjenjuje jednostavno osvjetljenje. Razvijeni program je upotrebljen u implementaciji grafičke aplikacije koja koristi DirectX sučelje za komunikaciju sa grafičkim sklopovljem. Za razvoj su korišteni programski alati Render Monkey i Microsoft Visual Studio 2010. Analizirani su dobiveni rezultati i ustanovljeno je kako se programima za sjenčanje postižu bolje performanse.

Ključne riječi: grafički cjevovod, jedinice za sjenčanje, sjenčanje, sjenčanje vrhova, sjenčanje geometrije, sjenčanje fragmenata, jezici za sjenčanje, HLSL, sintaksa HLSL, DirectX, programi za sjenčanje, primjena programa za sjenčanje, animacija teksture, animacija boje, osvjetljenje.

Abstract

Shader programs

This paper gives an overview of graphics pipeline with an emphasis on the shading process and shading units. Available shading methods have been described, as well as required shading programming languages for their implementation. An introduction to HLSL language has been made, as well as a shading program that animates texture and applies simple lighting. The developed shading program has been used in a graphics application using DirectX interface to communicate with the graphics hardware. Render Monkey and Microsoft Visual Studio 2010 were used as development tools. An analysis of the results was made and it has become apparent that the use of shading programs gives better performance.

Key words: graphics pipeline, shading units, shading, vertex shading, geometry shading, pixel shading, shading languages, HLSL, HLSL syntax, DirectX, shading programs, aplication of shading programs, texture animation, color animation, lighting.