ISSN 0005–1144
ATKAAF 47(3–4), 177–181 (2006)

Uređuje: Mirko Klaić, dipl. ing.

PROGRAMIRLJIVO GRAFIČKO SKLOPOVLJE

UVOD

Iako programirljivo grafičko sklopovlje u profesionalnoj primjeni postoji već nekoliko desetljeća, tek je prije nekoliko godina postalo dostupno na osobnim računalima. Sve do 2001. godine grafičke kartice na osobnim računalima koristile su sklopovski ostvarene funkcije koje su služile za ubrzavanje temeljnih grafičkih operacija. Tako izvedeno sklopovlje ponekad je dosta teško programirati, upravo zbog toga što se ugrađene funkcije ne mogu mijenjati nego im se samo mogu prosljeđivati parametri. Od 2001. godine mnogo toga se promijenilo. Sada su gotovo sve grafičke kartice programirljive, odnosno imaju ugrađen grafički procesor (GPU) koji se može programirati slično kao i centralni procesor računala (CPU). Funkcije se u pravilu izvode kao kratki programi na grafičkom procesoru.

Na tržištu postoji mnogo programskih alata koji olakšavaju izradu programa za grafičke procesore. Takvi programski alati imaju predloške gotovih programa, provjeravaju sintaksu programa, mjere performance, itd. Programski alati obično su besplatni, a mnogi su i otvorenog koda, što korisnicima omogućava da ih prepravljaju i poboljšavaju kako god im to odgovara. Valja napomenuti kako grafički procesori ne moraju isključivo izvršavati programe koji su namijenjeni za računalnu grafiku, nego i sve ostale programe koji intenzivno koriste vektore i matrice. U tom slučaju dobro je poznavati temeljne principe i mehanizme koji se koriste u računalnoj grafici jer je grafička sklopovska oprema u prvom redu prilagođena izvođenju grafičkih zadataka. Općenite zadatke također možemo obavljati na grafičkim karticama, no prvo ih treba prilagoditi raspoloživoj opremi što će zasigurno rezultirati izuzetnim performancama. Kratica GPGPU (engl. *General-Purpose computation on GPUs*) vezana je uz načine programiranja i izvođenja zadataka opće namjene na grafičkoj sklopovskoj opremi.

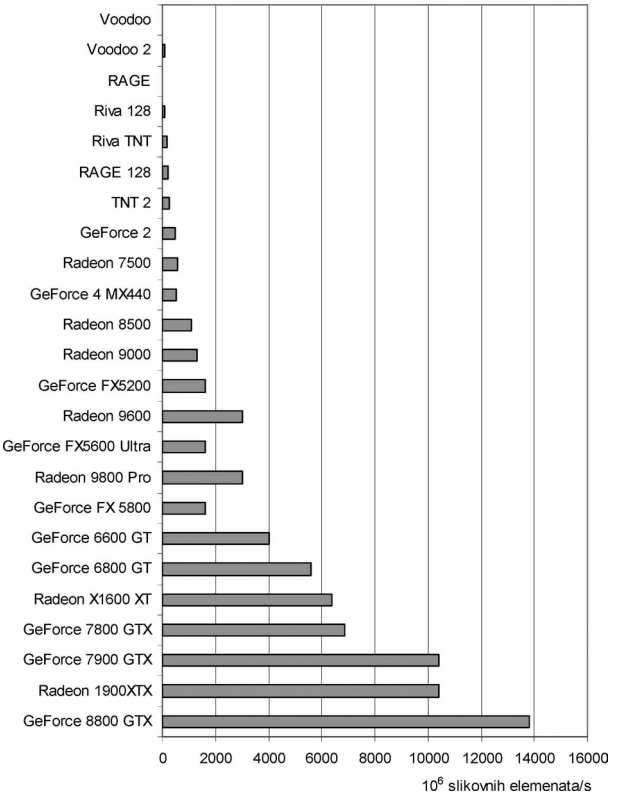
U ovom radu se daje pregled programirljivog grafičkog sklopovlja namijenjenog osobnim računalima. Za pristup grafičkom sklopovlju koriste se dva programska sučelja: *OpenGL* i *DirectX* (točnije, *Direct3D* – podskup DirectX-a za rad s grafikom). Svi opisi odnosit će se na OpenGL koji je otvoren standard, za razliku od DirectX-a koji je u vlasništvu *Microsofta*. OpenGL se nalazi pod nadzorom ARB-a (engl. *Architecture Review Board*), a taj odbor čine proizvođači grafičkog sklopovlja, kao i stručnjaci iz računalne grafike.

RAZVOJ GRAFIČKOG SKLOPOVLJA NA OSOBNIM RAČUNALIMA

Moglo bi se reći da doba 3D računalne grafike na osobnim računalima započinje 1995. godine. Te godine je tvrtka S3 napravila grafičku karticu po imenu ViRGE (*Virtual Reality Graphics Engine*). Grafička kartica ViRGE podržavala je VGA standard i imala mogućnosti ubrzavanja 2D i 3D grafike. No, bila je proglašena »prvim grafičkim usporivačem«. Iako je elementarna podrška za 3D grafiku radila brže nego na centralnom procesoru (CPU), naprednije mogućnosti bile su dosta sporije. Jednako tako, nije imala podršku za OpenGL, pa nije imala ni profesionalnu primjenu.

Sljedeće godine je tvrtka *3dfx* napravila grafičku karticu po imenu *Voodoo*. Ta grafička kartica smatra se prvom pravom 3D ubrzivačkom grafičkom karticom na osobnim računalima. I ona je imala nekoliko ograničenja. Bila je isključivo namijenjena za 3D grafiku, pa se za 2D grafiku morala spojiti dodatna grafička kartica. Osim toga, dosta je loše filtrirala teksture i imala je problema s prikazom velikog broja boja.

Sve naredne grafičke kartice do 2001. godine nastavile su trend sklopovske implementacije funkcija. Prednost sklopovske implementacije funkcija je velika brzina, a nedostatak je nefleksibilnost. Radi toga je ponekad potrebno čak do 5 ili 6 prolaza za izračun složene scene.



Slika 1. Grafičke kartice i brzina izračuna slikovnih elemenata

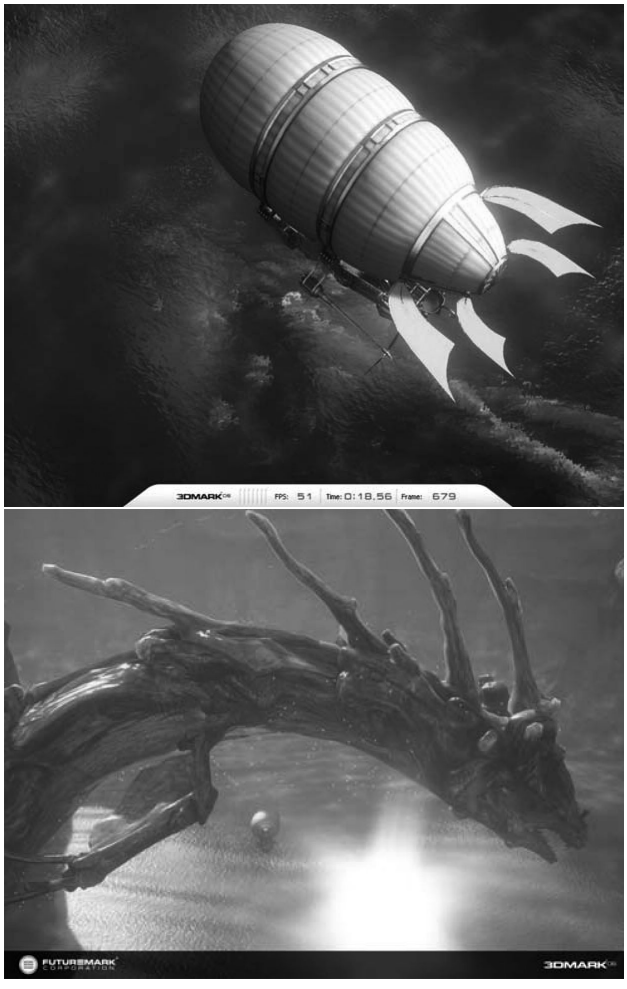
2001. godine izlaze prve programirljive grafičke kartice (NVIDIA GeForce 3). Ta programirljivost ograničena je samo na geometrijsku fazu, dok se rasterizacijska faza i dalje koristi sklopovski ostvarenim funkcijama. Daljnja ograničenja su broj i poredak instrukcija. Uz sva navedena ograničenja, programirljivo sklopovlje ipak je brže, upravo zbog fleksibilnosti. Sada se složene scene mogu izračunati u svega 2 do 3 prolaza. Već iduće godine izlaze grafičke kartice koje imaju programirljivu i rasterizacijsku fazu. Broj instrukcija još je uvijek vrlo ograničen, a ne postoje ni instrukcije grananja (petlje se moraju razmo-

tati u fazi prevođenja programa), ali već je moguće raditi složenije algoritme. Od sada se programi koji se pokreću na grafičkom sklopovlju nazivaju programi za sjenčanje. U grafičkom protočnom sustavu važne faze su geometrijska i rasterizacijska faza. U geometrijskoj fazi imamo programe za sjenčanje vrhova (engl. *vertex shader*), a u rasterizacijskoj fazi programe za sjenčanje fragmenata (engl. *fragment shader*).

Trend brzog razvoja grafičkih kartica nastavio se do pred kraj 2004. godine kada je dosegnuta razina programirljivosti grafičkih kartica usporediva s onom procesora opće namjene. I nakon toga razvoj je nastavljen i grafičke kartice prestižu procesore opće namjene.

Slika 1 prikazuje trend razvoja grafičkih kartica u zadnjih desetak godina. Na grafu je prikazana brzina izračuna slikovnih elemenata za pojedinu grafičku karticu izražena u milijunima slikovnih elemenata u sekundi. Broj elemenata teksture u sekundi znatno je veći, pa tako, na primjer, za GeForce 8800 GTX iznosi $36{,}8 \cdot 10^9$. Ovako ekstreman trend razvoja grafičkih procesora razlog je sve većeg interesa u ostvarivanju njihove primjene za različite namjene.

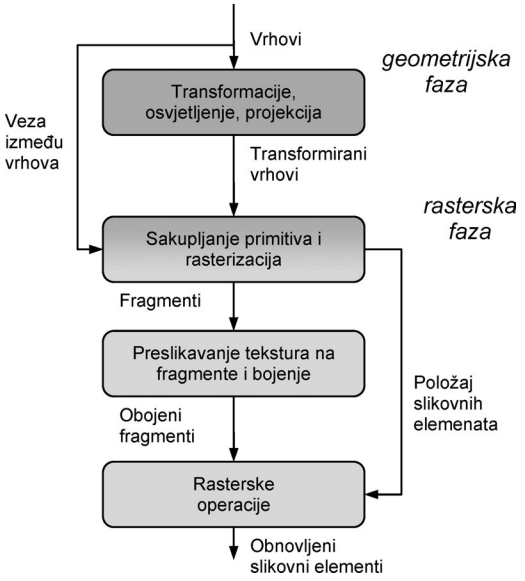
Osnovna funkcija grafičkih kartica je iscrtavanje složenih scena što, naravno, uključuje i primjenu u igrama. Ispitni program koji se često koristi za mjerenje performanci rada grafičke kartice je 3DMark. Na slici 2 prikazane su scene iz tog programa za testiranje.



Slika 2. Scene iz programa za testiranje grafičkih kartica 3DMark06

GRAFIČKI PROTOČNI SUSTAV

Grafički protočni sustav (cjevovod) sastoji se od niza stupnjeva koji obavljaju određene grafičke operacije po fiksnom redoslijedu. Na slici 3 prikazan je vrlo pojednostavljeni grafički protočni sustav.

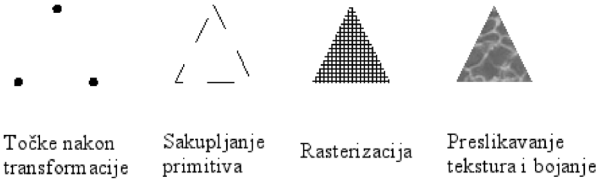


Slika 3. Grafički protočni sustav

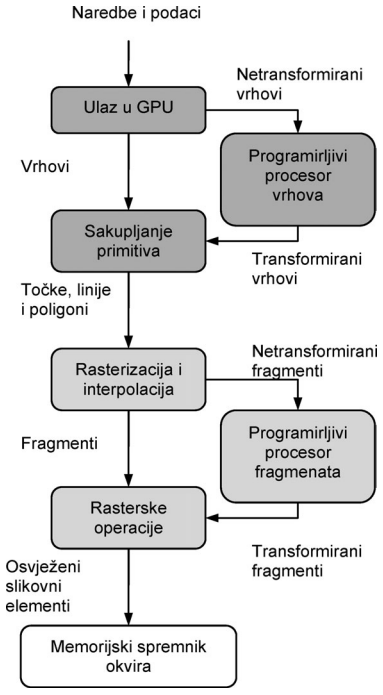
Ulaz u grafički protočni sustav čine koordinate vrhova i ostali parametri koji se postavljaju u grafičkoj aplikaciji. Svi ti parametri transformiraju se nizom matrica radi pravilnog smještanja u scenu, proračuna osvjetljenja i projekcije scene. Ta faza se naziva geometrijska faza (prva kućica na slici 3).

Nakon toga slijedi faza sakupljanja primitiva (iz pojedinih vrhova slažu se geometrijski elementi) i rasterizacija (određivanje prekrivenosti geometrijskih elemenata slikovnim elementima – pikselima). Poslije toga dolaze faze preslikavanja tekstura na fragmente (potencijalne slikovne elemente) i rasterske operacije koje uključuju razne testove koje prolazi fragment da bi se utvrdilo da li će se zapisati u memoriju. Ako se fragment zapiše u memoriju, točnije memorijski spremnik okvira, on će postati slikovni element koji se prikazuje na zaslonu. Te podfaze čine rasterizacijsku fazu (preostala 3 kvadrata na slici 3). Memorijski spremnik okvira se sastoji od nekoliko spremnika (spremnik boje, dubine, maske, akumulacijski, itd.). Oni neće biti opisivani, kao ni pojedine faze rasterskih operacija.

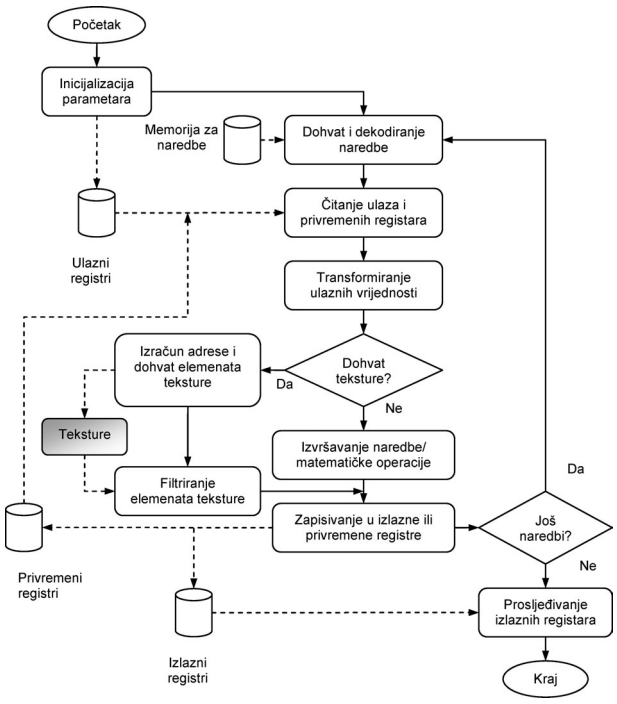
Na slici 4 prikazani su rezultati pojedinih faza kroz koje prolaze geometrijski elementi. Dosadašnji opis grafičkog protočnog sustava odgovara neprogramirljivom sklopovlju. Na slici 5 prikazan je grafički protočni sustav koji odgovara programirljivom sklopovlju.



Slika 4. Rezultati pojedinih faza kroz koje prolaze geometrijski elementi



Slika 5. Grafički protočni sustav koji odgovara programirljivom sklopovlju



Slika 6. Procesiranje fragmenata

Slika 5 je vrlo slična slici 3 – jedini dodaci su procesor vrhova i procesor fragmenata. Procesor vrhova zamjenjuje transformacije vrhova izvedene sklopovski ostvarenim funkcijama (ne može se koristiti oboje istovremeno). Dakle, sve transformacije se moraju izvesti na procesoru vrhova ili sklopovski ostvarenim funkcijama. Analogno vrijedi i za procesor fragmenata. Procesori vrhova i fragmenata mogu raditi nezavisno, odnosno, ako je jedan uključen, drugi ne mora biti.

Moguće su 4 kombinacije:

1. geometrijska faza – sklopovski ostvarene funkcije, rasterizacijska faza – sklopovski ostvarene funkcije,
2. geometrijska faza – procesor vrhova, rasterizacijska faza – sklopovski ostvarene funkcije,
3. geometrijska faza – procesor vrhova, rasterizacijska faza – procesor fragmenata,
4. geometrijska faza – sklopovski ostvarene funkcije, rasterizacijska faza – procesor fragmenata.

Valja uočiti da se dio grafičkih operacija i dalje isključivo izvodi pomoću sklopovski ostvarenih funkcija (dio sakupljanja primitiva i rasterizacije te interpolacije, kao i dio rasterskih operacija).

PROCESOR VRHOVA I PROCESOR FRAGMENATA

Shema procesiranja vrhova i fragmenata slična je, a procesiranje fragmenata prikazano je na slici 6. Procesor fragmenata nešto je složeniji zbog toga što je rasterizacijska faza složenija od geometrijske. Također broj slikovnih elemenata u rasterizacijskoj fazi mnogo je veći od broja vrhova u geometrijskoj fazi. Zbog toga moderne grafičke kartice imaju više procesora fragmenata od procesora vrhova. Tipični omjeri su 16/6 i 24/8. Procesori vrhova i procesori fragmenata vrlo su specifični procesori. Namijenjeni su za rad s vektorima i matricama (SIMD princip – *Single Instruction Multiple Data*). Zbog toga su dobri i za neke poslove koji nisu isključivo vezani uz računalnu grafiku, npr. fizikalne simulacije, obrada slike. Njihova konstrukcija nije pogodna izvođenju sasvim općenitih problema, već problema koji se mogu rješavati paralelizacijom.

Ulazni parametri vrhova/fragmenata primaju se preko ulaznih parametarskih registara, a rezultati transformacija se zapisuju u izlazne parametarske registre (transformira se jedan po jedan vrh/fragment neovisno o ostalima). Za pohranu međurezultata koriste se privremeni registri, a također postoji memorija za naredbe. Znači nije prisutan klasičan oblik pristupa RAM memoriji kao kod CPU. Kapacitet memorije za naredbe tipično iznosi 512 naredbi. Ukupan broj izvršenih naredbi je 65536 (128 iteracija po 512 naredbi).

Kako se u rasterizacijskoj fazi teksture koriste za bojanje, mora biti moguće koristiti mnogo tekstura odjednom. Procesori fragmenata mogu pristupiti do 16 tekstura. U geometrijskoj fazi teksture uglavnom služe samo za pohranu raznih koeficijenata, pa ih ne treba puno. Zbog toga procesori vrhova mogu pristupiti samo do 4 teksture.

Slika 7. Grafička kartica GeForce 8800 GTX

U arhitekturi GeForce 8800 GTX više nije prisutna podjela na procesore vrhova i fragmenata već je načinjeno 128 unificiranih procesora. Grafičke kartice (slika 7.) GeForce 8800 GTX imaju 768 MB GDDR3 memorije na 900 MHz, te memorijsku propusnost od 86 GB/s, te podršku za DirectX 10.

JEZICI ZA PROGRAMIRANJE GRAFIČKOG SKLOPOVLJA

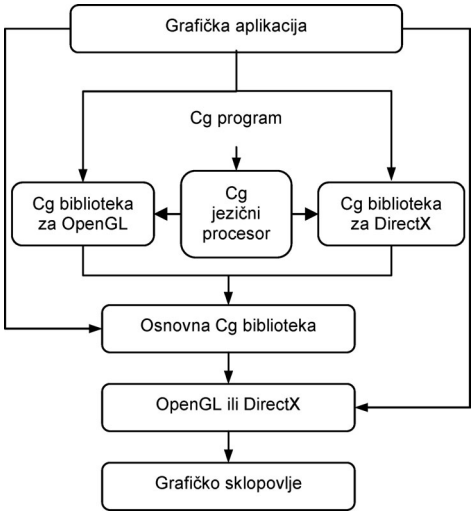
Iako programirljivo grafičko sklopovlje postoji tek nekoliko godina, za njega je razvijeno nekoliko programskih jezika. Prvi jezik koji se pojavio bio je, naravno, strojni jezik – asembler. Kako asembler inače nije baš popularan, vrlo brzo su se pojavili jezici više razine slični C-u.

Programirljive grafičke kartice prvih nekoliko generacija mogle su se programirati u asembleru, dok se neke novije ne mogu. Tvrtka NVIDIA i dalje ima potpunu podršku za asembler, dok tvrtka ATI podržava samo osnovni skup naredbi koje je definirao ARB, a pristup naprednijim mogućnostima dostupan je isključivo preko jezika više razine.

Trenutačno postoje 3 jezika više razine za programiranje grafičkog sklopovlja: HLSL, Cg i GLSL. Kako su svi jezici više razine nastali iz C-a, bit će opisane samo značajnije razlike. Tehnologija CUDA (engl. *Compute Unified Device Architecture*) je inovativna kombinacija računalnih mogućnosti grafičkog procesora kojima je omogućen pristup iz standardnog C-jezika u novim generacijama GPU-jedinica.

Programski jezici HLSL i Cg

Jezik HLSL (*High Level Shading Language*) proizvod je tvrtke *Microsoft*. Radi isključivo s Microsoft tehnologijama (DirectX na operacijskom sustavu Windows). Kako se većina profesionalnih grafičkih aplikacija koristi pod operacijskim sustavima UNIX/Linux, HLSL se ne može koristiti. Zbog toga je tvrtka NVIDIA preuzela sintaksu programskog jezika HLSL i napravila jezik Cg (*C for graphics*). Programski jezik Cg ne ovisi o operacijskom sustavu, tj. ima dinamičke biblioteke funkcija za svaki OS na kojem se koristi, a ne ovisi ni o programskom grafičkom sučelju, odnosno podržava DirectX i OpenGL. Princip rada prikazan je na slici 8. Kako je Cg nastao iz jezika C, svatko tko zna programski jezik C lako može naučiti Cg. Kroz nekoliko sljedećih karakteristika dat će se uvid u programski jezik Cg.



Slika 8. Programski jezik Cg

Tipovi podataka

Najvažnija razlika između C-a i Cg-a je ta što Cg nema pokazivače (što je vrlo logično, jer nema ni pristup RAM-u). Skalarni tipovi su: bool, int, unsigned int, float, double (trenutno isto što i float), half (16 bitni realni broj) i fixed (12 bitni realni broj). Tipovi half i fixed se koriste kada je potrebna velika preciznost, a mali raspon brojeva.

Vektorski tipovi se slažu od skalarnih kao vektori ili matrice. Na primjer, vektor je definiran kao skalarni_tip[X], gdje je X broj iz intervala 2 do 4. Odnosno konkretno int3 je vektor tri cijela broja. Slično je za matricu skalarni_tip[X]×[Y], X broj iz intervala 2 do 4, int3x4 cjelobrojna matrica.

Elementima vektora se pristupa skupom oznaka rgba, xyzw ili stpq (oznake iz različitih skupina se ne smiju miješati u istom izrazu uz istu varijablu).

Primjer:

```
float4 a = {1.0, 2.0, 3.0, 4.0};    // a = {1.0, 2.0, 3.0, 4.0}
float3 b = a.xyw;                   // b = {1.0, 2.0, 4.0}
a.zy = float2 (5.0, 6.0);           // a = {1.0, 6.0, 5.0, 4.0}
float2 c = b.xz + a.xy;             // c = {2.0, 10.0}
```

Iz primjera se može vidjeti da je rad s vektorima isti kao i sa skalarima, samo što se radi s dimenzijom 2 do 4 skalara paralelno, pa je i brzina izračuna isto toliko puta veća.

Također postoje i polja. Definiraju se na isti način kao u programskom jeziku C. Jedina razlika je u tome što moraju biti definirana statički, jer ne postoje funkcije za alokaciju i oslobađanje memorije. Isto tako, postoje i strukture koje se definiraju vrlo slično kao u programskom jeziku C. Podržani su gotovo svi operatori iz programskog jezika C. Jedino se matrice i vektori množe korištenjem ugrađene funkcije mul(), a ne operatora *. Osim funkcije mul(), postoji još mnogo ugrađenih funkcija kao što su: dot() – skalarni umnožak, sin() – sinus, log() – prirodni logaritam, all() – ispituje da li su sve komponente vektora različite od nule, any() – da li je barem jedna komponenta različita od nule, lerp() – linearna interpolacija, itd.

Funkcije

Preuzeto je svojstvo C++-a poznato kao preopterećivanje funkcija. To se odnosi na funkcije koje mogu imati isto ime, ali različite parametre ili isto ime, ali različite parametre i povratne tipove.

Primjer:

```
float2 funkcija(float2 a);
float2 funkcija (float3 a);
float3 funkcija (float4 a, float b);
```

Kako grafički procesori nemaju mogućnosti pristupa memoriji, a broj privremenih registara je ograničen, javlja se problem prijenosa velikog broja parametara u i iz funkcije. Da bi se riješio taj problem, preuzet je pristup iz jezika VHDL-a. Taj pristup koristi ključne riječi »in«, »out« i »inout« kod deklaracije varijabli on označava »smjer« djelovanja varijabli.

Primjer:

```
void funkcija2 (in float a, out int b, inout float4 c) {
c.y = a * 15;
b = c.w – 21;
...
}
```

Vezanje varijabli uz parametarske registre

Ovaj problem je sličan kao i problem prenošenja parametara u funkcije, samo što se odnosi na prijenos ulaznih i izlaznih parametara u programe koji se izvode na procesorima vrhova/fragmenata. Taj problem je riješen tako da se prilikom

deklaracije varijable navede dvotočka (:) i ime registra uz kojega se varijabla veže.

```
float4 main ( float4 polozaj : POSITION,
              float4 boja : COLOR) : POSITION {
              ...
}
```

Još neki od često korištenih registara: TEXCOORDn – koordinate teksture, NORMAL – normala, PSIZE – veličina točke.

Pristup teksturama

Teksturama se pristupa tako da se u glavnom programu veže tekstura (slika) uz teksturnu jedinicu grafičke kartice, u programu za sjenčanje odredi varijabla koja drži koordinate i na kraju koristi funkcija koja obavlja uzorkovanje.

Primjer:

```
float4 main ( float4 pozicija: POSITION, uniform
              sampler2D jedinica0,
              float2 koord : TEXCOORD0) : POSITION
{
              return tex2D(jedinica0, koord);
}
```

U gornjem odsječku su nepoznata dva pojma: uniform i sampler2D. Parametar uniform govori prevodiocu da se parametar ne mijenja tijekom izračuna cjelokupne scene, a ne samo tijekom izračuna jednog programa. Sampler2D je objekt pomoću kojega funkcija za uzorkovanje tex2D uzorkuje teksturu.

ZAKLJUČAK

U ovom radu dan je kratak pregled programirljive grafičke sklopovske opreme. Uz povijesni razvoj ukratko je objašnjen grafički protočni sustav, dane su glavne značajke procesora vrhova i procesora fragmenata, te osnove jezika za programiranje grafičkog sklopovlja uz nekoliko primjera za programski jezik Cg.

Grafička sklopovska oprema postala je izuzetno jako i moćno oružje, stoga je važno upoznati osnovne ideje i principe rada kako bi je mogli primijeniti i za rješavanje različitih drugih problema.

LITERATURA

[1] R. Fernando, M. J. Kilgard, **The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics.** Addison-Wesley, Boston, 2003.

[2] R. J. Rost, **OpenGL Shading Language.** Addison-Wesley Professional, second edition, 2006.

[3] D. Hearn, P. Baker, **Computer Graphics with OpenGL.** Pearson/Prentice-Hall, 2004.

dipl. ing. Krešimir JOZIĆ
prof. dr. sc. Željka MIHAJLOVIĆ