ŠIAULIŲ UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

INFORMATIKOS KATEDRA

Asta Margienė
Informatikos specialybės II kurso magistrantė

Dinaminių spalvinimo efektų kūrimo kalbos galimybių analizė ir taikymas trimatės grafikos sistemose

BAIGIAMASIS MAGISTRO DARBAS

Darbo vadovas:
Doc. V. Sirius

Recenzentas :
Lekt. L. Kaklauskas

Šiauliai, 2005/2006 m.m.

Turinys

1. Įvadas

Pastaraisiais metais interaktyvioji trimatė grafika yra viena iš greičiausiai besivystančių informacinių technologijų. Interaktyvioje trimatėje grafikoje vaizdo kūrimas vyksta realiu laiku ir priklauso nuo vartotojo veiksmų, todėl vaizdo kokybė ir realistiškumas labai priklauso nuo kompiuterio techninės ir programinės įrangos.

Atliekant scenos vizualizaciją, geometrinė ir tekstūrų informacija perduodama apdoroti grafinei plokštei. Anksčiau grafinės plokštės turėjo keletą algoritmų, kuriuos buvo galima panaudoti šių duomenų apdorojimui. Programuotojas galėjo pasirinkti fiksuotą funkciją ir nustatyti keletą plokštės būsenų, todėl buvo labai sunku sukurti unikaliai atrodančius žaidimus. Apie 2001 metus atsirado dinaminių spalvinimo efektų (*shader*) kūrimo priemonės, kurios suteikė programuotojams daugiau laisvės.

Spalvinimo efektas yra nedidelė programa, kuri matematiškai aprašo kaip atvaizduoti (*render*) kiekvieną objekto medžiagą (*material*) ir šviesų sąveikavimą su bendra visuma. Anksčiau spalvinimo efektus vykdavo centriniai procesoriai ir tik prieš keletą metų ATI ir nVidia kompanijos pagamino vaizdo plokštes su grafiniu procesoriumi, galinčias didžiule sparta atlikti šiuos spalvinimo efektus.

Iš pradžių spalvinimo efektai buvo rašomi žemo lygio asemblerio programavimo kalba, bet vėliau, siekiant palengvinti jų programavimą, buvo sukurtos specialios, aukšto lygmens programavimo kalbos. Kadangi šios kalbos yra palyginti naujos ir mažai žinomos, šiame darbe nutariau atlikti konkrečios spalvinimo efektų programavimo kalbos galimybių analizę ir jos panaudojimą trimatės grafikos sistemose.

Tikslai:

1. Atlikti pasirinktos dinaminių spalvinimo efektų programavimo kalbos galimybių analizę;
2. Išnagrinėti jos taikymą trimatės grafikos sistemose.

Uždaviniai:

1. Išsiaiškinti kas yra dinaminiai spalvinimo efektai ir kur jie naudojami;
2. Apžvelgti dinaminių spalvinimo efektų kūrimo priemones ir išanalizuoti jų taikymo galimybes;
3. Išsiaiškinti kokios dinaminių spalvinimo efektų programavimo kalbos šiuo metu yra populiariausios;
4. Atlikti pasirinktos dinaminių spalvinimo efektų programavimo kalbos galimybių analizę;
5. Atrinkti jau sukurtus pavyzdžius ir išnagrinėti jų veikimą.
6. Remiantis atlikta analize, sukurti savo spalvinimo efektą.

Praktinė reikšmė

Lietuvoje interaktyvioji kompiuterinė grafika nėra populiari, todėl mažai kas yra girdėjęs apie dinaminius spalvinimo efektus ir jų aukšto lygmens programavimo kalbas, sukurtas tik prieš keletą metų. Mano darbe yra surinkta ir išaiškinta svarbiausia informacija apie dinaminius spalvinimo efektus ir jų programavimui skirtas aukšto lygmens programavimo kalbas, o sukurtus spalvinimo efektus galima panaudoti 3D scenose.

2. Temos analizė

Kompiuterinės grafikos programavimui reikalingi įrankiai, leidžiantys įgyvendinti bet kokį programuotojo ar projektuotojo sumanymą. Galima naudoti asemblerį ir kiekvieną kartą rašyti savo operacinę sistemą (daugiausia laisvės - daugiausia pastangų ir žinių), arba pasinaudoti RAD aplinka ir specializuota taikomąja programavimo kalba su komponentų biblioteka (mažiausiai laisvės -mažiausiai pastangų ir žinių). Trečias kelias - pasinaudoti grafiniu API ir jo plėtiniais. [5]

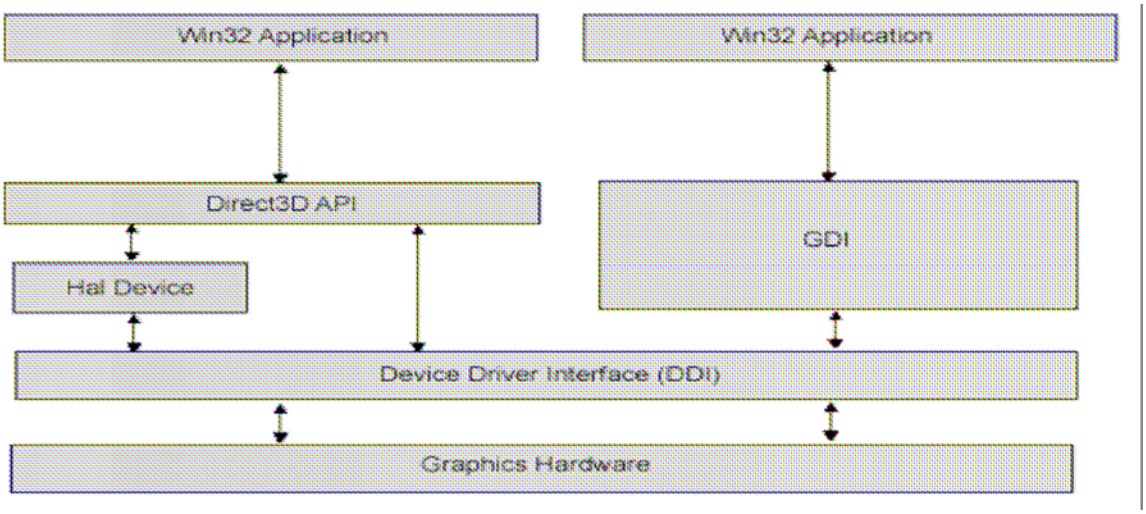
2.1. Trimatis grafinis API

Patobulėjus kompiuteriams (ypač - procesoriams) buvo pradėti kurti trimačiai grafiniai API. Kadangi trimatės grafikos principai gan griežti, šie API mažai skyrėsi vienas nuo kito. Jų sukurta buvo nedaug, o nuo 1992 metų, kai trimačių API rinkoje įsitvirtino SGI kompanijos sukurtas OpenGL, beveik visi neatlaikę konkurencijos buvo nustoti naudoti. Maždaug 1996-1997 metais OpenGL tapo trimačių grafinių API standartu *defacto* visoms kompiuterinėms platformoms ir operacinėms sistemoms. Vienintelė išimtis - Windows operacinėse sistemose naudojamas ir Microsoft aktyviai palaikomas DirectX.

„Microsoft" taip pat palaiko savo operacinėse sistemose OpenGL, tačiau Windows aplinkose OpenGL bibliotekos funkcijos veikia lėčiau nei kitose operacinėse sistemose ir lėčiau nei analogiškos Direct3D funkcijos. Panašu, kad tokių problemų naujose Windows versijose tik daugės - neseniai paskelbta, kad dėl kitokio OpenGL funkcijų realizavimo mechanizmo Longhorn'e jos veiks mažiausiai porą kartų lėčiau už DirectX.[5] Kadangi Lietuvoje populiariausia operacinė sistema yra Windows, o trimatės grafikos programavime ypač svarbus operacijų atlikimo laikas, todėl pasirinkau ne OpenGL, o Direct3D.

2.2. Direct3D vaizdo kūrimo sistemos integravimas

Direct3D vaizdo kūrimo sistema dabar aktyviai integruojama į Windows operacinių sistemų aplinkas. Siekiant padidinti vaizdo formavimo greitį, Direct3D gali veikti apeidamas operacinės sistemos tvarkyklių lygmenį (žr. schemą paimtą iš [11]).



Skirtingai nuo kitų Windows programų, Direct3D nesinaudoja grafinio įrenginio abstrakcijos, arba GDI (*Graphical Device Interface*), lygmeniu, tiesiogiai siųsdamas informaciją vaizdo plokštės tvarkyklei.[5]

2.2.1. HAL (*hardware abstraction layer*) įrenginys

Direct3D negali tiesiogiai kreiptis į grafinius įrenginius, kadangi yra labai daug skirtingų vaizdo plokščių, o kiekviena plokštė turi skirtingas galimybes ir jų įgyvendinimo būdus. Todėl, Direct3D pareikalavo grafinių įrankių gamintojų realizuoti HAL. HAL yra įrenginio specifinio kodo rinkinys, tiesiogiai perduodantis komandas ir duomenis vaizdo plokštės procesoriams, apeidamas standartines Windows tvarkykles. Tokiu būdu Direct3D nebereikia žinoti įrenginio specifinių savybių. Grafinių įrenginių gamintojai visas savo įrenginio ypatybes ir galimybes patalpina į HAL įrenginį. Savybės, kurias palaiko Direct3D, bet nepalaiko vaizdo plokštė, nėra realizuotos HAL įrenginyje, todėl Direct3D funkcijų, kurių nėra HAL įrenginyje, įvykdyti neįmanoma. Išskyrus trimačių transformacijų vykdymą (*vertex processing*), kuris tokiu atveju gali būti emuliuojamas programinėje įrangoje, panaudojant REF įrenginį.

2.2.2. REF įrenginys

Direct3D palaiko papildomą įrenginį vadinamą nuorodų (*reference*) įrenginiu arba nuorodų apdorojimo (*reference rasterizer*) (REF). REF palaiko visas Direct3D savybes (*feature*). Kadangi šios savybės yra atliekamos tiksliai ir kruopščiai, be to jos realizuojamos programinėje įrangoje, todėl jų vykdymas užtrunka. Dėl šio trūkumo, REF įrenginys galutinio produkto naudojimui netinka, o yra naudojamas tik programų testavimui ar demonstravimui.

2.3. Spalvinimo efektai (*shader*)

Spalvinimo efektai (*shader*) yra trumpos programos, vykdomos grafinių plokščių procesorių (*GPU*) ir pakeičiančios dalį fiksuotų grafinio konvejerio funkcijų. Pakeitus konvejerio fiksuotas funkcijas sukurtais spalvinimo efektais, atsiranda didžiulės įvairių grafinių efektų sukūrimo galimybės, o programuotojas yra išlaisvinamas nuo apribojimų, naudojant fiksuotas operacijas. Šiuo metu spalvinimo efektai yra dviejų tipų: viršūnių spalvinimo efektai (*vertex shader*) ir pikselių spalvinimo efektai (*pixel shader*). *(Platesnė jų apžvalga yra „Viršūnių salvinimo efektai“ ir „Pikselių spalvinimo efektai“ dalyse).* Tačiau Microsoft jau skelbia, kad šiuo metu kuriamoje Direct3D 10 bibliotekoje bus jau trijų tipų spalvinimo efektai: viršūnių (*vertex shader*), geometrijos (*geometry shader*) ir pikselių (*pixel shader*) spalvinimo efektai. Tai dar nėra įgyvendinta, todėl šiame darbe yra nagrinėjami tik viršūnių ir pikselių spalvinimo efektai. *(Plačiau pasiskaityti apie geometrinius spalvinimo efektus jau galima Direct3D 10 dokumentacijoje).*

Siekiant lengviau suvokti spalvinimo efektų panaudojimo galimybes reikia panagrinėti kaip yra formuojami ir apdorojami trimačiai objektai, bei pažvelgti į spalvinimo efektų padėtį Direct3D grafiniame konvejeryje.

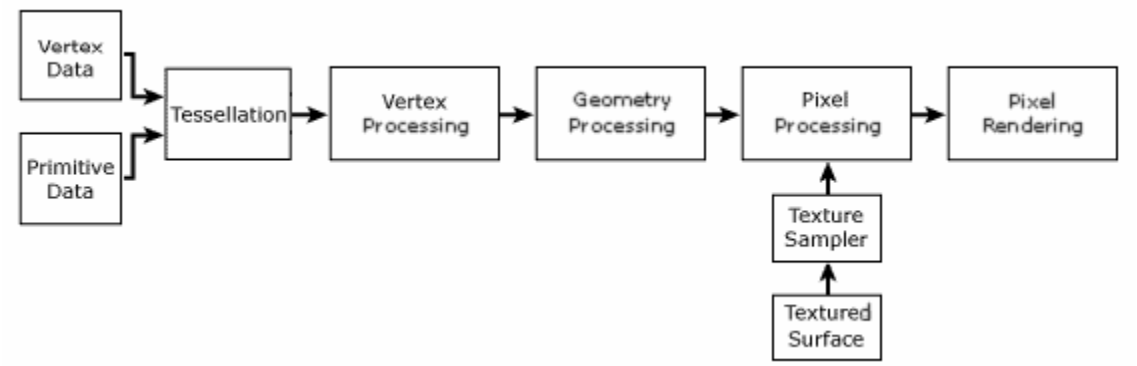
2.4. Grafinis konvejeris

Visi realaus laiko sistemos trimačiai objektai (tiek žaidimų, tiek virtualios realybės) yra sudaryti iš šimtų ar tūkstančių daugiakampių. 3D grafikos plokštės darbas yra, piešiant spalvotus trikampius ir jų kombinacijas, atvaizduoti trimačius objektus ekrane. Jos tai atlieka labai greitai – milijonai trikampių per sekundę tam, kad atvaizduotų šimtus objektų, susidedančių iš tūkstančių

trikampių ir dar daugiau nei 30 kartų per sekundę (jei norima, kad veiksmas vyktų realiame laike, tai 60 ir daugiau kartų per sekundę)[16].

Pagrindinis procesorius siunčia trikampio viršūnių koordinačių sąrašą vaizdo plokštei, kuri duomenis apdoroja konvejeriu.

Grafinis konvejeris (*graphics pipeline*) yra skirtas efektyviai apdoroti ir vizualizuoti (*render*) Direct3D scenas ekrane, panaudojant techninės įrangos privalumus. Deja, Direct3D grafinis konvejeris yra vienaip suvokiamas vartotojų-programuotojų, o kitaip grafinio konvejerio realizuotojų. Todėl šiame darbe man tenka panagrinėti abu grafinio konvejerio atlikimo aiškinimo variantus. Kadangi šioje dalyje, siekiama tik parodyti spalvinimo efektų vietą Direct3D grafiniame konvejeryje, todėl čia tinka vartotojams – programuotojams skirta grafinio konvejerio schema, o gilesnis grafinio konvejerio nagrinėjimas yra atliktas „Viršūnių spalvinimo efektai“ ir „Pikselių spalvinimo efektai“ dalyse. Šioje schemoje yra pavaizduoti pagrindiniai konvejerio etapai:



Modelio duomenų išsaugojimas (vertex data). Modelinio vaizdo viršūnių neapdoroti duomenys išsaugomi specialiuose atminties masyvuose (*vertex memory buffers*) tolesniam jų apdorojimui.

Geometrinių primityvų apdorojimas (primitive data). Specialiuose primityvų sąrašų buferiuose (*index buffers*) surenkama informacija apie vaizdo primityvus – taškus, atkarpas, trikampius ir daugiakampius.

Aukštesnio lygio primityvų kūrimas (tessellation). Speciali programa (*tesselator*) perkelia į viršūnių buferius informaciją apie aukštesnio lygmens primityvus, perkėlimo masyvus ir paviršių dalis.

Trimačių transformacijų vykdymas (vertex processing). Atliekamos trimatės transformacijos su duomenimis, saugomais viršūnių buferiuose. Modelių duomenys yra pervedami iš vienos koordinačių sistemos erdvės į kitą. Šis etapas gali būti atliekamas keliais būdais:

1. Grafinio konvejerio fiksuotomis funkcijomis (*fixed-function*). Tai yra standartinis trimačių transformacijų vykdymo (*vertex processing*) etapo atlikimas, panaudojant standartinius Direct3D metodus, nustatant apšvietimo ir materialų parametrus, vizualizavimo būsenas (*render states*).
2. Viršūnių spalvinimo efektais (*vertex shaders*). Vietoj parametrų nustatymo yra rašoma viršūnių spalvinimo efekto programa, kurią vykdo grafinės plokštės GPU.

(Plačiau šis grafinio konvejerio etapas aprašytas "Viršūnių spalvinimo efektai" dalyje.)

Geometrinio vaizdo parametrų skaičiavimas (geometry processing). Transformuotoms viršūnėms atliekamos nematomų dalių iškirpimo (*clipping* ir *culling*), atributų skaičiavimo ir rasterizacijos operacijos.

Tekstūrų koordinačių skaičiavimas (texture surface). Per IDirect3DTexture9 aplinką pateikiamos trimačių objektų tekstūrų koordinatės ir susiejamos su viršūnėmis.

Tekstūrų dalių apdorojimas (texture sampler). Jei reikia, atliekamos atskirų tekstūrų dalių papildomos apdorojimo operacijos.

Pikselių apdorojimas, pritaikant spalvinimo modelius (pixel processing). Iš saugomos viršūnių bei tekstūrų informacijos skaičiuojamos pikselių spalvų reikšmės, pritaikant pikselių spalvinimo modelius ir procedūras. *(Plačiau šis grafinio konvejerio etapas aprašytas "Pikselių spalvinimo efektai" dalyje.)*

Galutinio vaizdo kūrimo pabaiga (pixel rendering). Atliekami alfa, gylio ar trafareto testai, skaičiuojami spalvų maišymo parametrai, imituojamas rūkas ir kt. Galutinai suformuojamas kadro buferis.

3. Darbo srities analizė

3.1. Spalvinimo efektų (*shader*) programavimo kalbos

Istoriškai, grafinė techninė įranga (*hardware*) buvo programuojama žemame lygyje ir dar visai neseniai programuotojai konfigūravo programinius konvejerius asemblerio kalba, naudodami programines sąsajas (*interfaces)*. Teoriškai šios žemo lygio programavimo sąsajos aprūpintos didžiuliu lankstumu. Praktiškai - jas buvo sunku panaudoti, todėl jos tapo rimtu barjeru efektyviam techninės įrangos naudojimui.

Stipriai patobulėjus vaizdo plokštėms, atsirado galimybė trumpas viršūnių ir fragmentų programas vykdyti jų grafiniuose procesoriuose. O dabar, kai grafiniai procesoriai jau gali vykdyti programas, turinčias šimtus ar tūkstančius instrukcijų, programavimas asembleriu tapo nebepatogus. Tam buvo sukurtos naujos, panašios į C, aukšto lygmens programavimo kalbos.

Visose skaitytose knygose bei internete teigiama, kad šiuo metu spalvinimo efektų programavimui yra skirtos trys pagrindinės programavimo kalbos: OpenGL‘o – GLSL (GL Shader Language), Microsoft‘o – HLSL (High-Level Shading Language) ir nVidia’os – Cg (C for graphics). Tačiau nVidia dokumentacijoje radau, kad „bendradarbiaujant nVidia ir Microsoft kompanijoms buvo sukurta Cg spalvinimo efektų programavimo kalba, kurios realizaciją Microsoft‘as vadina HLSL, o NVIDIA - Cg. HLSL ir Cg yra ta pati programavimo kalba literatūroje nurodoma skirtingais, kalbos identifikavimui ir jos technologijoms atskirti naudojamais, pavadinimais.“ Trumpiau tariant, Cg ir HLSL yra tos pačios programavimo kalbos dvi skirtingos realizacijas. HLSL yra Microsoft’o Direct3D dalis. Cg yra nepriklausoma nuo 3D programavimo vartotojo sąsajos ir pilnai integruota į Direct3D ir OpenGL. Tinkamai parašytas spalvinimo efektas gali veikti ir su OpenGL, ir su Direct3D. Šis lankstumas reiškia, kad nVidia’os Cg realizacija suteikia programuotojams galimybę dirbti su abiem dominuojančiomis 3D programinėmis sąsajomis (*3D programming interfaces*) ir pasirinkta operacine sistema, kadangi Cg yra realizuota su Windows, Linux, Mac OS. Cg programos yra vykdomos pagrindinių vaizdo plokščių gamintojų (3Dlabs, ATI, Matrox, ir NVIDIA) sukurtuose grafiniuose procesoriuose (taip rašoma nVidia SDK dokumentacijoje, tačiau iš tiesų naudojant kitų kompanijų sukurtas grafines plokštes gali atsirasti įvairių nesklandumų, todėl Cg realizavimas labai padidino nVidia vaizdo plokščių paklausą).

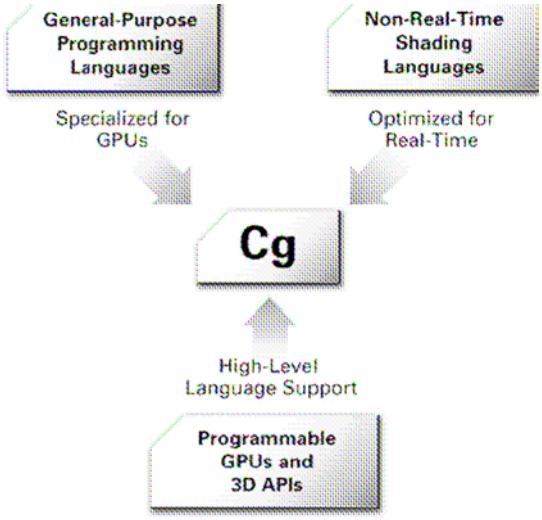
3.1.1. Aukšto lygmens dinaminių spalvinimo efektų programavimo kalbos privalumai

Aukšto lygmens programavimo kalbos panaudojimas spalvinimo efektų rašymui turi tokius privalumus:

- Padidėja našumas (produktyvumas) – rašyti programas aukšto lygio programavimo kalba yra nepalyginamai greičiau.
- Palengvėja programų skaitymas – programos, parašytos aukšto lygio programavimo kalba yra lengviau skaitomos, o tai reiškia, kad jos yra lengviau derinamos (*debug*) ir aptarnaujamos (*maintain*).
- Kompiliatoriai dažniausiai iš aukšto lygmens kalbos į asemblerį sukompiliuotą kodą generuoja daug efektyviau nei ranka parašytą asemblerio kodą.
- Naudojant aukšto lygmens kalbos kompiliatorių, galima kompiliuoti savo parašytą kodą į bet kurią tinkamą spalvinimo efekto versiją. Naudojant asemblerio kalbą, reikia prijungti kodą kiekvienai reikiamai versijai.

3.2. Cg-HLSL programavimo kalba

Cg-HLSL yra aukšto lygmens programavimo kalba, sukurta C programavimo kalbos pagrindu, bei turinti nemažai neinteraktyvių spalvinimo efektų programavimo kalbų savybių (pvz., RenderMan) ir papildomų galimybių, kurios padeda lengvai parašyti grafiniame procesoriuje vykdomas programas. nVidia kompanijos dokumentacijoje radau schemą, apibūdinančią Cg - HLSL kalbą:



Cg-HLSL kodas atrodo beveik taip pat, kaip ir C kalbos kodas (ta pati kintamųjų aprašymo sintaksė, funkcijų iškvietimas, ciklai, sąlyginiai sakiniai, naudojami masyvai ir struktūros, yra daug tokių pat duomenų tipų ir t.t.).

Savaime suprantama, kad tarp šių kalbų yra ir skirtumų. Visų pirma Cg-HLSL (kaip ir kitos spalvinimo efektų kalbos) yra paremta (*based on*) duomenų srauto skaičiavimo modeliu. Antra, tai yra labai specializuota programavimo kalba, todėl niekur nepamatysite Cg-HLSL kalba sukurtos skaičiuoklės ar tekstų rengimo programos. Dabartinė Cg-HLSL programavimo kalba nepalaiko rodyklių ir atminties paskirstymo (*memory allocation*), failų įvesties/išvesties operacijų ir kt. Tačiau šie apribojimai nėra ilgalaikiai, nes jie atsirado tik dėl nepakankamų grafinių plokščių galimybių. Grafinei techninei įrangai dar patobulėjus, bus galima laukti atitinkamo Cg-HLSL programavimo kalbos augimo. Cg natūraliai palaiko vektorius ir matricas, kadangi šie duomenų tipai ir juos naudojančios matematinės operacijos grafikoje yra labai svarbios ir daugelis grafikos techninės įrangos tiesiogiai palaiko vektorių duomenų tipus.

Cg-HLSL spalvinimo efektams yra reikalingi 3D modeliai, tekstūros ir kt. duomenys, todėl spalvinimo efektų paleidimui yra reikalingos programos, parašytos centriniam procesoriui tradicinėmis kalbomis (C, C++ ir kt.). Šios programos informuoja GPU kaip atlikti programinius vizualizavimo efektus.

Norint geriau suvokti Cg-HLSL kalbos panaudojimą, reikia panagrinėti GPU programavimo modelį.

3.2.1. GPU programavimo modelis

GPU yra specializuoti, todėl jie yra nepalyginamai greitesni už CPU, kai reikia atlikti grafikos užduotis (pvz., 3D scenų vizualizacija). Dabartiniai grafiniai procesoriai apdoroja dešimtis milijonų viršūnių per sekundę ir rasterizuoja (*rasterize*) milijardus pikselių per sekundę, o ateityje jie bus dar greitesni.

CPU paprastai turi vieną programinį procesorių, tuo tarpu GPU – mažiausiai du: viršūnių ir fragmentų, bei kt. neprogramininius techninės įrangos elementus. Procesoriai, grafinės techninės įrangos neprogramininė dalis ir programa yra susieti per duomenų srautus (*data flows*). Cg-HLSL kalba yra rašomos programos viršūnių ir fragmentų procesoriams. Atitinkamai šios programos ir vadinamos viršūnių ir fragmentų programomis arba viršūnių ir pikselių spalvinimo efektais (*vertex shader* ir *pixel shader*). Cg kodas yra kompiliuojamas į GPU asemblerio kodą paleidimo metu, arba iš anksto. Cg-HLSL leidžia lengvai kombinuoti Cg-HLSL pikselių spalvinimo efektų programą, su ranka parašyta viršūnių spalvinimo efektų programa arba su neprogramuojamu OpenGL ar DirectX (priklausomai nuo realizacijos) viršūnių konvejeriu ir atvirkščiai, Cg-HLSL viršūnių spalvinimo efekto programą galima kombinuoti su ranka parašytu pikselių spalvinimo efektu ar neprogramuojamu OpenGL ar DirectX pikselių konvejeriu.[9]

GPU programiniai procesoriai valdo (*operate*) duomenų srautus: viršūnių procesorius valdo viršūnių srautus, fragmentų – fragmentų (pikselių) srautus. Programos, vykdomos grafiniame procesoriuje yra atliekamos po viena kartą kiekvienam duomenų srauto elementui t.y. viršūnių spalvinimo efektas yra vykdomas kiekvienai viršūnei, o pikselių spalvinimo efektas – kiekvienam pikseliui.

3.2.2. Spalvinimo efektų galimybių apribojimas

Programa, parašyta kuria nors centrinio procesoriaus (CPU) programų kūrimui skirta kalba, yra kompiliuojama ir vykdoma teisingai, nepriklausomai nuo to kokios kompanijos sukurtas CPU ją vykdo. Taip yra todėl, kad CPU yra sukurti taip, kad visi turi pakankamai reikiamų resursų. Visiškai kitokia situacija šiuo metu yra su grafiniais procesoriais, nes šiuo metu gaminami GPU yra skirtingų galimybių, todėl ne visi, Cg-HLSL kalba parašyti spalvinimo efektai, gali būti vykdomi konkrečiame grafiniame procesoriuje. Siekiant, kad Cg-HLSL butų palaikoma daugelio grafinių procesorių, HLSL realizacijoje buvo sukurtos versijos, o Cg realizacijoje – profiliai (*profile*s). Cg profilis ar HLSL versija apibrėžia Cg programavimo kalbos poaibį, kurį palaiko konkreti techninė įranga ar API (Cg realizacijoje). Todėl spalvinimo efektas turi būti ne tik teisingai parašytas, bet ir parinkta tinkama versija (profilis) jo kompiliavimui.

Ateityje, tobulėjant grafinei techninei įrangai, toks skirstymas į versijas (profilius) taps nebe toks ryškus, o gal ir visai išnyks. Tačiau dabar programuotojai turi apriboti spalvinimo efektų panaudojimo galimybes tam, kad juos būtų galima kompiliuoti ir vykdyti šiuo metu naudojamuose GPU.

Kuriant spalvinimo efektus reikia nepamiršti, kad kuo programa mažesnė, tuo greičiau ji bus atlikta, o tai yra labai svarbu realaus laiko grafikoje, nes norint, kad animacija būtų vykdoma realiame laike, reikia, kad per vieną sekundę būtų perpiešiama 60 ar daugiau kadrų, o kompiuterio displėjus gali turėti milijoną ir daugiau pikselių. Pavyzdžiui, jeigu jūsų ekrano

rezoliucija yra 1024 x 768, tai pikselių ekrane yra 1024 * 768 = 786 432, o per vieną sekundę pikselių spalvinimo efektas turės būti įvykdytas apie 786 432 * 60 = 47 185 920 kartų. Be to reikia nepamiršti, kad 3D modeliai yra sudaryti iš viršūnių, kurias reikia teisingai transformuoti, todėl per vieną sekundę dar gali tekti transformuoti ir dešimtis milijonų viršūnių.

.

3.3 Cg realizacija

3.3.1. Cg profiliai (*profiles*)

Šiuo metu Cg kompiliatorius palaiko tokius profilius [13]:

Viršūnių spalvinimo efektų profiliai		**Pikselių spalvinimo efektų profiliai**	
vp20, vp30, vp40	*OpenGL NV_vertex_program*	fp30, fp40	*OpenGL NV_fragment_program*
arbvp1	*OpenGL ARB_vertex_program*	fp20	*NV_register_combiners and NV_texture_shader*
glslv	*GLSL Vertex Shader*	arbfp1	*OpenGL ARB_fragment_program*
vs_1_1	*DirectX 8 Vertex Shaders 1.1*	glslf	*GLSL Fragment Shader*
vs_2_0, vs_2_x	*DirectX 9 Vertex Shaders 2.0, 2.x*	ps_1_1, ps_1_2, ps_1_3	*DirectX 8 Pixel Shaders 1.1 - 1.3*
		ps_2_0, ps_2_x	*DirectX 9 Pixel Shaders 2.0, 2.x*
vs_3_0	*DX9 Shader Model 3 Vertex Shader*	ps_3_0	*DX9 Shader Model 3 Pixel Shader*

3.3.2. Cg spalvinimo efektų kompiliavimas

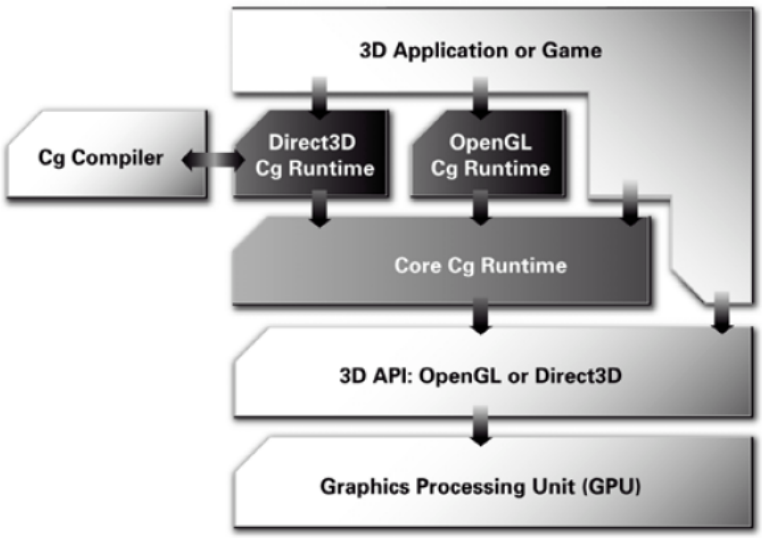
Nė vienas GPU negali tiesiogiai vykdyti Cg spalvinimo efektų, todėl jie yra kompiliuojami. Cg palaiko statinį (kompiliatorius programą iš karto kompiliuoja į tokį kodą, kuris tiesiogiai yra vykdomas GPU) ir dinaminį kompiliavimą, kuris leidžia optimizuoti Cg spalvinimo efektus konkrečiam GPU.

Iš pradžių kompiliatorius transliuoja Cg spalvinimo efektus į 3D programinei vartotojo sąsajai (OpenGl arba DirectX) tinkamą formą, o tada programa, panaudodama reikiamas OpenGL ar DirectX komandas, perduoda Cg spalvinimo efektų vertimą į GPU. OpenGl ar Direct3D tvarkyklė atlieka paskutinį transliavimą, po kurio GPU jau gali vykdyti spalvinimo efektus. Daugelis kompiliavimo detalių priklauso nuo naudojamo GPU, nes jis gali nepalaikyti reikiamų savybių (pvz., Cg pikselių spalvinimo efektas nebus kompiliuojamas jeigu bus naudojama daugiau tekstūrų, nei palaiko GPU).[12]

Cg kompiliatorius yra ne savarankiška programa, o *Cg runtime* bibliotekos dalis. (Cg runtime yra standartinis paprogramių rinkinys, naudojamas Cg programų užkrovimui, kompiliavimui, valdymui ir konfigūravimui.) Programos, naudojančios Cg, iškviečia *Cg runtime* paprogrames Cg spalvinimo efektų kompiliavimui. *Cg runtime* branduolyje yra dvi glaudžiai susijusios bibliotekos: CgGL ir CgD3D. CgGL biblioteka naudojama, kai reikia iškviesti atitinkamas OpenGL paprogrames, kurios transliuotą Cg spalvinimo efekto kodą perduotų į OpenGL tvarkyklę. CgD3D - kai reikia iškviesti atitinkamas Direct3D paprogrames, kurios transliuotą Cg spalvinimo efekto kodą perduotų į Direct3D tvarkyklę. Paprastai yra naudojama

arba CgGL , arba CgD3D biblioteka, bet ne abi iš karto, nes daugelis programų naudoja arba OpenGl arba Direct3D [12].

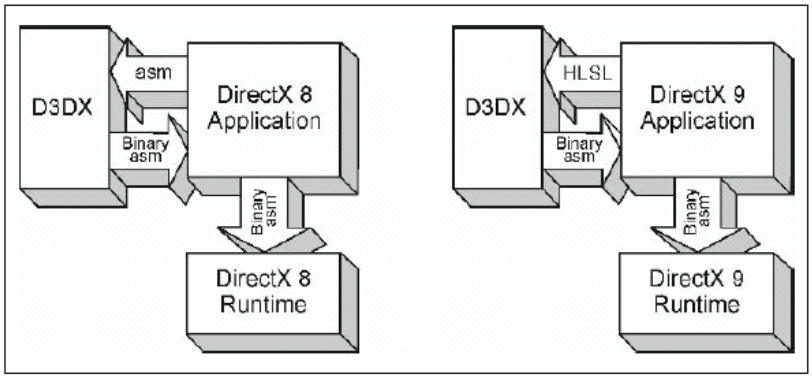
Paveikslėlyje (paimtame iš [12] šaltinio) yra pavaizduota kaip 3D programa naudoja Cg bibliotekas.



3.4. HLSL spalvinimo efektų kompiliavimas

HLSL kalba yra susieta su Direct3D, D3DX, asemblerio kalba parašyto spalvinimo efekto modeliu ir programa. Pirmą kartą spalvinimo efektai buvo panaudoti Direct3D DirectX 8.0 versijoje. DirectX 8.0 ir DirectX 8.1 naudojami vs_1_1 ir ps_1_1, ps_1_2, ps_1_3, ps_1_4 spalvinimo efektų modeliai buvo palyginti trumpi ir rašomi asemblerio kalba. Kaip parodyta paveikslėlio kairėje pusėje, programa perduoda asemblerio kalba parašytą kodą D3DX bibliotekai per D3DXAssembleShader() ir gauna dvejetainį spalvinimo kodą, kurį perduoda Direct3D per CreatePixelShader() arba CreateVertexShader().

Pažvelgę į paveiksliuko [8] dešinę pusę, matome, kad situacija labai panaši ir su DirectX 9, tik programa D3DX bibliotekai per D3DXCompileShader() API perduoda HLSL kalba parašytą spalvinimo efekto kodą ir gauna atgal sukompiliuoto spalvinimo efekto dvejetainį kodą, kurį perduoda į Direct3D per CreatePixelShader() arba CreateVertexShader(). Generuojamas dvejetainis asemblerio kodas nepriklauso nuo gamintojų ir yra atliekamas lygiai taip pat, nepriklausomai nuo to kur jis bus kompiliuojamas ar paleistas. Funkcijų atlikimo metu Direct3D nieko nežino apie HLSL, o tik apie dvejetainius asemblerio spalvinimo efektų modelius t.y. HLSL kompiliatorius gali būti atnaujintas nepriklausomai nuo Direct3D.



Be to HLSL kompiliatoriaus tobulinimui D3DX bibliotekoje, DirectX 9 pateikė papildomus asemblerio spalvinimo efektų modelius (*models*) naujos kartos 3D grafinių plokščių funkcionalumui (*functionality*) panaudoti: vs_2_0, vs_3_0, ps_2_0, ir ps_3_0, o kuriamame Direct3D 10, jau numatyti ir vs_4_0, ps_4_0 ir gs_4_0.

HLSL spalvinimo efekto kompiliavimo nutraukimas yra požymis, kad jis yra per daug sudėtingas parinktai realizacijos versijai (*compile target*) . Tai reiškia, kad arba spalvinimo efektas reikalauja per daug resursų, arba jis reikalauja kokių nors galimybių (*capability*)(pvz., dinaminio išsišakojimo, kurio pasirinkta realizacijos versija (*compile target*) nepalaiko). Pavyzdžiui, HLSL spalvinimo efekto algoritme reikia kreiptis į nurodytą tekstūrą (*texture map*) šešis kartus. Jeigu šis spalvinimo efektas yra kompiliuojamas su ps_1_1 realizacijos versija (*compile target*), kompiliavimas bus nutrauktas, nes ps_1_1 versija palaiko tik keturias tekstūras. Kitas dažnai pasitaikantis kompiliavimo nutraukimo atvejis yra viršytas instrukcijų skaičius. HLSL spalvinimo efekto algoritmui atlikti gali prireikti daugiau instrukcijų, nei parinkta realizacijos versija (compile target) gali atlikti. Svarbu pastebėti, kad realizacijos versijos pasirinkimas nevaržo HLSL sintaksės. Pavyzdžiui, spalvinimo efektų programuotojas gali naudoti for ciklus, paprogrames, if-else sakinius ir t.t. ir jeigu pasirinktos realizacijos versija paprastai nepalaiko ciklų, išsišakojimo ar if-else sakinių, tada kompiliatorius išardo ciklą ir atlieka visas reikiamas operacijas pavieniui , suderina funkcijų iškvietimus (*inline function calls*) ir įvykdo abi if-else sakinio atšakas, išrinkdamas tinkamus rezultatus, paremtus originalia reikšme (*value*), panaudota if-else sakinyje. Žinoma jeigu spalvinimo efekto rezultatas bus per ilgas arba viršys realizacijos versijos resursus, kompiliavimas bus nutrauktas.[8]

3.4.1. Vaizdo plokštės

Kaip jau buvo minėta, HLSL kalba parašyto, nors ir elementaraus spalvinimo efekto vykdymas priklauso nuo jūsų kompiuteryje esančios techninės įrangos, o tiksliau nuo video plokštės. Programa iškviečia D3DX biblioteką sukompiliuoti HLSL spalvinimo efektą į dvejetainį kodą per D3DXCompileShader() API. Vienas iš šios API pagrindinių parametrų yra parametras, nurodantis kurį iš asemblerio kalbos modelių ar realizacijos versijų HLSL kompiliatorius turi naudoti galutiniam spalvinimo efekto kodui išreikšti. Jei programa atlieka HLSL spalvinimo efekto kompiliavimą realiu laiku, ji turi patikrinti Direct3D įrenginio (device) pajėgumus ir parinkti tinkamą realizacijos versiją (*compile target*). Jei HLSL spalvinimo efekto algoritmas yra per sudėtingas pasirinktos realizacijos versijai (*compile target)*, kompiliavimas bus nutrauktas.[8] Tai reiškia, kad nors HLSL yra labai naudinga spalvinimo efektų plėtrai, tačiau ji neišlaisvina vartotojų nuo priklausomybės nuo grafinių plokščių galimybių. Todėl žaidimų kūrėjai vis dar turi naudoti HLSL realizacijos versijas, rašydami sudėtingesnius spalvinimo efektus geresnėms plokštėms ir elementaresnius – senesnėms vaizdo plokštėms.

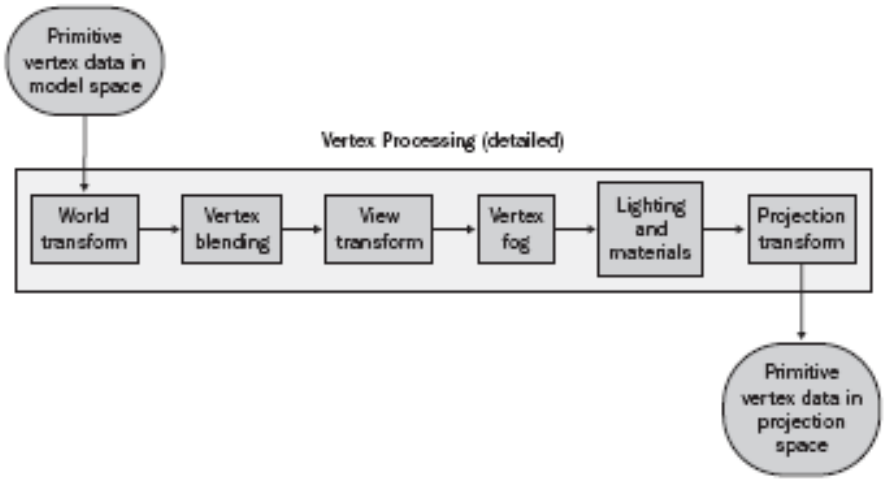
Prieš pradedant nagrinėti spalvinimo efektus, pirmiausia reikia susipažinti su vaizdo plokštėmis, palaikančiomis viršūnių ir pikselių spalvinimo efektus. Šiuo metu yra trys pagrindiniai vaizdo plokščių, palaikančių spalvinimo efektus, gamintojai: nVidia, ATI, ir 3Dlabs [13],[14],[15]. Sekančioje lentelėje yra pateiktos vaizdo plokštės ir spalvinimo efektų versijos, kurias jos palaiko:

Gamintojas	***Pavadinimas***	***VS versija***	***PS versija***
nVidia	GeForce2	--- 3.0sw	--- 3.0sw
	GeForce3–Ti	1.1	1.3
	Xbox	1.0	1.0x
	GeForce4–MX (NV18)	1.1	---
	GeForce4–Ti (NV28)	1.1	1.3
	GeForceFX (NV30)	2.0+	2.0+
	GeForce 6 serija	3.0	3.0
	GeForce 7 serija	3.0	3.0
ATI	Radeon 8500/9100 (RV200)	1.1	1.4
	Radeon 9000 (RV250)	1.1	1.0-1.4
	Radeon 9200 (RV280)	1.1	1.0-1.4
	Radeon 9500/9700 (RV300)	1.1, 2.0	1.4, 2.0
	Radeon 9600 (RV350)	2.0	2.0
	Radeon 9800 (R350)	2.0	2.0
	Radeon® X300 Graphics Technology	2.0	2.0
	Radeon® X550 Graphics Technology	2.0	2.0
	Radeon® X600 Graphics Technology	2.0	2.0
	Radeon® X700 Graphics Technology	2.0+	2.0+
	Radeon® X800 Graphics Technology	2.0+	2.0+
	Radeon® X850 Graphics Technology	2.0+	2.0+
	Radeon® X1300 serija	3.0	3.0
	Radeon® X1600 serija	3.0	3.0
	Radeon® X1800 serija	3.0	3.0
	Radeon® X1900 serija	3.0	3.0
3Dlabs	Wildcat VP 560	1.1	1.2
	Wildcat VP 760	1.1	1.2
	Wildcat VP 880 Pro	1.1	1.2
	Wildcat VP 990	1.1	1.2
	Wildcat Realizm 100	2.0	3.0
	Wildcat Realizm 200	2.0	3.0
	Wildcat Realizm 500	2.0	2.0
	Wildcat Realizm 800	2.0	3.0

Renkantis vaizdo plokštę svarbu žinoti, kad viršūnių spalvinimo efektai gali būti emuliuojami programinėje įrangoje, todėl neturint daug pinigų galima pasirinkti pigesnę plokštę, kuri nepalaiko reikiamo standarto viršūnių spalvinimo efektų, tačiau turi greitą programinės įrangos emuliatorių. Nors emuliatorius vykdys viršūnių spalvinimo efektus daug lėčiau, tačiau jų algoritmus galėsite patikrinti.

3.4.2. Viršūnių spalvinimo efektai

Viršūnių spalvinimo efektas (vertex shader) yra programa, vykdoma vaizdo plokštės GPU, kuri pakeičia fiksuotus transformacijos ir apšvietimo etapus grafiniame konvejeryje.(Iš tiesų tai nėra visiška tiesa, kadangi viršūnių spalvinimo efektai Direct3D veikimo metu (runtime) gali būti emuliuojami programinės įrangos (jeigu techninė įranga nepalaiko viršūnių spalvinimo efektų.)) Paveikslėlyje [4] yra pavaizduota grafinio konvejerio trimačių transformacijų vykdymo (*vertex processing*) dalis, kurią galima pakeisti viršūnių spalvinimo efektu.



Bendroji transformacija (World Transform). Transformuoja viršūnių duomenis (*vertex data*) iš modelio erdvės į bendrąją erdvę (*world space*). Yra nustatomas objektų išsidėstymas ir pasisukimas vienas kito atžvilgiu bendroje erdvėje.

Viršūnių maišymas (Vertex Blending). Maišymo (*blending*) metu yra atliekamas vieno ar kelių viršūnių rinkinių maišymas (*combine*) viršūnių duomenų animacijai atlikti. Tam yra naudojamas vienas ar keli viršūnių duomenų rinkiniai ir vienas ar keletas pertvarkymų. (kiekvienas su atitinkamu svoriu). Viršūnių duomenims yra atliekami visi pertvarkymai, o rezultatai yra derinami pagal tų pertvarkymų svorį.

Peržiūros transformacija (View Transform). Perveda viršūnių duomenis iš bendrosios erdvės į peržiūros erdvę. Kamera yra peržiūros erdvės pradiniame taške.

Viršūnių rūkas (Vertex Fog). Kiekvienai viršūnei skaičiuojama rūko spalva**.** Viršūnės rūko spalva yra maišoma (derinama) su kitomis viršūnės (per-vertex) spalvomis (pvz., vartotojo parinkta spalva, apšvietimu ir materialo spalvomis) ir taip prieš *geometry processing* etapą yra gaunama galutinė kiekvienos viršūnės spalva.

Apšvietimas ir medžiagos (Lighting and Materials). Atsižvelgiant į apšvietimą ir medžiagas (*materials*) yra skaičiuojama kiekvienos viršūnės (*per-vertex*) spalva. Apšvietimas yra skaičiuojamas nuo aktyvių scenos šviesos šaltinių. Medžiagos spalvos yra nustatomos kaip konstantos. Medžiagų (*materials*) ir šviesų derinimas sukuria tikrovišką vaizdą.

***Projekcijos transformacija (Projection Transform*).** Transformuoja viršūnių duomenis iš peržiūros erdvės į projekcijos. Šiame etape objektų mastelis yra keičiamas priklausomai nuo

atstumo iki žiūrovo, tam kad susidarytų scenos gylio įspūdis (arčiau esantys objektai vaizduojami didesni už nutolusius).

Etapų atlikimo tvarka gali keistis, kai kurie iš jų gali būti praleisti. Paprasčiausias *vertex processing* konvejeris gali būti sudarytas tik iš bendrosios, peržiūros ir projekcijos transformacijų.

Paveikslėlyje matyti, kad viršūnės perduodamos viršūnių spalvinimo efektui su vietinėmis koordinatėmis ir viršūnių spalvinimo efektas turi grąžinti (*output*) nuspalvintas viršūnes projekcijos erdvėje. Viršūnių transformavimui iš vietinės erdvės į projekcijos erdvę yra atliekamos : bendroji transformacija (*world transformation*), peržiūros transformacija (*view transformation*) ir projekcijos transformacija (*projection transformation*). Šios transformacijos atliekamos panaudojant pasaulio matricą (*world matrix*), peržiūros matricą (*view matrix*) ir projekcijos matricą (*projection matrix*). Taško primityvui viršūnių spalvinimo efektas gali būti naudojamas kiekvienos viršūnės dydžio manipuliavimui.

Kadangi viršūnių spalvinimo efektas yra programa, parašyta pasirinkta programavimo kalba (pvz. HLSL), todėl vartotojas įgyja daug lankstumo kuriant grafinius efektus. Pavyzdžiui, viršūnių spalvinimo efektams galima pritaikyti bet kokį apšvietimo algoritmą, kurį galima įterpti į viršūnių spalvinimo efektą. Programuotojas nebėra priverstas naudoti tik Direct3D fiksuotus apšvietimo algoritmus. Be to atsiranda viršūnės pozicijos manipuliavimo galimybė, padedanti atlikti vėliavos plevėsavimą, rūbų kritimą, plaukų plaikstymąsi, vandens raibuliavimą, manipuliavimas taško dydžių (tai svarbu kuriant lietų, fontanus, dulkes ir kt.) ar viršūnių maišymą ir perėjimą iš vienos į kitą (pvz., žmogaus veidas pereina į šuns) *(blending/morphing*). Be to viršūnių duomenų struktūros yra lankstesnės ir gali turėti daug daugiau duomenų programiniame konvejeryje nei fiksuotų funkcijų konvejeryje.

3.4.2.1. Viršūnių spalvinimo efektų versijų instrukcijų kiekiai

Dirbant su Direct3D 8.0 ir 8.1 versijomis yra palaikomos tik 1.0 ir 1.1 viršūnių spalvinimo efektų versijų instrukcijos. Jei dirbama su Direct3D 9.0 ar vėlesne versija, tada palaikomos 2.0 ar 3.0 versijų instrukcijos. Instrukcijų kiekiai 2x ir aukštesnėse versijose yra didesni, kadangi yra palaikomi ciklai ir vykdomų instrukcijų skaičius efektyviai padidinamas.

Direct3D	**8.1**	**9.0**				
Versija	1.1	2.0	2.0+	2.0_{sw}	3.0	3.0_{sw}
Instrukcijų kiekis	128	256	256	256	512+	512+

3.4.3. Pikselių spalvinimo efektai

Pikselių spalvinimo efektas yra programa vykdoma grafinės plokštės GPU rasterizacijos proceso metu kiekvienam pikseliui. (Skirtingai nei viršūnių spalvinimo efektai, Direct3D neemuliuoja pikselių spalvinimo efektų programinėje įrangoje.) Šie spalvinimo efektai suteikia programuotojui galimybę tiesiogiai manipuliuoti atskirais pikseliais ir prieiti prie kiekvieno pikselio tekstūrų koordinačių. Šis tiesioginis priėjimas prie pikselių ir tekstūrų koordinačių leidžia sukurti daug įvairių efektų, tokių kaip debesų, ugnies imitavimą, daugelio

tekstūrų panaudojimą (*multitexturing*), kiekvieno pikselio apšvietimą, sudėtingus šešėlių uždėjimo būdus ir kt.

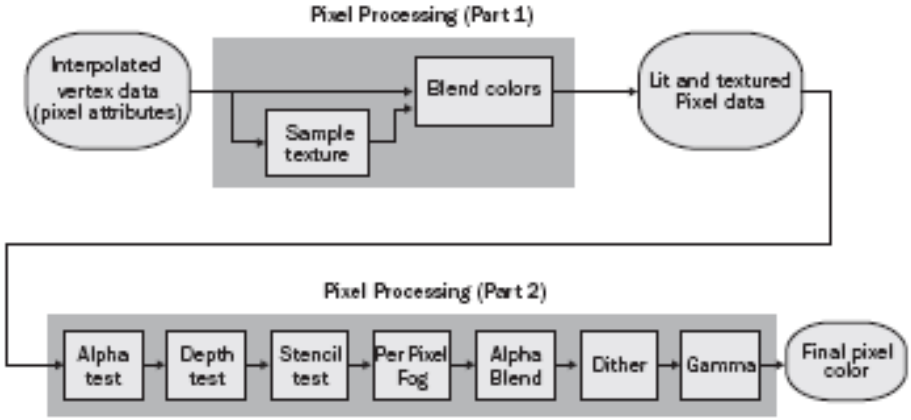
Norint geriau suprasti šių spalvinimo efektų galimybes, reikia išsamiau panagrinėti Direct3D grafinio konvejerio pikselių apdorojimo (*pixel processing*) etapą.

3.4.3.1. Pikselių apdorojimo etapas (pixel processing)

Įvykdžius *geometry processing* etapą duomenims jau būna atlikta rasterizacija, t.y. viršūnių duomenys yra interpoliuoti į pikselius. Pikselių apdorojimo etape yra sumaišoma keletas pikselių duomenų tipų (*pixel data types*), tekstūros pavyzdžių (*texture samples*) ir grąžinamos pikselių spalvos.

Pikselių apdorojimo etapą galima padalyti į dvi dalis (pavaizduota schemoje, paimtoje iš [4] šaltinio):

- pirmoje dalyje interpoliuoti viršūnių duomenys (pvz., šviesos išsklaidymo spalva (diffuse color), atspindžio spalva, tekstūrų koordinatės) yra konvertuojami į vieną ar daugiau pikselio spalvų.
- antroje dalyje pikselių spalvos konvertuojamos į galutinę vizualizuotą pikselio spalvą.



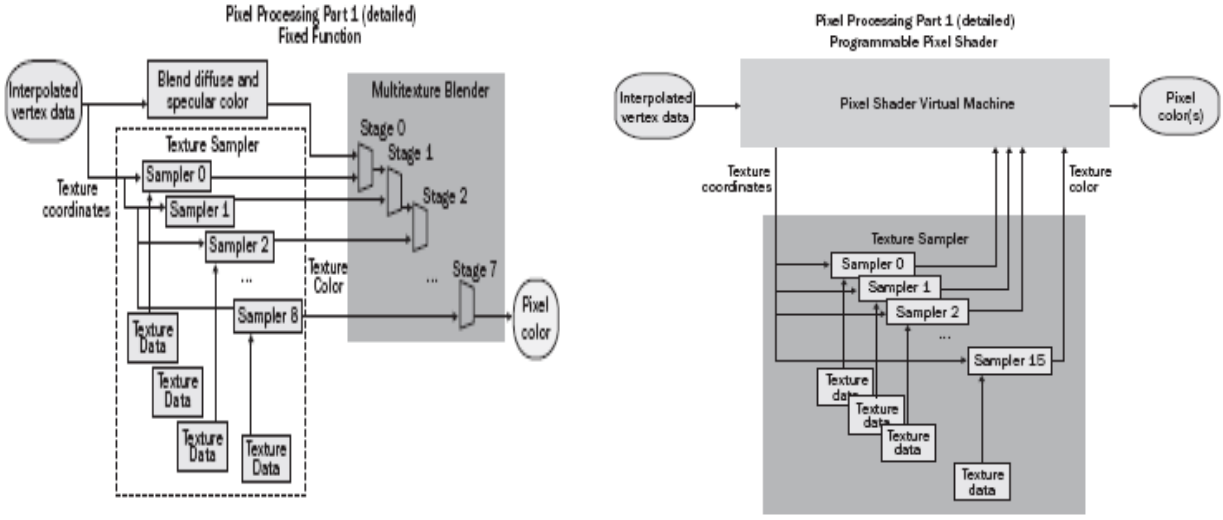
Pirmąją pikselių apdorojimo dalį sudaro tokie etapai:

Tekstūros parinkimas (*Sample texture*). Atrenka vieną ar daugiau tekstūrų.
Maišymas (*Blend*). Maišomi kiekvieno pikselio atributai (dažniausiai difuzinės *(diffuse*) ir atspindžio (*specular*) spalvos ir/arba atrinktos tekstūros).

Antroje dalyje atliekami šie pikselių apdorojimo etapai: alfa testas, gylio testas, šablono (*stencil*) testas, rūko kiekvienam pikseliui uždėjimas, alpha blend, dither, gamma.

Pikselių apdorojimo etapas buvo išskirstytas į dvi dalis tam, kad būtų lengviau suvokti pikselių spalvinimo efektų programavimą. Pirmosios pikselių apdorojimo etapo dalies žingsniai gali būti programuojami panaudojant pikselių spalvinimo efektus, o antrosios – atliekami konvejerio fiksuotomis funkcijomis, po to kai yra įvykdomi pikselių spalvinimo efektai.

Fiksuotų funkcijų konvejeryje pikselių apdorojimo etapo pirmos dalies atlikimui yra naudojamos tekstūros etapų būsenos (*texture stage states*) ir daugelio tekstūrų maišymas (*multitexture blender*). Tai yra pavaizduota pirmojoje diagramoje[4]:



Tekstūrų dalių apdorojimas (*texture sampler*) yra sudarytas iš tekstūrų duomenų (tekstūros failas ar vizualizuota scena (*render target*)) ir atrinkėjų (sampler) (naudojamas tekstūrų parinkimui t.y. naudoja tekstūros koordinates tekstūros spalvos nustatymui). Fiksuotų funkcijų konvejeryje yra 8 atrinkėjai (*samplers*). Daugelio tekstūrų maišymas (*blender*) yra atliekamas aštuoniais maišymo etapais. Maišymo etapai yra išdėstyti tekstūros maišymo pakopomis taip, kad 0-inio etapo gražinimas yra 1-mo etapo įvestis, 1-mo išvestis yra 2-ro etapo įvestis ir t.t.

Sekančioje diagramoje [4] yra pavaizduotas pikselių spalvinimo efekto atlikimas. Pikselių spalvinimo efektas taip pat naudoja atrinkėjus (*samplers*) tekstūrų atrinkimui, tačiau daugelio tekstūrų maišymas yra atliekamas spalvinimo efekto virtualioje mašinoje (*pixel shader virtual machine*). Tokiu būdu maišymo operacijos tampa programuojamomis. Tai ir yra esminis pikselių spalvinimo efektų naudojimo privalumas, nes

- nebereikia konfigūruoti tekstūrų maišymo ar naudoti viso maišymo operacijų rinkinio.
- galima pasinaudoti pikselių spalvinimo efektų kūrimui skirtų programavimo kalbų operacijomis arba parašyti savo kodą.
- pikselių spalvinimo efektų pradiniais duomenimis gali būti viršūnių spalvinimo efektų išvesties duomenys.

Grafinis konvejeris pikselių apdorojimo etape gali atlikti arba pikselių spalvinimo efektus, arba fiksuotas konvejerio funkcijas, tačiau pikselių apdorojimas negali būti atliekamas abiem būdais tuo pačiu laiku. Jei yra naudojamas pikselių spalvinimo efektas, jis turi atlikti visus pikselių apdorojimo etapo pirmos dalies funkcinius žingsnius.

3.4.3.2. Pikselių spalvinimo efektų versijų instrukcijos

Direct3D	**8.0**		**8.1**		**9.0**				
Versija	1.1	1.2	1.3	1.4	2.0	2.0_x	2.0_{sw}	3.0	3.0_{sw}
Instrukcijų kiekis	8	8	8		96	512			

3.4.4. HLSL spalvinimo efektų integravimas į 3D programą

Kaip jau minėjau anksčiau, HLSL spalvinimo efektams yra reikalingi 3D modeliai, tekstūros ir kt. duomenys, todėl spalvinimo efektų paleidimui yra reikalingos programos, parašytos centriniam procesoriui tradicinėmis kalbomis (C, C++ ir kt.). HLSL spalvinimo efektus į programą galima integruoti dviem būdais:

1. panaudojant D3DX efektus;
2. nepanaudojant D3DX efektų.

3.4.5.1. HLSL spalvinimo efektų integravimas į 3D programą, panaudojant D3DX efektus

D3DX efektai yra dažnai naudojamas D3DX bibliotekos komponentas. DirectX 9 versijoje D3DX efektai buvo atnaujinti, pridedant HLSL spalvinimo efektų palaikymą. Šie efektai yra aukšto lygio abstrakcija, sukurta vizualizuotų specialių efektų išdėstymui 3D programose. Efektą dažniausiai sudaro tokie komponentai: viršūnių ar/ir pikselių spalvinimo efektai, įrenginio (*device*) būsenų sąrašas, ir vienas ar daugiau vizualizavimo praėjimai (*passes*). Efektas paprastai yra saugomas viename .fx arba .fxl faile, kuriame gali būti daug efekto versijų. Pavyzdžiui, jūs galite sukurti to paties efekto keletą versijų, skirtų skirtingų galimybių techninei įrangai. Efektų panaudojimas turi tokius privalumus:

- galima keisti efekto realizavimą (*implementation*) neperkompiliuojant 3D programos kodo;
- visi efekto komponentai yra patalpinami viename faile.
- su .fx failais yra suderinami 3D grafikos paketai: Alias Maya, Discreet 3ds MAX ir Softimage XSI. Čia .fx failai naudojami norint, kad žaidimų kūrėjai galėtų pamatyti kaip 3D scena atrodys žaidime.

3.4.5.2. HLSL spalvinimo efektų integravimas į 3D programą, nepanaudojant D3DX efektų

Yra nemažai savarankiškų programinės įrangos tiekėjų (ISVs), kurie nenori savo kodo labai glaudžiai susieti su D3DX dėl daugiaplaformių programų kūrimo (*cross-platform development*) bei priežiūros ypatumų. Taigi, nors D3DX efektų naudojimas HLSL spalvinimo efektu valdymui (*management*) yra patogus, tačiau jis ne visiems yra priimtinas.

HLSL spalvinimo efekto kodą galima tiesiog įrašyti į 3D programą kaip ilgą simbolių eilutę, tačiau daug patogiau yra spalvinimo efekto kodą atskirti nuo 3D programos kodo. Todėl spalvinimo efekto kodas yra rašomas Notepad‘e ir išsaugomas kaip ASCII tekstinis failas.

Atsisakius D3DX efektų naudojimo patogumo, 3D programoje reikia atlikti reikiamų konstantų ir atrinkėjų (*samplers*) nustatymą ir perdavimą spalvinimo efektui, iškviesti HLSL kompiliatorių ir t.t. Tiesą sakant 3D programos kodas turės būti rašomas labai panašiai, kaip naudojant asembleriu parašytus spalvinimo efektus, išskyrus tai, kad vietoj kurios nors D3DXAssembleShader*() paprogramės, reikės iškviesti vieną iš D3DXCompileShader*() paprogramių (jeigu HLSL spalvinimo efekto kodas yra atskirame faile, tai D3DXCompileShaderFromFile paprogramę). D3DXCompileShader*() paprogramės turi keletą papildomų parametrų: spalvinimo efekto pagrindinės funkcijos (*main*) pavadinimą, realizacijos

versiją (*compile target*). Taip pat galima (nebūtinai) nustatyti #defines reikšmes, įterptus (*include*) failus, kompiliavimo optimizavimo vėliavėles ir kt. (priklausomai nuo paprogramės). Šie nustatymai yra perduodami D3DXCompileShader*() paprogramėms per pirmuosius šešis parametrus, o likę trys parametrai yra rodyklės į buferius, kuriuos užpildo kompiliatorius: sukompiliuotas spalvinimo efekto kodas, klaidų ir perspėjimų sąrašas ir konstantų lentelė (*constant table*). Sukompiliuotas kodas perduodamas CreatePixelShader() arba CreateVertexShader(), o programa, pagal konstantų lentelėje esančią informaciją, užkrauna HLSL spalvinimo efektui reikalingas konstantas.[8]

D3DXCompileShader*() paprogramės gražinama konstantų lentelė yra naudojama aukšto lygmens konstantų ir atrinkėjų (samplers) susiejimui su konkrečiomis techninės įrangos konstantomis ir atrinkėjais. Konstantų lentelės informacija yra pasiekiama per ID3DXConstantTable sąsajos metodus: ID3DXConstantTable::GetDesc(), ID3DXConstantTable::GetConstantElement(), ID3DXConstantTable::GetSamplerIndex, ID3DXConstantTable::GetConstantByName ir kt. (*Plačiau pasiskaityti galima DirectX SDK dokumentacijoje*).

3.5. Spalvinimo efektų panaudojimo pavyzdžiai

Anksčiau buvo minėta, kad spalvinimo efektų kūrimas labai priklauso nuo turimos grafinės techninės įrangos ir, kad dabartinės vaizdo plokštės apriboja spalvinimo efektų galimybes. Iš to buvo galima susidaryti vaizdą, kad dabar kuriami spalvinimo efektai yra labai paprasti ir prastai atrodo, todėl norėdama parodyti bent menkutę Cg-HLSL programavimo kalbos galimybių dalį, į darbą įdėjau šiuos, naudojant spalvinimo efektus, sukurtus pavyzdžius:

3.5.1. Apšvietimas

Prieš pradedant nagrinėti viršūnių ir pikselių spalvinimo efektų panaudojimą objektų apšvietimui, reikia paminėti, kad čia nėra naudojamos fizikoje žinomos šviesos sklidimo, atspindžių ir kt. formulės, kadangi jos yra per daug sudėtingos ir per daug ilgai atliekamos. Apšvietimui kurti viršūnių ir pikselių spalvinimo efektuose yra naudojamos paprastos, daugelio bandymų metu atrastos trumpos formulės, kurias panaudojus išgaunamas panašus į natūralų apšvietimas. Šias formules su paaiškinimais galima rasti Wolfgang F. Engel., Beginning Direct3D® Game Programming 2nd Edition. knygoje.

Aplinkos (bendras) apšvietimas (*Ambient Lighting*)

Scenoje, kurioje yra naudojamas aplinkos apšvietimas, iš visų pusių, vienodo intensyvumo šviesos yra nukreiptos į objektą.

Išsklaidytas apšvietimas (*Diffuse Lighting*)

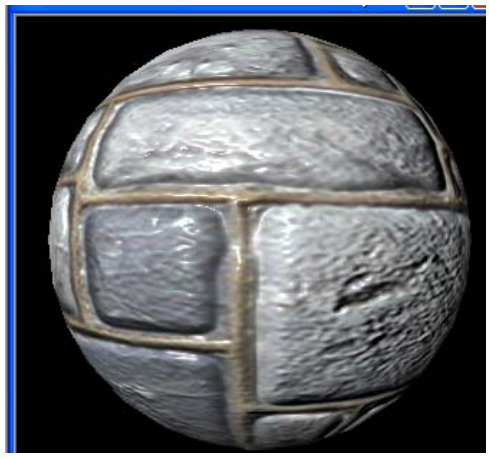
Išsklaidytas apšvietimas turi nurodytą šviesos šaltinio padėtį erdvėje. Tai yra pagrindinis skirtumas tarp išsklaidytos šviesos ir aplinkos šviesos, kuri neturi padėties erdvėje ir todėl laikoma globalia. Išsklaidytos šviesos atspindys nepriklauso nuo žiūrovo pozicijos. Išsklaidyta šviesa dažnai naudojama matinių paviršių imitavimui.

Atspindintis apšvietimas (*Specular Lighting*)

Naudojamas, kai reikia imituoti lygius ir blizgius paviršius.

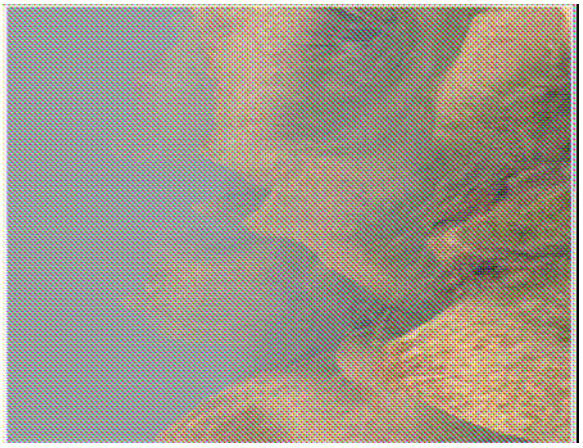
3.5.2. Gruoblėtų tekstūrų uždėjimas (*bump mapping*)

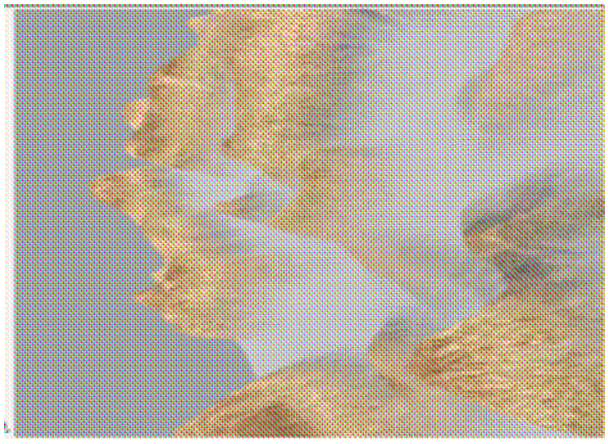
Nelygumų kūrime yra imituojami įvairūs nelygumai pvz., įdubimai, iškilimai ir kt. Gruoblėtų tekstūrų (*Bump Map*) spalvų reikmės yra saugomos tokiuose grafiniuose failuose kaip *.dds ar *.tga, kuriuose yra saugomos normalės, kurios skaičiuojant apšvietimą yra naudojamos vietoje viršūnių normalių.



3.5.3. Rūkas

Rūko efektai gali būti labai įvairūs (*jų kūrimo būdai yra paaiškinti Wolfgang F. Engel., ShaderX2 Introductions and Tutorials with DirectX9*): linijinis (1 pav.), sluoksniuotas (2 pav.), eksponentinis ir kt.



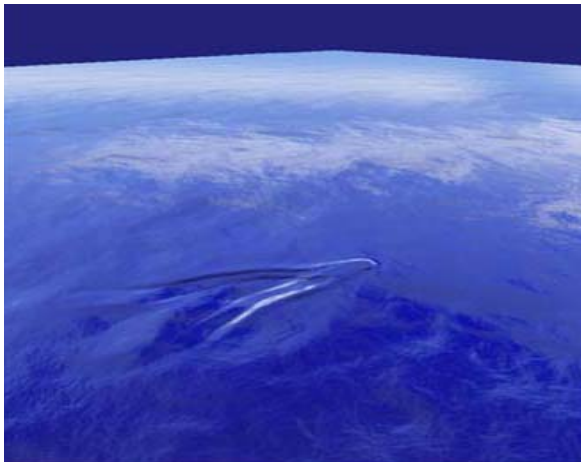


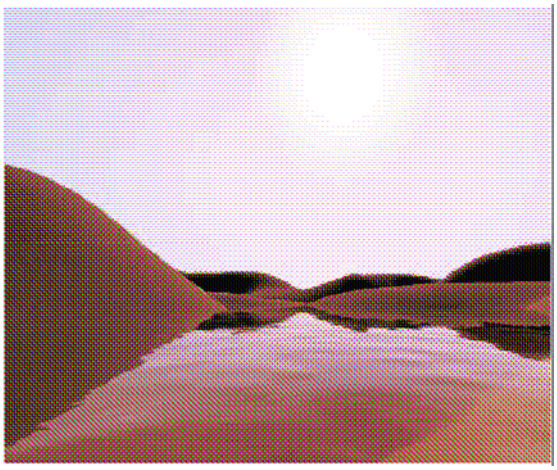
3.5.4. Animacija

Šiame pavyzdyje yra naudojamas viršūnių spalvinimo efektas (*vertex shader*), kuris pagal laiko trukmę, atlieka viršūnių animaciją. Šio pavyzdžio platesnis aprašymas ir algoritmas yra pateiktas *DirectX SDK (April 2006)*.

3.5.5. Vanduo

Vandens efektų kūrimas aiškinamas *Wolfgang F. Engel., Shader Programming Tips and tricks with DirectX 9* knygoje.





3.5.6. Veidrodinis atspindys

Naujausiame metode yra naudojamas aplinkos modelis (*Environment-Map*): tekstūroje-pavyzdyje sutalpinamas visas aplinkos vaizdas. Pikselių spalvinimo efektas nurodo iš kurios pusės kris šviesos spindulys, kuris atsispindės stebėtojui kiekviename objekto taške. Šią kryptį atitinka tam tikras aplinkos modelio (*Environment-Map)* taškas, kurio spalva atsispindi nuo objekto. Aplinkos modeliui vaizduoti naudojamas kubas (*Cube-Map*), kurio šešiose sienose yra pavaizduota: vaizdas nuo objekto į priekį, atgal, kairėn, dešinėn, į viršų ir į apačią. Daugiau apie veidrodinį atspindį galima pasiskaityti [16] šaltinyje.

4. Įrankių ir priemonių pasirinkimas

4.1. C++, C ar C#

Yra daug programavimo kalbų, kuriomis būtų galima kurti žaidimus, tačiau šiuo metu daugiausia žaidimų yra sukurta pasinaudojant C++ programavimo kalba. Žaidimų kūrimui dar naudojama C kalba, tačiau kuriant vis sudėtingesnius ir galingesnius žaidimus dažniausiai yra pasirenkama C++, kadangi C++ kalbos klasių sistema ir operacijų perdengimas (*overloading*) palengvina didžiulės apimties kodo rašymą ir tvarkymą, be to C++ geriau palaiko COM, nei C. Pastaruoju metu konkurente tampa C# , tačiau žaidimai parašyti su C# eina lėčiau nei su C++.

4.2. Visual Studio.net 2003, Visual Studio 2005

DirectX dokumentacijoje yra nurodyta, kad DirectX programų kompiliavimui ir derinimui turi būti naudojami tik Visual Studio .NET 2002 ir vėlesni įrankiai, o spalvinimo efektų kompiliavimui geriausia naudoti Visual Studio .NET 2003 ar Visual Studio 2005. Iš pažiūros Visual Studio .NET 2003 ir Visual Studio 2005 yra labai panašūs, tačiau su Visual Studio 2005 atsidarant su Visual Studio .NET 2003 parašytas programas, jos yra pritaikomos darbui su Visual Studio 2005 , o po to atlikus kompiliavimą dažnai nebegali atidaryti kai kurių failų. Kadangi daugelis, spalvinimo efektus palaikančių, programų yra sukurtos panaudojant Visual Studio .NET 2003, todėl aš jį daugiausia ir naudojau.

4.3. Specialios spalvinimo efektų kūrimo priemonės

Daugelis šiandieninių 3D grafikos programų bei žaidimų vartotojų ir kūrėjų susiduria su grafinių spalvinimo efektų kūrimu. Šie efektai kuo toliau, tuo dažniau yra naudojami, todėl labai tikėtina, kad ateityje visos vaizdo plokštės palaikys spalvinimo efektus. Spalvinimo efektai yra glaudžiai susieti su modeliais (svarbi modelio padėtis bendroje koordinačių sistemoje, kameros padėtis, tekstūrų koordinatės ir t.t.), todėl norint juos peržiūrėti reikia prieš tai sukurti modelį kuriam tie efektai bus uždedami. Su tuo pačiu projektu dirbant keliems žmonėms, spalvinimo efektų programuotojui tenka laukti kol modeliuotojas sukurs modelį.

Siekiant padėti spalvinimo efektų kūrėjams, buvo sukurtos spalvinimo efektų kūrimo ir peržiūros programos, naudojančios standartinius trimačius modelius, kuriems ir galima uždėti parašytus spalvinimo efektus. Šiuo metu labiausiai paplitusios yra:

- ATI sukurta RenderMonkey integruotų programų rengimo terpė (IDE) (nemokamas);
- nVidia sukurtas FX composer (nemokamas);

FX Composer ir RenderMonkey suteikia galimybę kurti aukštos kokybės spalvinimo efektus ir peržiūrėti juos realiame laike. Šiuose įrankiuose yra panaudotos įvairios priemonės, palengvinančios spalvinimo efektų kūrimą, pvz: programos rašomas tekstas vaizduojamas skirtingomis spalvomis; sukurti importavimo/eksportavimo įrankiai, todėl efektus galima uždėti ne tik ant standartinių, bet ir ant savo sukurtų modelių; sukurta nemažai standartinių spalvinimo efektų, kuriuos galima panaudoti kuriant savo efektus; parodomos kompiliavimo metu rastos klaidos ir t.t.

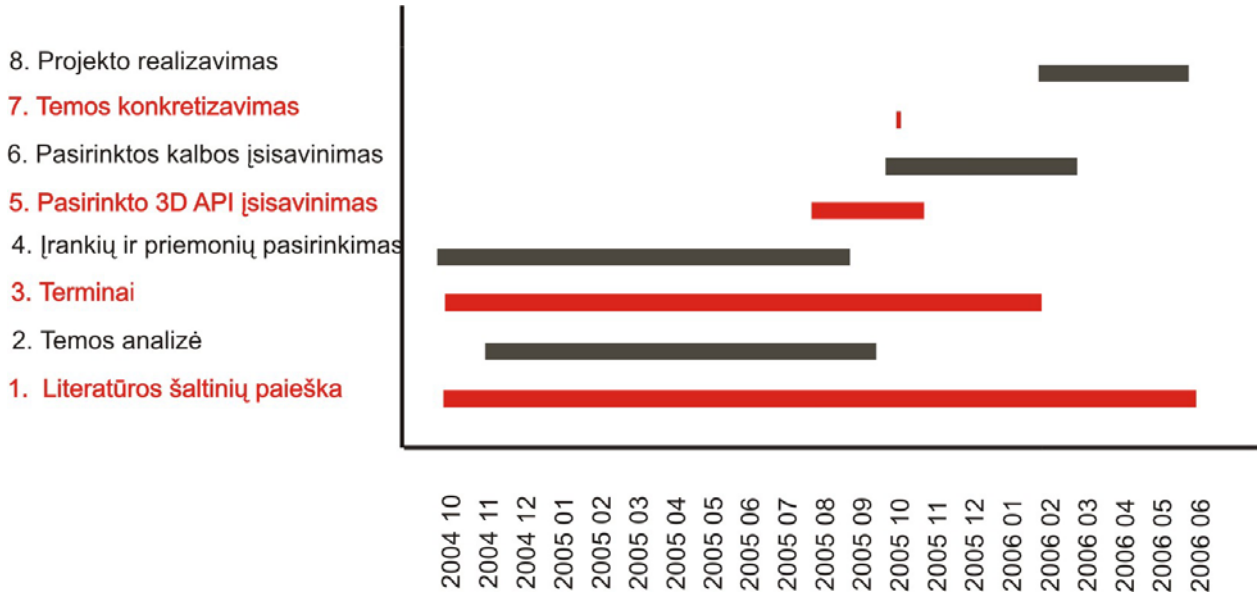
Mažiau populiarios yra šios spalvinimo efektų kūrimo ir redagavimo priemonės:

- Effect Edit , kuris buvo patalpintas DirectX 9.0 SDK, tačiau jau išimtas iš DirectX SDK (April 2006)(nemokama);
- Shader Ninjam (nemokama);
- ShaderworksXT (nemokama);
- RT/shader (mokama) ir kt.;

5. Projekto vykdymo planas.

1. Literatūros šaltinių paieška .*Internetas, knygos, straipsniai (2004.10-2006.05)*
2. Išanalizuoti temą. *Kas yra spalvinimo efektai? Kam jie naudojami? Kas jau yra atlikta panaudojant spalvinimo efektus ir kokios yra jų galimybės?(2004 11-2005 09)*

3. Išsiaiškinti terminus. *Kas yra Vertex shader'is, Pixel shader'is, 3D grafinis konvejeris ir kt.(2004.12-2006.05)*
4. Pasirinkti įrankius ir priemones. *DirectX ar OpenGL, C++, C ar C#, Microsoft Visual Studio 6.0 ar Microsoft Visual Studio .NET. Pasirinkti spalvinimo efektų programavimo kalbą (iki 2005.09)*
5. Įsisavinti pasirinktą trimatį grafinį API. *Išsiaiškinti jo architektūrą, spalvinimo efektų įterpimą, kompiliavimą ir derinimą. (2005.08-2005.10)*
6. Įsisavinti pasirinktą spalvinimo efektų programavimo kalbą. *Įsisavinti pasirinktos programavimo kalbos pagrindus, spalvinimo efektų rašymo ypatybes ir panaudojimą.(2005.10-2006.02)*
7. Temos konkretizavimas (2005.10).
8. Projekto realizavimas (2006.02-2006.05).



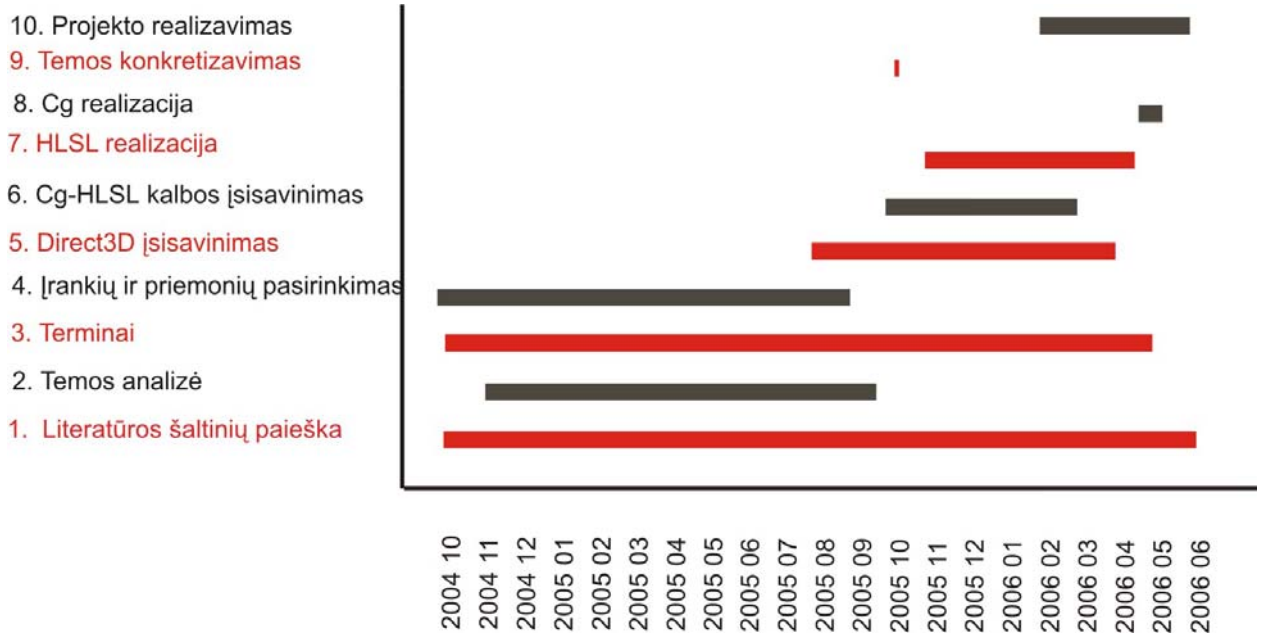
6. Pradinis projekto aprašymas

Šiame darbe atliekant pasirinktos aukšto lygmens spalvinimo efektų programavimo kalbos galimybių analizę ir taikymą trimatės grafikos programose numačiau :

- išsiaiškinti kas yra dinaminiai spalvinimo efektai, kokių jie yra tipų ir kur naudojami;
- apžvelgti dinaminių spalvinimo efektų kūrimo priemones ir išanalizuoti jų taikymo galimybes;
- išsiaiškinti kokia techninė įranga juos palaiko;
- nustatyti kokios yra dinaminių spalvinimo efektų aukšto lygmens programavimo kalbos yra populiariausios ir atlikti pasirinktos dinaminių spalvinimo efektų programavimo kalbos galimybių analizę;
- išsiaiškinti kaip yra vykdomi aukšto lygmens kalba parašyti dinaminiai spalvinimo efektai ir kaip jie įterpiami į 3D programą;
- išsinagrinėti jau sukurtus pavyzdžius ir išsiaiškinti naudojamus metodus, kad galėčiau jais pasinaudojant sukurti keletą savo spalvinimo efektų.

7. Darbo eigos aprašymas

7.1. Darbų eigos grafas



7-8**.** Išsiaiškinus, kad Cg ir HLSL yra dvi skirtingos Cg-HLSL programavimo kalbos realizacijos, teko apžvelgti ne tik HLSL, bet ir Cg..

7.2. Problemų ir jų sprendimų aprašymai ir pagrindimai

Atlikdama šį darbą, susidūriau su nemažai problemų. Aiškumo dėlei jas sugrupavau:

1. Technines – finansinės;
2. Mažai prieinamos literatūros;
3. Terminai;
4. Įvairūs informacijos neatitikimai rastoje literatūroje;
5. Spalvinimo efektų algoritmų rašymo sudėtingumas.

1. **Technines – finansinės**. Kai tik įsigilinau į temą ir galų gale suvokiau kam yra reikalingi spalvinimo efektai ir kokia grafinė techninė įranga juos palaiko, man tapo aišku, kad šio darbo atlikimui bet koks kompiuteris (o tuo labiau mano 6-erių metų „senelis“ su sena vaizdo plokšte) tikrai netiks. Pirkti naujo galingo kompiuterio neturėjau už ką, o manasis negalėjo atlikti net ir paprasčiausio spalvinimo efekto. Teko kreiptis į Šiaulių universiteto multimedijos laboratoriją, kur man buvo leista atlikti eksperimentus su multimedijos laboratorijos įranga. Eksperimentai buvo atlikti su kompiuteriu, turinčiu tokią vidinę įrangą:

1. Procesorius INTEL 630 84 3,06 GHz;
2. Video plokštė MSI NX6200-TD/128MB;
3. Pagrindinė plokštė Asus P5N 2 – SLI;
4. RAM AM1 1024MB DDRII (2x512 MB);
5. HDD 2vnt. po 80GB

2. **Literatūra.** Aukšto lygmens spalvinimo efektų programavimo kalbos atsirado dar visai neseniai, todėl internete daugiausiai galima rasti tik įvadą į spalvinimo efektų programavimą, o rimtesnės informacijos rasti labai sunku. Mikrosoft‘o tinklapyje ir DirectX dokumentacijoje pateikta taip pat nedaug informacijos apie HLSL spalvinimo efektų kalbą, o mokymo programa gana glausta. Rasti man tinkančių knygų nei techninėje, nei P.Višinskio viešojoje bibliotekoje nepavyko, todėl literatūros paieška gerokai užtruko.

3. **Terminai.** Interaktyvioji trimatė kompiuterinė grafika yra nauja ir labai sparčiai besivystanti informatikos sritis, todėl šioje srityje yra sukuriami vis nauji terminai. Kadangi Lietuvoje yra mažai interaktyvios trimatės kompiuterinės grafikos žinovų, o ir jie arba nesidomi spalvinimo efektų kūrimu, arba to neskelbia, todėl susidūriau su labai rimta problema – nėra šių terminų lietuviškų vertimų, o ir kai kurie esami yra išversti pažodžiui ir nelabai atitinka prasmę pavyzdžiui, spalvinimo efektus (*shader*) siūloma vadinti šešėliuote, o tai neatitinka (tiksliau labai susiaurina) suvokimą apie spalvinimo efektus, nes šešėlių uždėjimas ant objektų tėra tik labai siaura spalvinimo efektų panaudojimo sritis. Taigi, skaitydama anglišką literatūrą, sugaišau nemažai laiko, nes nesuprantant terminų buvo sudėtinga suprasti esmę, todėl sutikus naują terminą iš pradžių teko išsiaiškinti ką jis reiškia, o rašant darbą ir surasti tinkamiausią (bent jau mano subjektyvia nuomone) lietuvišką atitikmenį. Dėl nežinomų trimatės grafikos terminų gausumo, buvo labai sunku įsigilinti į visas technines spalvinimo efektų detales, pavyzdžiui Direct3D grafiniame konvejeryje yra atliekamos operacijos (maišymas (*blending*), tekstūrų atrinkimas (*texture sample*), alfa testas, rasterizacija ir daug kitų), kurios man iš pat pradžių buvo nežinomos ir nesuprantamos. Nemažai terminų paaiškinimų (tiesa teko atsirinkti, nes ne visi teisingi) radau http://www.karpis.puslapiai.lt/

4. **Įvairūs informacijos neatitikimai rastoje literatūroje**. Manau reikia paminėti, kad DirectX dokumentacijoje neradau informacijos apie tai, kad Cg ir HLSL yra tos pačios programavimo kalbos, dvi skirtingos realizacijos. Be to, kituose tinklapiuose ir gautose knygose rašoma kad, HLSL ir Cg yra skirtingos spalvinimo efektų programavimo kalbos (nors kai kur ir paminima, kad jos labai panašios), todėl iš pradžių ieškojau literatūros tik apie HLSL. Į NVIDIA kompanijos tinklalapį užėjau tik norėdama pažiūrėti kaip atrodo Cg spalvinimo efektų programavimo kalba ir radau, kad tai viena ir ta pati programavimo kalba. Šios informacijos suradimas nemažai pakeitė mano darbą (nes iki tol rašiau tik apie HLSL realizaciją) ir kartu uždavė dar vieną galvosūkį – nebežinojau kaip vadinti šią aukšto lygmens programavimo kalbą: Cg ar HLSL, o gal Cg-HLSL? Žinoma ši problema nebūtų iškilusi, jeigu Microsoft‘as būtų paminėjęs, kad HLSL yra Cg programavimo kalbos realizacija, o dabar tenka remtis tik NVIDIA kompanijos informacija, kuri gali būti subjektyvi. Todėl nenorėdama suklysti, šiame darbe šią programavimo kalba vadinu Cg-HLSL, o jos realizacijas Cg ir HLSL.

Kitą informacijos neatitikimą radau nagrinėdama Direct3D grafinį konvejerį. Netgi Direct3D dokumentacijoje šis grafinis konvejeris skirtingose vietose aiškinamas skirtingai. Nagrinėjant Direct3D architektūrą, yra išskirti du pikselių apdorojimo etapai: pixel processing ir pixel rendering, o jau nagrinėjant pikselių spalvinimo efektus, aiškinama, kad pikselių apdorojimo etapas (pixel processing) yra sudarytas iš dviejų dalių, o pixel rendering nebėra. Paskaičius nemažai knygų susipainiojau visai, nes kiekvienoje tas pats Direct3D konvejeris aiškinamas visiškai kitaip, matyt autoriai jį supranta skirtingai. Nenorėdama suklysti, šiame darbe rėmiausi Direct3D dokumentacijoje rastais abiem Direct3D konvejerio etapų skirstymais ir Microsoft DirectX programuotojo Kris‘o Gray‘aus , dirbančio su Direct3D, parašyta knyga „Microsoft DirectX 9 programmable graphics pipeline“.

5. **Spalvinimo efektų algoritmų rašymo sudėtingumas.** Pradėjus įsisavinti Cg-HLSL programavimo kalbą, viskas atrodė gana aišku: ciklai, daug įvairių matematinių funkcijų, sąlygos sakiniai... Trumpai tariant kažko labai naujo ir nesuprantamo šios kalbos sintaksėje nemačiau ir net nemaniau, kad su programavimu turėsiu rimtų bėdų. Tačiau taip buvo tik iš pradžių, nes panagrinėjusi kelis pavyzdžius supratau, kad nors aš ir moku programuoti, tačiau neturiu jokio supratimo ką ir iš ko reikia padauginti ar kokią panaudoti

funkciją kad gaučiau norimą rezultatą. Pasirodo spalvinimo efektų programavime yra sukurti kažkokie metodai, kuriuos naudojant gaunamas reikiamas rezultatas. Pavyzdžiui, kuriant apšvietimą, yra naudojama formulė:

$I = I_{ambient} + I_{diffuse} + I_{specular}$, kur

$$I_{ambient} = k_a \cdot I_a$$
$$I_{diffuse} = k_d \cdot I_d \cdot (\vec{N} \bullet \vec{L})$$
$$I_{specular} = k_s \cdot I_s \cdot (\vec{V} \bullet \vec{R})^{n_s}$$

$\vec{N}, \vec{V}, \vec{R}$ - Normalės, peržiūros ir atspindžio vektoriai.

k_a, k_d , k_s – koeficientai nuo 0 iki 1;
I_a,I_d, I_s – intensyvumo spalva;

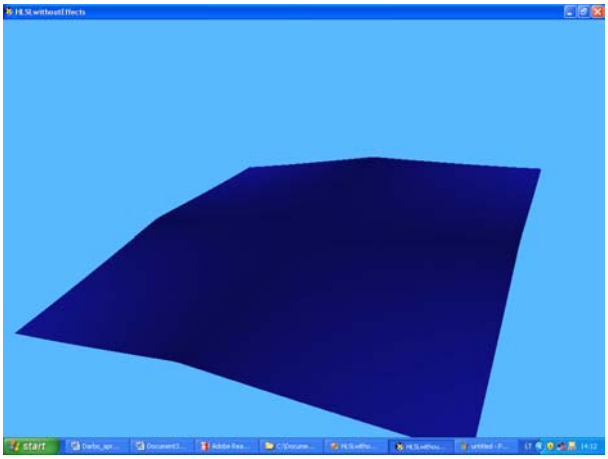
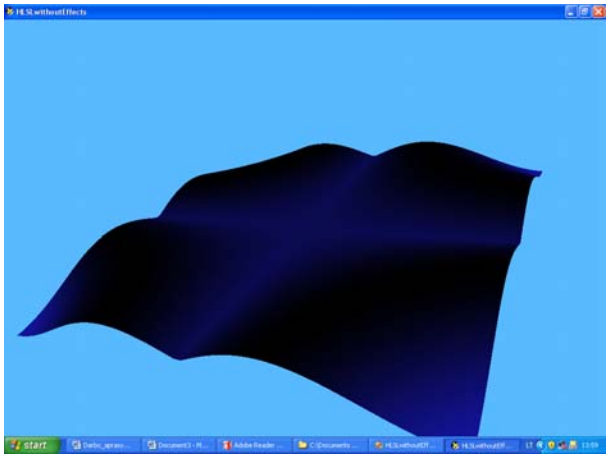
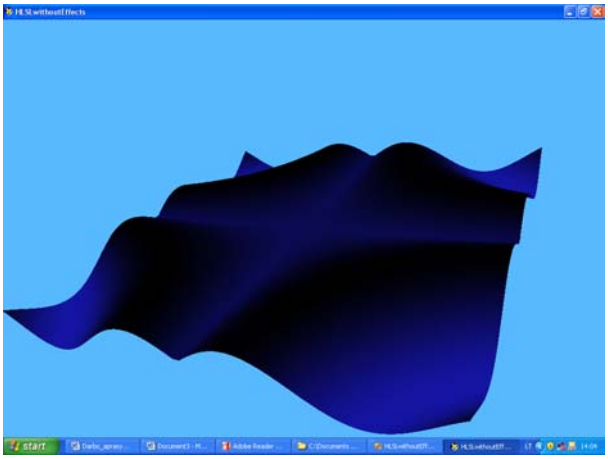
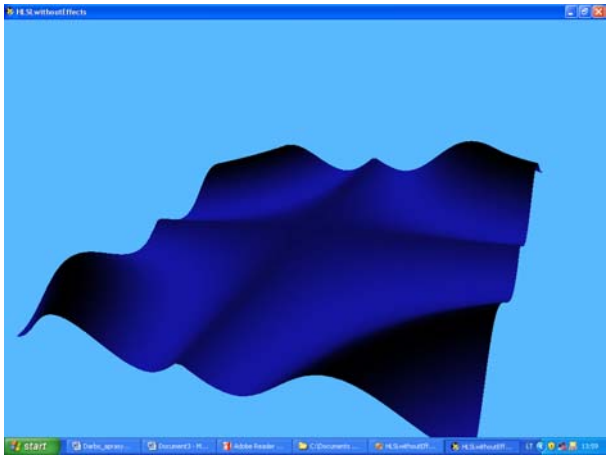
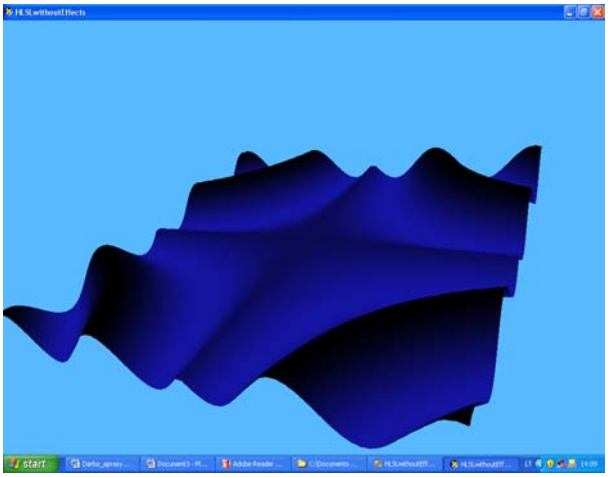
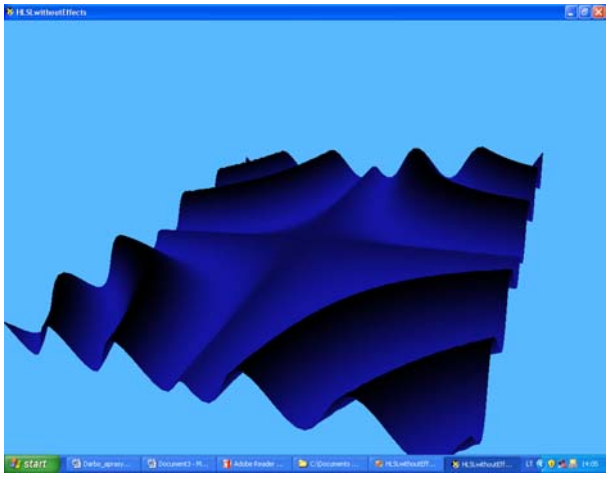
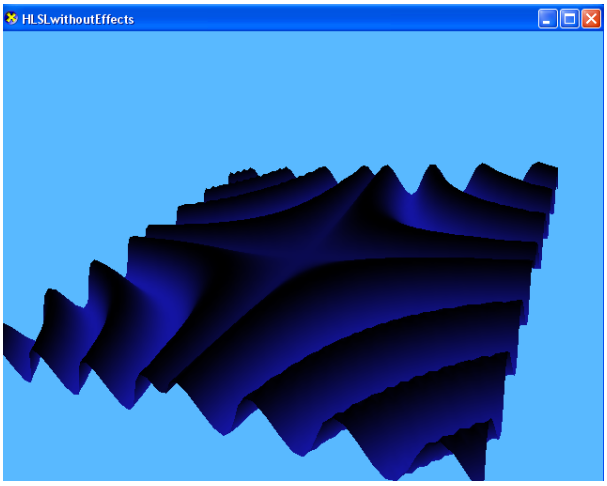
Todėl norint suprasti kaip rašyti spalvinimo efektų algoritmus, prieš tai reikia išnagrinėti labai daug jau sukurtų pavyzdžių ir perprasti juose taikomus metodus.

7.3. Galutinio projekto stovio aprašymas.

Šiame darbe, pasinaudojant patikimais šaltiniais, yra surinkta daug ir svarbios informacijos apie dinaminius spalvinimo efektus ir jų programavimui sukurtą aukšto lygmens Cg-HLSL programavimo kalbą:

- išsiaiškinta kas yra viršūnių ir pikselių dinaminiai spalvinimo efektai, kokias Direct3D grafinio konvejerio dalis jais galima pakeisti;
- keliais būdais išnagrinėtas Direct3D grafinis konvejeris;
- aprašyti dinaminių spalvinimo efektų atlikimui reikalingi HAL ir REF įrenginiai;
- apžvelgtos dinaminių spalvinimo efektų kūrimo priemones ir išanalizuotos jų taikymo galimybes;
- išsiaiškinta kokias HLSL spalvinimo efektų versijas palaiko konkreti techninė įranga;
- ištirta kokios dinaminių spalvinimo efektų aukšto lygmens programavimo kalbos yra populiariausios ir atlikta Cg-HLSL dinaminių spalvinimo efektų programavimo kalbos galimybių analizė;
- išnagrinėtas spalvinimo efektų ir Cg-HLSL programavimo kalbos galimybių apribojimas;
- išsiaiškinta kaip yra kompiliuojami HLSL ir Cg spalvinimo efektai;
- apžvelgti du HLSL spalvinimo efektų įterpimo į 3D programą būdai;
- išnagrinėta nemažai sukurtų pavyzdžių ir išsiaiškinti juose naudojami metodai, kuriuos panaudojau kurdama savo spalvinimo efektus:
  - *menulio_šviesa.rfx* sukurtas panaudojant RenderMonkey 1.6;

- *animacija.txt* sukurtas su Microsoft Visual Studio.NET 2003.

7.4. Darbo rezultatų analizė.

Šio darbo pradžioje ieškojau literatūros tik apie dinaminius spalvinimo efektus ir jų programavimo kalbas, tačiau kai pradėjau nagrinėti jų pavyzdžius ir pabandžiau parašyti savo spalvinimo efektą, supratau, kad vien tik šios informacijos nepakanka. Teko išsiaiškinti, kai kuriuos kompiuterinės grafikos ypatumus (pvz. kas yra primityvai, viršūnės, bendroji, peržiūros ar projekcijos matricos ir kt.), išsamiai išnagrinėti Direct3D grafinį konvejerį, o kai pagaliau suvokiau, kad norint atlikti spalvinimo efektus juos būtinai reikia įterpti į 3D programą, tai teko išmokti programuoti Direct3D, pasidomėti efektų (*effect*) kūrimu ir išbandyti specialias spalvinimo efektų kūrimo priemones (RenderMonkey 1.6 ir FX Composer).

Paprastai yra manoma, kad naudojant specialias priemones labai supaprastėja programavimas ir jomis gali naudotis mažiau žinių turintys vartotojai, tačiau šiuo atveju pačių spalvinimo efektų programavimas yra lygiai toks pats, nes vis tiek tenka rašyti viršūnių ir pikselių spalvinimo efektų kodą. Tiesa, čia kompiliatorius tiksliai nurodo klaidų vietas, yra pateikta nemažai pavyzdžių, kuriais galima pasinaudoti kuriant savo efektus ir norint pamatyti spalvinimo efektų rezultatą, nereikia rašyti 3D programos. Tačiau vis tiek (nors ir kitais būdais) tenka kurti tekstūrų objektus ir juos susieti su tekstūromis, aprašyti bendrąją, peržiūros, projekcijos matricas ir kt. kintamuosius, nustatyti objekto poziciją, normales, tekstūrų koordinates ir t.t., be to čia taip pat reikia suprasti efektų (*effect*) kūrimo ypatybes.

Šiame darbe iš pradžių buvau numačiusi atlikti HLSL spalvinimo efektų programavimo kalbos analizę, tačiau visai neseniai išsiaiškinau, kad HLSL ir Cg yra tos pačios spalvinimo efektų programavimo kalbos dvi skirtingos realizacijos (nors daugelyje literatūros šaltinių yra teigiama, kad Cg ir HLSL yra skirtingos programavimo kalbos), todėl šiame darbe visai netikėtai teko apžvelgti ne tik HLSL, bet ir Cg. Tiesa šiame darbe daugiau dėmesio skirta HLSL realizacijai (pavyzdžiai yra sukurti HLSL aplinkai) , nes dėl laiko stokos nebespėjau pakankamai gerai įsigilinti į Cg realizaciją.

Atlikdama šį darbą, pastebėjau, kad daugelis HLSL spalvinimo efektų kūrėjų skirtingai supranta ir aiškina Diret3D grafinį konvejerį, todėl buvo gana sunku nuspręsti, kuris aiškinimas yra teisingiausias. Stengdamasi nesuklysti, šiame darbe rėmiausi Microsoft DirectX (April 2006) dokumentacija ir Microsoft DirectX programuotojo Kris‘o Gray‘aus , dirbančio su Direct3D, knyga.

Šio darbo metu išnagrinėjau nemažai viršūnių ir pikselių spalvinimo efektų pavyzdžių (rastų [6],[7],[8],[1],[2],[3],[9],[11],[12] šaltiniuose) bei, pasiremdama jais, sukūriau keletą savų: vienas buvo sukurtas panaudojant RenderMonkey 1.6 spalvinimo efektų kūrimo priemonę, o kitas – Microsoft Visual Studio.NET 2003.

Taigi, šiame darbe yra surinkta ir išaiškinta pati svarbiausia informacija apie dinaminius spalvinimo efektus ir jų panaudojimą. Atliekant šį darbą, buvo išnagrinėta kas yra dinaminiai spalvinimo efektai ir kam jie naudojami, kokios yra dinaminių spalvinimo efektų programavimo kalbos ir kūrimo priemonės, atlikta Cg-HLSL aukšto lygmens spalvinimo efektų programavimo kalbos galimybių analizė, aptartas šios programavimo kalbos palaikymas šiuo metu populiariausiuose trimačiuose grafiniuose API, aptarti spalvinimo efektų įterpimo į 3D programą būdai, išnagrinėta nemažai spalvinimo efektų pavyzdžių ir du pavyzdžiai sukurti, prieduose yra atlikta HLSL apžvalga, bei HLSL spalvinimo efektų panaudojimas 3ds MAX 7 grafiniame pakete.

8. Išvados

Cg – HLSL yra puiki aukšto lygmens spalvinimo efektų programavimo kalba skirta grafiniuose procesoriuose vykdomų spalvinimo efektų kūrimui. Jos HLSL realizacija yra Microsoft'o Direct3D dalis, o Cg yra nepriklausoma nuo 3D programavimo interfeiso ir pilnai integruota į Direct3D ir OpenGL. Taigi šis kalbos lankstumas suteikia programuotojams galimybę dirbti su abiem dominuojančiomis 3D programinėmis sąsajomis ir pasirinkta operacine sistema, kadangi Cg yra realizuota su Windows, Linux, Mac OS. Cg-HLSL programavimo kalba sukurti spalvinimo efektai yra vykdomi pagrindinių vaizdo plokščių gamintojų (3Dlabs, ATI, Matrox, ir NVIDIA) sukurtuose grafiniuose procesoriuose. Nors su Cg-HLSL jau dabar galima sukurti įvairiausių spalvinimo efektų, tačiau dėl nepakankamai ištobulintos grafinės techninės įrangos, šiuo metu dar negalima išnaudoti visų šios programavimo kalbos galimybių.

Atliekant šios programavimo kalbos analizę ir jos panaudojimą trimatės grafikos sistemose buvo:

1. Išanalizuota, kas yra dinaminiai spalvinimo efektai ir kur jie naudojami;
2. Apžvelgtos dinaminių spalvinimo efektų kūrimo priemones ir išanalizuotos jų taikymo galimybes;
3. Išanalizuota, kokios dinaminių spalvinimo efektų programavimo kalbos šiuo metu yra populiariausios;
4. Atlikta pasirinktos dinaminių spalvinimo efektų programavimo kalbos galimybių analizė;
5. Atrinkti jau sukurti pavyzdžiai ir išnagrinėtas jų veikimas.
6. Sukurti du spalvinimo efektai.

9. Literatūros ir informacinių šaltinių sąrašas

1. Frank D. Luna., Introduction to.3D Game Programming with DirectX.9.0. 2003
2. James C. Leiterman., Learn Vertex and Pixel Shader Programming With Directx9. 2004
3. Kelly Dempski., Real time Rendering Tricks and Techniques in DirectX. 2002
4. Kris Gray., Microsoft DirectX 9 programmable graphics pipeline. 2003
5. T. Lūžienė, G. Lūža., Kompiuterinė grafika. 2006 ŠU
6. Wolfgang F. Engel., Beginning Direct3D® Game Programming 2nd Edition. 2003
7. Wolfgang F. Engel., Shader Programming Tips and tricks with Directx9. 2004
8. Wolfgang F. Engel., ShaderX2 Introductions and Tutorials with DirectX9. 2004
9. NVIDIA Developer Web Site, http://developer.nvidia.com, 2006 05 20
10. Microsoft Corporation, http://www.microsoft.com, 2006 05 03
11. DirectX SDK (2006 April) dokumentacija
12. NVIDIA SDK dokumentacija
13. NVIDIA Home, http://www.nvidia.com, 2006 05 20
14. 3Dlabs.com, http://www.3Dlabs.com, 2006 05 10
15. ATI Technologies Inc., http://www.ati.com 2006 05 10
16. C‘t Kompiuterinių technologijų žurnalas., Dr. Jörn Loviscach., Pikselių apdorojimas, 2006 Balandis

Anotacija

Pastaraisiais metais interaktyvios trimatės grafikos raidą labai paspartino atsiradusios dinaminių spalvinimo efektų aukšto lygmens programavimo kalbos ir jų kūrimo priemonės.

Šiame darbe yra surinkta ir išaiškinta svarbiausia informacija apie dinaminius spalvinimo efektus ir jų panaudojimą, išnagrinėtos jų kūrimo priemonės, atlikta Cg-HLSL programavimo kalbos galimybių analizė ir jos taikymas trimatės grafikos sistemose, apžvelgtos jos Cg ir HLSL realizacijos ir sukurta keletas dinaminių spalvinimo efektų pavyzdžių.

Šiauliai University
Faculty of Mathematics and Informatics
Department of Informatics

Master's of Science Final Work
Cg-HLSL case study and using in 3D applications
Author: Asta Margiene

Summary

Last year evolution of interactive 3D graphics became quicken when emerged high level programming language of interactive shaders and their creation tools.

In this project I found out and chose all information about interactive shaders and their practice. I explored interactive shaders creation tools, analyzed Cg-HLSL programming language and how to practice it for three-dimensional graphics system. In this project I reviewed Cg and HLSL implementation and I built few examples of interactive shaders effects.

PRIEDAI

1 priedas. 3D žodynas

3D: trimatis, erdvinis, turintis plotį, aukštį ir storį (gylį).

3D Scene: Trimačių objektų, kuriuos reikia pavaizduoti ekrane, visuma.

Aliasing: taškinės (rastrinės) grafikos savybė įstrižas linijas atkurti "laiptuotai".

Alpha Buffer (alfa buferis): rezervuota video atminties sritis, kuriame saugoma informacija apie objektų skaidrumą.

Alpha Blending (alfa suliejimas): 3D vaizdo kūrimo (Rendering metu atliekama operacija, kurios metu, panaudojant Alpha Channel informaciją, perteikiamas objektų skaidrumas. Taip gana realistiškai galima pavaizduoti vandenį, stiklą, rūką, debesis.

Animation (animacija): Judančio (besikeičiančio laike) vaizdo kūrimo ir atvaizdavimo procesas.

Anti-Aliasing (gludinimas): Vaizdo apdorojimo operacija, kurios metu sugludinamas taškinės grafikos laiptuotumas.

API (Application Programming Interface): Sąsaja (interface) tarp programinės įrangos (software) ir techninės įrangos (hardware)**;** mūsų atveju - tarp grafinės programos ar žaidimo ir konkrečios vaizdo plokštės su savo tvarkykle (driver). API dėka programos vienodai veikia su visomis grafinėmis plokštėmis, "suprantančiomis" tą API standartą.

Bilinear Filtering: Tekstūrų dengimo (texture mapping) būdas, kai tekstūros labiau nutolusiems objektams sumažinamos (kiekvienam mažesnės tekstūros taškui imamas keturių gretimų didesnės tekstūros taškų spalvos vidurkis).

Blending (maišymas): Vaizdo apdorojimo operacija, taško spalvai nustatyti sumaišant kelių kitų vaizdo taškų spalvas.

Clipping (iškarpymas): Nematomų (į ekrano rėmus nepatenkančių) objektų ir jų dalių pašalinimas iš tolesnių skaičiavimų.

Direct3D: Kompanijos Microsoft sukurto kompiuterinės grafikos API (Application Programming Interface "DirectX" dalis, skirta operacijoms su 3D grafika. Trūkumai - vartojama tik "MS Windows" terpėje, o pats DirectX/Direct3D programavimas yra gana sudėtingas, lyginant su kitais API.

Dithering: Daugiau spalvų turinčio vaizdo (pvz. "TrueColor") konvertavimo į mažiau spalvų (pvz. 256) turintį metodas, kai tam tikrai spalvai išgauti vartojamas kitas spalvas turinčių taškų mišinys.

Flat Shading: Į poligonus (trikampius) suskaidyto ir tekstūromis padengto trimačio objekto vaizdo apdorojimo būdas, kai visam trikampiui suteikiama ta pati spalva arba tas pats apšviestumas. Apvalūs ar lenkti paviršiai lieka pastebimai "briaunuoti".

Fog (rūkas): Rūko, miglos, drumsto vandens pavaizdavimas. Vartojama Alpha Blending operacija, ir nutolę objektai tampa tamsesniais ar įgyja ryškesnį atspalvį, nei arti esantys.

Frame Buffer (kadro buferis): Videoatminties dalis, kurioje patalpinta grafinė informacija rodoma monitoriuje.

Gouraud Shading (Smooth Shading): Į trikampius suskaidyto ir tekstūromis padengto trimačio objekto tolesnio apdorojimo būdas, kuriuo sugludinamos skaidymo metu atsiradusios briaunos. Šis metodas numato, jog kiekvienas trikampio kampas turi atskirą spalvą (ar apšviestumo reikšmę), o trikampio vidaus taškų spalva apskaičiuojama, atsižvelgiant į jų atstumą nuo kampų

Graphics Pipeline (grafinis konvejeris): Objektų, apibūdintų geometrinėmis koordinatėmis, pavaizdavimo rastriniame (susidedančiame iš taškų) ekrane eiga.

Grafinis API: Grafiniams API *(Application Programming Interface)* priskiriamos programavimo kalbose grafinio vaizdo kūrimui naudojamos funkcijų arba klasių bibliotekos. Jiems nepriskiriamos dvimačių grafinių vartotojo sąsajų (GUI - Graphical User Interface) kūrimo priemonės.

Interactive graphic (interaktyvioji grafika): Prie interaktyviosios grafikos priemonių priskirtinos funkcijų ir klasių bibliotekos, programavimo sąsajos (*application programming interface*, arba API) ir kitos panašios priemonės nuo vartotojo veiksmų priklausančiam vaizdui kurti.

Hidden Surface Removal (uždengtų paviršių pašalinimas): Nematomų (uždengtų vieni kitais) objektų ir jų dalių pašalinimas iš tolesnių skaičiavimų. Vienas iš būdų - naudojant vadinamą Z buferį.

Lighting (apšvietimas): Graphics Pipeline dalis, kurios metu apskaičiuojamas objektų apšviestumas, atsižvelgiant į jų tarpusavio bei šviesos šaltinių padėtį. Apšviestumo reikšmė paprastai priskiriama kiekvienam trikampio kampui po to, kai 3D objektai suskaidomi į poligonus (trikampius). Skirtingai nuo vėlesnio Shading Lighting operacija atliekama su vektoriniais objektais, jų dar nekonvertavus į rastrinę (taškinę) grafiką.

Material (medžiaga): visuma paviršiaus parametrų, apibrėžiančių objekto išorinį vaizdą vizualizacijos metu.

OpenGL: Koncerno Silicon Graphics sukurta API sąsaja, skirta visų pirma galingoms grafinėms darbo stotims, bet vartojama ir asmeniniuose kompiuteriuose pav. 3D žaidimų akceleravimui. Skirtingai nuo DirectX/Direct3D , tai - visoms operacinėms sistemoms skirtas standartas.

Page Flipping (kadrų sukeitimas): Greitas tariamo dviejų vaizdo kadrų (buferių) sukeitimo vietomis metodas, vartojamas kompiuterinėje animacijoje. Viename iš videokortos registrų yra nurodytas monitoriaus ekrane pateikiamo vaizdo (Display Buffer pradžios adresas. Į šį registrą paeiliui rašanti pirmo jo ir antrojo buferio adresus, ekrane matysis atitinkamai pirmajame ir antrajame įrašyta grafinė informacija.

Perspective Correction: Tekstūrų dengimo metu taikomi metodai, kuriais koreguojamas įstrižų (ne stačiu kampu žiūrėtojo atžvilgiu esančių) paviršių dengimas.

Phong Shading: Matematiko Phong Bui-Tong sumąstytas į trikampius suskaidyto ir tekstūromis padengto trimačio objekto briaunų sugludinimo būdas, kurio metu atsižvelgiama ne tik į trikampio kampų spalvą ir apšviestumą, bet ir į paviršiaus atspindžio koeficientą.

Pixel (pikselis): mažiausias rastrinės (taškinės) grafikos elementas t.y. taškas.

Polygon (daugiakampis): Daugiakampis (beveik visada - trikampis), į kuriuos, siekiant supaprastinti skaičiavimus, suskaidomi 3D objektai.

Rasterization (rasterizacija): Graphics Pipeline dalis, kurios metu 3D objektų scena konvertuojama į 2D ir toliau apdorojama kaip rastrinė (taškinė) grafika. Svarbiausios rasterizacijos metu atliekamos operacijos yra Scan Conversion Texture Mapping Shading.

Rendering (vizualizavimas): 3D scenos atvaizdavimui 2D ekrane (rasterizacijai) reikalingų veiksmų ir skaičiavimų seka.

Scaling (mastelio keitimas): Objekto padidinimas ar sumažinimas, nekeičiant jo padėties erdvėje ar plokštumoje bei kitų jo savybių.

Scan Conversion: Į poligonus (trikampius) suskaidytų 3D objektų projektavimas į rastrinį (susidedantį iš taškų) paveikslėlį.

Set-Up Engine: API dalis, paruošianti ir siunčianti į grafinę plokštę jai reikalingą informaciją.

Local Space (vietinė erdvė): vietinė erdvė dar vadinama modeliavimo erdve yra koordinačių sistema, kurioje nurodome objekto trikampių sąrašą. Vietinė erdvė naudojama kadangi ji supaprastina modeliavimo procesą. Modelio kūrimas jo vietinėje koordinačių sistemoje yra lengvesnis, nei jo kūrimas bendroje koordinačių sistemoje. Kadangi vietinė erdvė leidžia konstruoti modelį neatsižvelgiant į jo padėtį ir dydį su kitais objektais

World Space (bendra erdvė): sukonstravus keletą skirtingų modelių jų vietinėse koordinačių sistemose, reikia juos patalpinti į vieną sceną, turinčią bendrą (pasaulinę) koordinačių sitemą. Objektai iš vietinių erdvių transformuojami į bendrąją, panaudojant pasaulinę transformaciją, kuri paprastai susideda iš perskaičiavimų, pasukimų, dydžio pakeitimo (scaling) operacijų, kurios nustato padėtį, orientaciją ir modelio dydį scenoje.

Specular Highlight: Šviesos atsispindėjimas nuo matinio paviršiaus.

Specular reflection (veidrodinis atspindys): Šviesos atsispindėjimas nuo lygaus (veidrodinio) paviršiaus.

Texture (tekstūra): Rastrinės (taškinės) grafikos paveikslėlis, naudojamas 3D objektų paviršių padengimui (Texture Mapping. Pav., medžio kamieno tekstūra būtų žievės gabalėlis.

Texture mapping (dengimas tekstūra): Rasterizacijos metu vykdomas 3D objektų dengimas tekstūromis.

Transformation (transformacija): Graphics Pipeline dalis, kurios metu vykdomos 3D objektų koordinačių perskaičiavimai, susiję perėjimas iš vienos koordinačių sistemos į kitą. Taip pat ir kitos su šiomis koordinatėmis atliekamos matematinės operacijos.

> **World transformation (pasaulinė transformacija):** modelio viršūnės yra transformuojamos iš vietinės koordinačių sistemos į bendrą koordinačių sistemą, kurią naudoja visi scenos objektai.
>
> **View transformation (peržiūros transformacija):** šiame etape, viršūnės yra orientuojamos atsižvelgiant į kamerą. Tai yra parenkama scenos peržiūros vieta (point-of-view) ir pasaulinės erdvės koordinatės yra perkeliamos į kameros matymo sritį, pakeičiant pasaulio (bendrą) erdvę į kameros erdvę.
>
> **Projection transformation (projekcijos transformacija)** – Šioje grafinio konvejerio dalyje objektų mastelis yra keičiamas priklausomai nuo atstumo iki žiūrovo, tam kad susidarytų scenos gylio įspūdis; arčiau esantys objektai vaizduojami didesni už nutolusius.

View point (Eye point, camera): Žiūrėjimo taškas; "stebėtojo" koordinatės, kurių atžvilgiu daromi visi 3D scenos objektų koordinačių apskaičiavimai.

Z-Buffer (Z-buferis): Atminties sritis, dydžiu lygi Frame Buffer, kurioje saugoma informacija apie jau konvertuotų į 2D objektų trečiąją koordinatę - tariamą nuotolį nuo stebėtojo (View Point Z buferis vartojamas nematomų plokštumų pašalinimui (Hidden Surface Removal)

2 priedas. CD turinys

1. Dokumentacija:
   1.1. Darbo_aprašymas;
   1.2. HLSL_apžvalga;
   1.3. HLSL_spalvinimo_efektų_naudojimas_3ds_MAX_7;
   1.4. CD_turinys;
   1.5. 3D_žodynas;
2. Spalvinimo efektų kūrimo įrankiai:
   2.1. DirectX SDK (2006 April);
   2.2.RenderMonkey 1.6;
3. Pavyzdžiai:
   3.1.Animacija;
   3.2.Menulio_šviesa.

3 priedas. HLSL apžvalga *(http://www.gamedev.ru/articles/?id=10109&page=2)*

Duomenų tipai

HLSL palaiko įvairius duomenų tipus(ir paprastus skaliarus, ir kompleksinius – vektorius ir matricas).

Skaliariniai tipai

Tipas	Reikšmė
bool	true arba false
int	32-bit signed integer
half	16-bit floating point value
float	32-bit floating point value
double	64-bit floating point value

Vektoriai

Vektorius galima aprašyti keliais būdais:
vector <type, size> (size – vektoriaus dydis , type – skaliarinis tipas)
typeN (N- vektoriaus ilgis, type – skaliarinis tipas)
Pavyzdžiui: float4 Vector;
type Vector[N] (N- vektoriaus ilgis, type – skaliarinis tipas)

Matricos

Kaip ir vektoriai, matricos gali turėti visus skaliarinius tipus. Jos gali būti aprašomos taip:
typeMxN (type – duomenų skaliarinis tipas, M ir N – matricos dydis)
matrix<type, M,N> (type – duomenų skaliarinis tipas, M ir N – matricos dydis)

Priėjimas prie matricos elementų:

```
mat[2]    (tas pats kaip mat.m20m21m22m23)
mat[2].w  (tas pats kaip mat.m23)
mat[2][3] (tas pats kaip mat.m23)
```

Kintamieji

Kintamieji gali būti static arba extern. Kiekvienas extern tipo kintamasis, aprašytas už spalvinimo efekto, gali būti pakeičiamas per API. Statinis kintamasis yra naudojamas tik spalvinimo efekto viduje ir per API nepasiekiamas.

Pavyzdžiui:

```
extern float a;
const  float b;
static float c;
```

```
float d;
```

Kintamieji a ir d pasiekiami per API funkciją Set*ShaderConstant*() ir juos spalvinimo efektas gali keisti. Kintamasis b taip pat pasiekiamas per Set*ShaderConstant*(), tačiau spalvinimo efektas negali pakeisti konstantos reikšmės. Kintamasis c nepasiekiamas per API, tačiau gali būti keičiamas spalvinimo efekte.
Kintamųjų inicializacija atliekama taip pat, kaip ir C kalboje:

```
float2x2 mat = {1.0f, 0.0f, // 1-ma eilė
                1.0f, 2.0f}; // 2-ra eilė
float4 pos = {1.0f, 0.5f, 12.0f, 1.0f};
float f = 0.01f;
```

Struktūros

HLSL palaiko sruktūras. Pavyzdžiui:

```
struct VS_OUTPUT
{
  float4 Pos;
  float3 View;
};
```

Struktūros gali būti sudarytos iš skaliarinių dydžių, vektorių, matricų ir kitų struktūrų.

Operatoriai

Operacijos	Operatoriai
Aritmetinės	-, +, *, /, %
Didėjimo, mažėjimo	++, --
Loginės	&&, ||, ?:
Unarinės	!, -, +
Palyginimo	<, >, <=, >=, ==, !=
Priskyrimo	=, -=, +=, *=, /=
Kablelis	,
Struktūros narys	.
Masyvo narys	[indeksas]

Šakojimasis

if (expr) then statement [else statement]

Ciklai

do statement while (expr);

while (expr) statement;

for (expr1;expr2;expr3) statement

Funkcijos

Panašios į C kalbos funkcijas, tačiau nepalaikoma rekursija. Funkcijų parametrai gali būti semantiškai surišti su duomenimis.
Programuojant HLSL galima naudotis matematinėmis funkcijomis:

abs(x)	*Sveikojo argumento modulis (per-component).*
acos(x)	*Gražina arkkosinusą kiekvieno komponento x. Kiekvienas komponentas turi būti diapozone [-1, 1].*
asin(x)	*Gražina arksinusą kiekvieno komponento x. Kiekvienas komponentas turi būti diapozone [-pi/2, pi/2].*
atan(x)	*Gražina arktangentą kiekvieno komponento x. Kiekvienas komponentas turi būti diapozone [-pi/2, pi/2].*
ceil(x)	*Gražina mažiausią sveiką skaičių, kuris yra didesnis ar lygus x.*
cos(x)	*Gražina kosinusą x.*
cosh(x)	*Gražina hiperbolinį kosinusą x.*
degrees(x)	*Konvertuoja x iš radianų į laipsnius.*
distance(a, b)	*Gražina atstumą tarp dviejų taškų a ir b.*
dot(a, b)	*Gražina dviejų vektorių a ir b dot produktą*
floor(x)	*Gražina didžiausią sveiką skaičių, kuris yra mažesnis arba lygus x.*
fwidth(x)	*Gražina abs(ddx(x))+abs(ddy(x)).*
len(v)	*Vektorinį ilgį.*
length(v)	*Gražina vektoriaus ilgį v.*
lerp(a, b, s)	*Gražina a + s (b - a).*
log(x)	*Gražina logaritmą x.*
log10(x)	*Gražina dešimtainį logaritmą x.*
mul(a, b)	*Gražina matricų a ir b daugybą.*
normalize(v)	*Gražina normalizuotą vektorių v.*
pow(x, y)	*Gražina xy.*
radians(x)	*Konvertuoja x iš laipsnių į radianus*
rsqrt(x)	*Gražina 1 / sqrt(x).*

sin(x)	*Gražina sinus x.*
sincos(x, out s, out c)	*Gražina sinus ir kosinus x.*
sinh(x)	*Gražina hiperbolinį sinusą x*
sqrt(x)	*Gražina kvadratinę šaknį (per-component).*
step(a, x)	*Gražina (x = a) ? 1 : 0.*
tan(x)	*Gražina tangentą x*
tanh(x)	*Gražina hiperbolinį tangentą x*

Tekstūrų naudojimas pikselių spalvinimo efektuose

Jeigu jūs norite pikselių spalvinimo efekte panaudoti tekstūras, tai būtinai reikės aprašyti extern tipo kintamąjį sampler. Faktiškai sampler apibrėžia tekstūrą, ir panaudojant sampler galima nustatyti spalvas pikselių spalvinimo efekte. Žemiau yra aprašytos funkcijos, kurios gražina tekstūros spalvą:

tex1D(s, t)	*Skaitoma iš 1D tekstūros. s - sampler. t - skaliaras.*
tex1D(s, t, ddx, ddy)	*Skaitoma iš 1D tekstūros, su išvestinėmis. s - sampler. t, ddx, ir ddy - skaliarai.*
tex1Dproj(s, t)	*Skaitoma iš 1D projekcinės(projective) tekstūros. s - sampler. t - 4D vektorius. t dalosi į t.w prieš funkcijos įvykdymą.*
tex1Dbias(s, t)	*Skaitoma iš 1D tekstūros su poslinkiu, s - sampler, t – 4D vektorius. Mip-lygis pasikeičia į t.w iki tol,kol prasideda paieška*
tex2D(s, t)	*Skaitoma iš 2D tekstūros. s – sampler, t – 2D vektorius.*
tex2D(s, t, ddx, ddy)	*Skaitoma iš 2D tekstūros, su išvestinėmis. s - sampler. t – 2D tekstūros koordinatės. ddx, ddy- 2D vektoriai.*
tex2Dproj(s, t)	*Skaitoma iš 2D projekcinės(projective) tekstūros. s - sampler. t - 4D vektorius. t dalosi į t.w prieš funkcijos įvykdymą.*
tex2Dbias(s, t)	*Skaitoma iš 2D tekstūros su poslinkiu, s - sampler, t – 4D vektorius. Mip-lygis pasikeičia į t.w iki tol,kol prasideda paieška.*
tex3D(s, t)	*Skaitoma iš 3D tekstūros. s – sampler, t – 3D vektorius*
tex3D(s, t, ddx, ddy)	*Skaitoma iš 3D tekstūros, su išvestinėmis. s - sampler. t – 2D tekstūros koordinatės. ddx, ddy- 3D vektoriai.*
	Skaitoma iš 3D projekcinės(projective) tekstūros. s - sampler. t - 4D

	vektorius. *t dalosi į t.w prieš funkcijos įvykdymą.*
tex3Dbias(s, t)	*Skaitoma iš 3D tekstūros su poslinkiu, s - sampler, t – 4D vektorius.* *Mip-lygis pasikeičia į t.w iki tol,kol prasideda paieška.*
texCUBE(s, t)	*Skaitoma iš kūbinės tekstūros. s – sampler,* *t – 3D tekstūrų koordinatės.*
texCUBE(s, t, ddx, ddy)	*Skaitoma iš kūbinės tekstūros. s - sampler.* *t –3 D tekstūrų koordinatės. ddx, ddy - 3D vektoriai.*
texCUBEproj(s, t)	*Skaitoma iš kubinės projekcinės(projektive) matricos. s – sampler, t - 4D vektorius .* *t dalosi į t.w prieš funkcijos įvykdymą.*
texCUBEbias(s, t)	Skaitoma iš kūbinės tekstūros. *s – sampler, t - 4D vektorius.* *Mip-lygis pasikeičia į t.w iki tol,kol prasideda paieška.*

HLSL programavimo kalboje yra naudinga konstantų savybė – konstantų surišimas su registrais, pavyzdžiui:
sampler tex : register (s0);
API tai atliekama taip: RenderDevice->SetTexture(0,texture0);

Įvesties ir išvesties parametrai viršūnių ir pikselių spalvinimo efektuose

Viršūnių ir pikselių spalvinimo efektai turi dviejų tipų įvesties duomenis: varying ir uniform.
Uniform — pastovūs, skirti daugkartiniam naudojimui duomenys spalvinimo efektuose. Aprašyti uniform tipo duomenis galima dviem būdais:
1)Aprašyti duomenis kaip extern tipo kintamąjį, pavyzdžiui:

```
float f;

float4 main () : COLOR
{
 return f;
}
```

2)Aprašyti duomenis pasinaudojant uniform. Pavyzdžiui:

```
float4 main (uniform float4 f) : COLOR
{
 return f;
}
```

Uniform kintamieji naudojami pasinaudojant konstantų lentele. Konstantų lentelėje yra visi registrai, kurie yra pastoviai naudojami spalvinimo efektuose.
Varying — duomenys, kurie yra unikalūs kiekvienam spalvinimo efekto iškvietimui. Pavyzdžiui: pozicija, normalė ir kt. Viršūnių spalvinimo efekte aprašomi varying duomenys, kurie yra perduodami iš viršūnių buferio, o pikselių spalvinimo efekte –interpoliuoti duomenys, gauti iš viršūnių spalvinimo efekto.
Pagrindiniai semantiniai įvesties tipai:

POSITIONn	*Pozicija.*
BLENDWEIGHTn	*Svorio koeficientas*

BLENDINDICESn	*Svorio matrcos indeksas*
NORMALn	*Normalė.*
PSIZEn	*Taško dydis.*
COLORn	*Spalva.*
TEXCOORDn	*Tekstūrų koordinatės.*
TANGENTn	*Tangentas*
BINORMALn	*Binormalė.*
TESSFACTORn	*Tesiliacijos faktorius*

Varying duomenų naudojimas pikselių spalvinimo efekte nustato vieno pikselio būseną. Pagrindiniai semantiniai įvesties tipai:

COLORn	*Spalva*
TEXCOORDn	*Tekstūrų koordinatės.*

n – nurodo semantinio tipo numerį, pavyzdžiui: **TEXCOORD1, NORMAL0**.
Viršūnių spalvinimo efekto išvesties duomenys:

POSITION	*Pozicija*
PSIZE	*Taško dydis*
FOG	*Viršūnės rūko koeficientas.*
COLORn	*Spalva.*
TEXCOORDn	*Tekstūrų koordinatės.*

Pikselių spalvinimo efekto išvesties duomenys:

COLORn	*Spalva*
DEPTH	*Gylis*

Viršūnių ir pikselių spalvinimo efektų pavyzdys (Dizzi efektas)

Šio efekto esmė yra animuotas, į spiralę susuktų ratų, vaizdas.

Viršūnių spalvinimo efekto algoritmas:

1. Randame viršūnės poziciją.
2. Išsaugome viršūnės poziciją kaip tekstūros koordinates (TEXCOORD0) (jos bus perduodamos pikselių shader‘iui). T.y. ivesties duomenys bus POSITION, o išvesties - TEXCOORD0.

Pikselių shader‘io algoritmas:

1. Gauname interpoliuotus duomenis (TEXCOORD0 – t.y. tekstūrų koordinates).
2. Apskaičiuojame arktangentą iš tekstūrų koordinačių x ir y.
3. Apskaičiuojame tekstūrinių koordinačių spindulį. $R^2=(x-a)^2+(y-b)^2$. Patogumo dėlei paimsime, kad a = 0, b = 0.
4. Pasinaudojant formule sin(kampas + spindulys) gražiname spalvą.

Tam, kad būtų galima gauti animuotą paveiksliuką, reikia dar įvesti laiko parametrą, kuris suks žiedą. Dar galima įvesti parametrą num_ring, kuris nustato besisukančių ratų kiekį. Galutinis gaunamos formulės vaizdas bus toks:
spalva=sin(kampas + žiedų_kiekis * spindulys + laikas).

Dizzi efekto pikselių spalvinimo efektas:

```
float4 main(float2 texCoord: TEXCOORD0) : COLOR
{
  float ang = atan2 (texCoord.x, texCoord.y);
  float rad = dot (texCoord, texCoord);

  return 0.5*(1 + sin (ang + rings * rad + time));
};
```

Viršūnių spalvinimo efektas:

```
float4x4 mvp;
//Išvestis duomenų struktūra. Šie duomenys bus perduodami į rasterizatorių, o po to //pikselių shader‘iui.
struct VS_OUTPUT
{
  float4 Pos: POSITION;
  float2 TexCoord: TEXCOORD0;
};
//Dėl įvesties duomenų reikalinga tik objekto pozicija, kuri randama viršūnių buferyje VS_OUTPUT main(float4 Pos: POSITION)
{
  VS_OUTPUT Out;
  Out.Pos = mul(mvp,Pos);
  //Prilyginame viršūnės koordinates, tekstūrų koordinatėms.
  Out.TexCoord = normalize(Pos.xy);
```

```
  return Out;
}
```

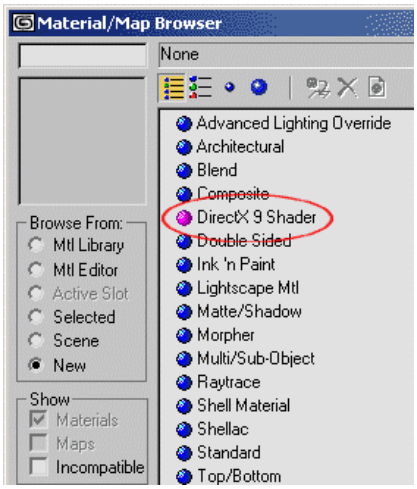
4 priedas. HLSL spalvinimo efektų panaudojimas 3ds MAX 7

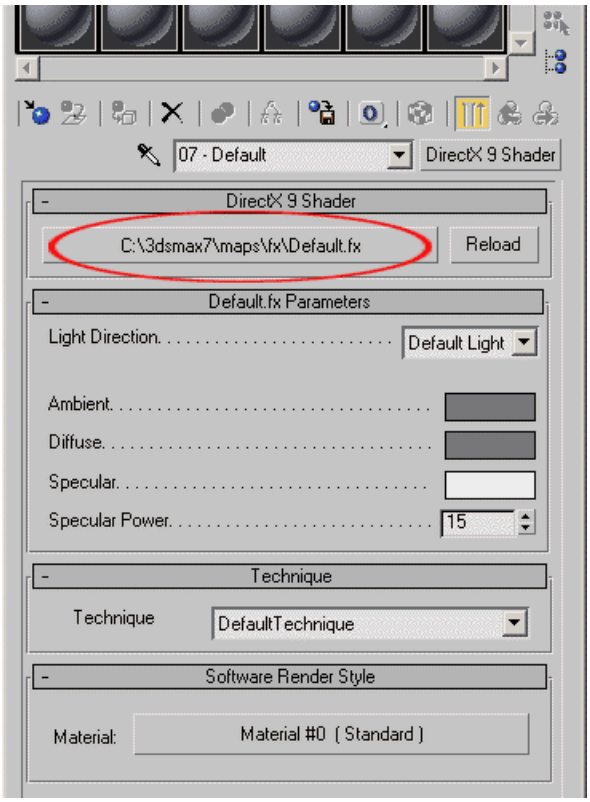
Video žaidimų dailininkai realaus laiko 3D objektus modeliuoja pasinaudodami tokiomis galingomis trimatės grafikos kūrimo sistemomis kaip 3ds MAX, Maya, Lightwave ir kt. Anksčiau dailininkas galėdavo tik įsivaizduoti kaip atrodys jo kūrinys žaidime, nes pamatyti tikrą modelio vaizdą buvo galima tik eksportavus ir paleidus jį žaidime, o tam reikia programuotojų pagalbos. Atsiradus HLSL spalvinimo efektų kūrimo priemonėms, žaidimų dailininkai gali pamatyti savo modelius tokius, kaip jie atrodys žaidime. Nuo 3DSMax 6 versijos atsirado galimybė paleisti FX formato HLSL spalvinimo efektus realaus laiko peržiūros srityje (viewport). Kadangi žaidimuose taip pat yra naudojamas FX formatas (Microsoft ir nVIDIA sukurtas standartas), todėl dailininkas galės panaudoti 3ds MAX'e tuos pačius spalvinimo efektus kaip ir žaidime.

HLSL spalvinimo efektai gali būti įterpiami panaudojant specialų materialo (material) tipą. Po to jie yra uždedami ant reikiamų modelių, o gautas rezultatas bus matomas peržiūros srityje (viewport).

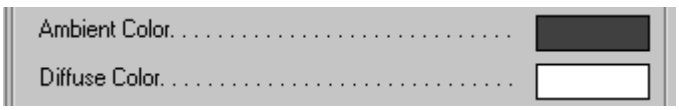
- 3DS Max 6 arba 3DS Max 7 (naudingiau 3DS Max 7) programose atsidarykite modelį, kuriam norite uždėti spalvinimo efektą.
- Atsidarykite Material Editor langą
- Pasirinkite vieną nepanaudotą Material paletės apskritimą ir įrašykite reikiamoje vietoje būsimo materialo pavadinimą.
- Pažymėkite peržiūros srityje modelį ir paspauskite „Assign Material to Selection" mygtuką, kad materialas būtų uždėtas ant modelio.
- Dabar reikia pakeisti šį materialą į DirectX 9 spalvinimo efektą. Paspauskite paveikslėlyje nurodytą mygtuką

- Pasirinkite „Direct X 9 Shader", kaip parodyta paveikslėlyje ir paspauskite OK.



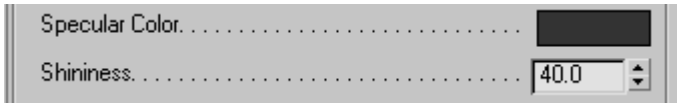


- Pagal nutylėjimą bus užkrautas „default.fx“ spalvinimo efektas. Jūs galite naudoti ir jį, tačiau internete galima rasti ir tinkamesnį, o jei mokate, parašykite savo.
- HLSL spalvinimo efektai turi spalvas, tekstūras ir reikšmes, kurias galima pakoreguoti. Pirmieji nustatymai, kuriuos galite atlikti yra "Ambient Color" ir "Diffuse Color“. Paspaudus ant spalvoto stačiakampio, atsidaro spalvų paletė ir jūs galite pasirinkti jums tinkančią.“ Ambient Color“ keičia visą modelį, nepaisant scenos šviesų. „Diffuse Color“ paveikia modelį tik tose vietose, kurios yra apšviestos šviesos šaltinio.



- „Diffuse Texture“ nustato paviršiaus spalvą, kiekvienam pikseliui. Paspauskite „None“ ir pasirinkite paveiksliuką, kuris bus panaudotas kaip difuzinė tekstūra.

- „Specular Color“ . daugelis objektų turi pilką atspindį, tačiau galima pasirinkti kokia tik norite atspindžio spalvą. „Shininess“ padarys jūsų atspindį ryškesnį.



- „Light position“ iškrentančiame meniu yra sudėti visi scenos šaltiniai.

- „ Technique“. Kartais spalvinimo efektai turi keletą metodų ar veikimo būdų. Pavyzdžiui programuotojas gali įterpti naujausios versijos spalvinimo efektą ir senesnės versijos spalvinimo efektą. Šiame iškrentančiame meniu galima pasirinkti, kurį naudoti.