KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Arnoldas Vaičikauskas

GLSL specialiųjų efektų šeiderių komplekto sukūrimas

Bakalauro darbas

Vadovas
prof. Rytis Maskeliūnas

KAUNAS, 2018

Turinys

Terminų ir santraukų žodynas

Šeideris	Programa vykdoma vaizdo plokštės procesoriuose (angl. shader).
GLSL	Šeiderių programavimo kalba naudojama OpenGL technologijos (angl. GL Shading Language).
HDR	Pikselių reikšmių aprašymui naudojami realieji o ne natūralieji skaičiai (angl. High Dynamic Range).
sRGB	Spalvų erdvė atitinkanti žmogaus akies veikimą. (angl. standard Red Green Blue.)

1 Įvadas

1.1 Darbo problematika ir aktualumas

1.2 Darbo tikslas ir uždaviniai

Darbo tikslas - sukurti specialiuosius efektus naudojant vaizdo plokščių programavimo technologijas.

Darbo uždaviniai:

1. išanalizuoti technologijas naudojamas 3D vaizdo efektų kūrimui;
2. sukurti 3D atvaizdavimo sistemą;
3. implementuoti pasirinktus grafinius efektus;
4. ištestuoti sukurtą sistemą ir efektus;

2 Analizė

2.1 Vaizdo plokščių programavimo technologijos

Grafiniams efektams implementuoti naudojamos vaizdo plokštės. Jos optimizuotos dirbti su dideliais duomenimis lygiagrečiai. Dėl to vaizdo plokštės turi žymiai daugiau branduolių nei centrinis kompiuterio procesorius. Grafikos atvaizdavimui egzistuoja aibė technologijų: Direct3D, OpenGL, Mantle, Vulkan, Metal. Visos šios technologijos turi savų privalumų ir trūkumų.

Direct3D – Naujausia versija yra Direct3D 12. Naujos versijos nėra suderinamos su senomis. Sukurta „Microsoft“ ir veikia tik ant „Windows“ operacinių sistemų. Versijos prieinamumas priklauso nuo vaizdo plokštės galimybių ir nuo pačios operacinės sistemos versijos. Direct3D 12 palaiko tik „Windows 10“ operacinė sistema. Šeiderių programavimo kalba HLSL (angl. „High Level Shading Language“).

OpenGL – Naujausia versija yra 4.6. Sukurta „Khronos“ grupės Priešingai nei Direct3D naujos versijos dažnai būna tik plėtiniai ankstesnės versijos, tačiau su keletą niuansų. OpenGL turi dvi pagrindines konteksto konfigūracijas: branduolio (angl. „Core profile“) ir suderinamumo (angl. „Compatibility profile“). Branduolio kontekstas neleidžia naudoti labai senų, neefektyvių ir šiuolaikinių vaizdo plokščių veikimo neatitinkančių funkcijų. Teoriškai naudojant branduolio kontekstą vaizdo plokštės tvarkyklė gali labiau optimizuoti OpenGL komandų vykdymą, nes jai nereikia tikėtis aparatinei įrangai neefektyvių veiksmų. OpenGL yra tiesiog specifikacija, todėl ją gali implementuoti visi vaizdo plokščių gamintojai ant visų operacinių sistemų. Tai yra gana didelis privalumas. Kadangi OpenGL specifikacija yra gana plati ir susideda iš daugelio versijų yra situacijų kai specifikacija nėra implementuota netaisyklingai. Ši problema gerai žinoma grafikos programuotojams, todėl kai kurios grafinės programos naudoja aparatinės įrangos, vaizdo plokštės tvarkyklės versijos ir OpenGL funkcionalumo nesuderinamumo sąrašą, pagal kurį programos veikimo metu gali išvengti klaidų. Šeiderių programavimo kalba GLSL (angl. „GL Shading Language“).

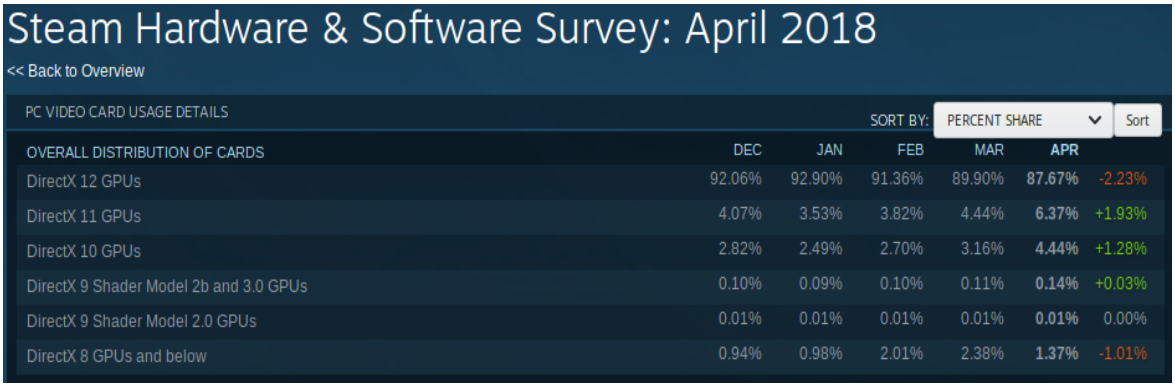
Mantle – Technologija sukurta AMD, siekianti sumažinti komunikaciją tarp programos ir vaizdo plokštės tvarkyklės. Šiuo metu nebetobulinama ir nebenaudojama, tačiau šis projektas nebuvo visiška nesėkmė, nes jis labai prisidėjo prie Vulkan specifikacijos ir implementacijos.

Vulkan – Kaip ir OpenGL yra specifikacija aprašanti sąsają su vaizdo plokštės tvarkykle. Tačiau priešingai nei OpenGL, ši specifikacija yra žymiai mažesnė ir ją lengviau visą implementuoti taisyklingai vaizdo plokščių tvarkyklių kūrėjams. Sukurta „Khronos“ grupės ši technologija fokusuojasi į modernios aparatinės įrangos veikimą ir suteikia grafikos programuotojui galimybes jas efektyviau išnaudoti. Ši technologija savo funkcionalumu labai panaši į Direct3D 12. Šeideriai turi būti SPIR-V formatu. SPIR-V tai binarinė šeiderio kalbos reprezentacija, tai reiškia, kad šeiderius galima rašyti bet kokia kalba, tačiau juos reikia sukompiliuoti į šį formatą. Plačiausiai naudojama GLSL šeiderių kalba.

Metal – Tai dar viena technologija leidžianti išnaudoti grafikos plokštės galimybes, tačiau prieinama tik „Apple“ sukurtose operacinėse sistemose. Naujausia versija Metal 2. Šeiderių programavimo kalba - „Metal Shading Language“.

Kadangi vienas iš projekto reikalavimų yra Linux operacinės sistemos palaikymas, tai tinkamiausia technologija yra OpenGL. Versijos pasirinkimas nulemia aparatinę įrangą, kurią

palaikys kuriama programa. Kad priimti geresnį sprendimą, galima atsižvelgti į 2018 metų „Steam“ aparatinės įrangos apklausą (žr. pav 1.). Joje galima matyti labiausiai naudojamų vaizdo plokščių ir jų grafikos galimybių lygius.



Steam Hardware & Software Survey: April 2018

<< Back to Overview

PC VIDEO CARD USAGE DETAILS — SORT BY: PERCENT SHARE — Sort

OVERALL DISTRIBUTION OF CARDS	DEC	JAN	FEB	MAR	APR	
DirectX 12 GPUs	92.06%	92.90%	91.36%	89.90%	**87.67%**	-2.23%
DirectX 11 GPUs	4.07%	3.53%	3.82%	4.44%	**6.37%**	+1.93%
DirectX 10 GPUs	2.82%	2.49%	2.70%	3.16%	**4.44%**	+1.28%
DirectX 9 Shader Model 2b and 3.0 GPUs	0.10%	0.09%	0.10%	0.11%	**0.14%**	+0.03%
DirectX 9 Shader Model 2.0 GPUs	0.01%	0.01%	0.01%	0.01%	**0.01%**	0.00%
DirectX 8 GPUs and below	0.94%	0.98%	2.01%	2.38%	**1.37%**	-1.01%

***pav. 1**: „Steam Hardware Survey“ 2018.*

Prieinama DirectX versija parodo vaizdo plokštės galimybes, todėl pagal tai galima pasirinkti ir OpenGL versiją. DirectX 11 apytiksliai atitinka OpenGL 4.2, o DirectX 12 tolimesnes versijas. OpenGL 4.3 sumažina reikiamą komunikaciją tarp programos ir vaizdo plokštės tvarkyklės praplečiant šeiderių programavimo kalbą. Pagal technologijos patogumą, prieinamumą ir kuriamos programos reikalavimus, tinkamiausia versija yra OpenGL 4.3. Su šia versija galima išnaudoti modernias vaizdo plokščių galimybes ir turėti suderinamumą tarp maždaug 90% naudojamų konfigūracijų.

2.2 **Programavimo kalbos**

Egzistuoja daugybė programavimo kalbų. Dažniausia jos skirstomos į interpretuojamas ir kompiliuojamas. Interpretuojamas kalbos dažnai būna gana aukšto lygio, jas interpretuoja interpretatorius, kuris skaito kodą kas eilutę ir vykdo atitinkamas komandas. Šiuo atveju programos klaidas galima sužinoti tik vykdymo metu. Kompiliuojamos kalbos iš pradžių yra paverčiamos į binarinę formą, kuri turi tikslias tai architektūrai reikalingas instrukcijas programai įvykdyti. Kompiliuojamos kalbos dažniausia turi stipresnę tipų sistemą, kuri leidžia atpažinti klaidas jau kompiliavimo metu. Plačiausiai naudojamos kalbos: C, C++, Java, Python, C#.

C – viena iš senesnių programavimo kalbų, sukurta 1972 metais, tačiau vis dar tobulinama ir naudojama. Kadangi kalba yra kompiliuojama, kiekvienai architektūrai reikia iš naujo sukompiliuoti programą. Leidžia efektyviai išnaudoti ir paskirstyti visus sistemos išteklius. Pati kalba neturi daug ypatybių, sintaksė yra gana paprasta, tačiau standartinė biblioteka turi labai mažai funkcionalumo.

C++ – turi C pagrindą, beveik visais atvejais korektiškas C kodas gali būti sukompiliuotas kaip C++ kodas. Tai viena iš šios kalbos stiprybių, nes labai lengvai galima kviesti C bibliotekas. Ši kalba prideda kelias naudingas ypatybes, tokias, kai klasės, funkcijų užklojimas, šablonai, bevardės funkcijos.

Java – populiari kalba, kompiliuojama į Java „Bytecode“. Tai binarinė programos reprezentacija kurią interpretuoja virtuali mašina ir paverčia „bytecode“ instrukcijas į esamos architektūros asemblerines instrukcijas. Ši kalba palaiko tik objektinio programavimo stilių ir turi stiprią tipų sistemą. Atmintis šioje kalboje yra valdoma automatiškai, todėl nereikia pačiam programuotojui jos valdyti. Tai gali būti naudinga arba žalinga priklausomai nuo programos reikalavimų.

Python – viena iš populiaresnių interpretuojamų programavimo kalbų. Pasižymi savo paprastumu ir dideliu lankstumu. Atminties valdymas taip pat yra automatiškas. Palaiko įvairius programavimo stilius.

C# - programavimo kalba dažniausiai naudojama „Microsoft Windows“ operacinės sistemos aplinkoje, tačiau galima ja naudoti ir kitur. Ši kalba panašiai kaip Java sukompiliuojama į „bytecode“ kuris programos veikimo metu paverčiamas į asemblerines esamo procesoriaus instrukcijas.

Visos išvardintos kalbos gali iškviesti C bibliotekas, todėl teoriškai įmanoma naudoti bet kokią iš jų norint panaudoti prieš tai minėtas vaizdo plokščių programavimo technologijas. Grafikos programavimas yra gana žemo lygio ir labiausiai tinkanti kalba yra C++, nes ji leidžia patogiai kviesti C bibliotekas, valdyti atmintį ir pasirašyti norimas abstrakcijas, kurios gali paprastinti programos rašymo procesą.

2.3 Programinė įranga

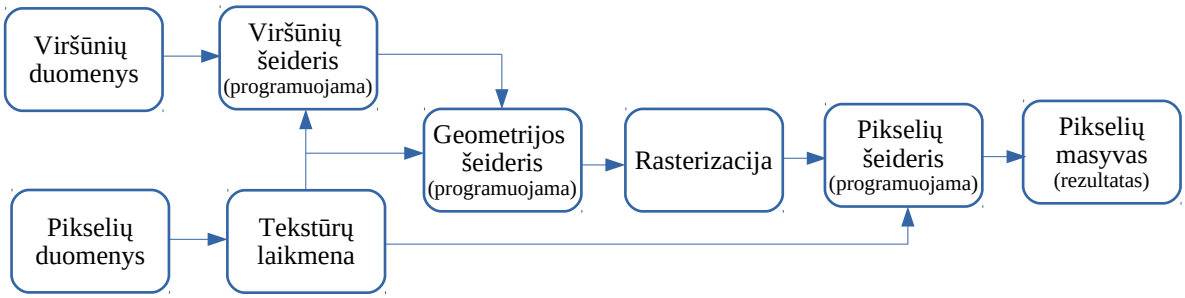
Programinei įrangai kurti yra naudojami įvairūs įrankiai: kompiliatoriai, teksto redaktoriai, programavimo aplinkos, statinės kodo analizės įrankiai arba kitos bibliotekos. Pasirinktai programavimo kalbai (C++) būtinas kompiliatorius. Labiausiai naudojami kompiliatoriai yra GCC, Clang ir Microsoft Visual C++. Visi jie yra aukštos kokybės, atitinka C++ ISO standartą ir gali sugeneruoti optimizuotą asemblerinį kodą. Linux aplinkoje dažniausiai naudojamas GCC kompiliatorius, o Windows aplinkoje Microsoft Visual C++. Pagal reikalavimus kuriama programa turi palaikyti ir Linux ir Windows operacines sistemas, todėl šiame darbe naudojami šie kompiliatoriai.

Kuriant programas, kartais tenka susidurti su problemomis, kurias išspręsti gali padėti kodo bibliotekos. Šiame darbe bus naudojamos trys bibliotekos:

1. SDL - „Simple DirectMedia Layer“ naudojama kaip lengva abstrakcija operacinės sistemos, leidžianti sukurti langą, OpenGL kontekstą ir paimti įvestis iš klaviatūros ar pelės.
2. GLAD – OpenGL implementacija yra suteikiama vaizdo plokščių tvarkyklių. Ši biblioteka yra automatiškai sugeneruojama. Joje egzistuoja visos OpenGL konstantos, funkcijų prototipai ir jų rodyklės.
3. stb_image – Vieno failo biblioteka suteikianti galimybę užkrauti paveikslėlius iš failų ar atminties.

2.4 Grafikos atvaizdavimo procesas

Visas realaus laiko kompiuterinės grafikos atvaizdavimo procesas yra sudarytas iš atskirų paprastų procesų atliekančių tam tikrą specifinę funkciją. Procesai kurie yra privalomi ir naudojami šiame darbe yra atvaizduoti pateiktame paveikslėlyje (žr. pav. 2).

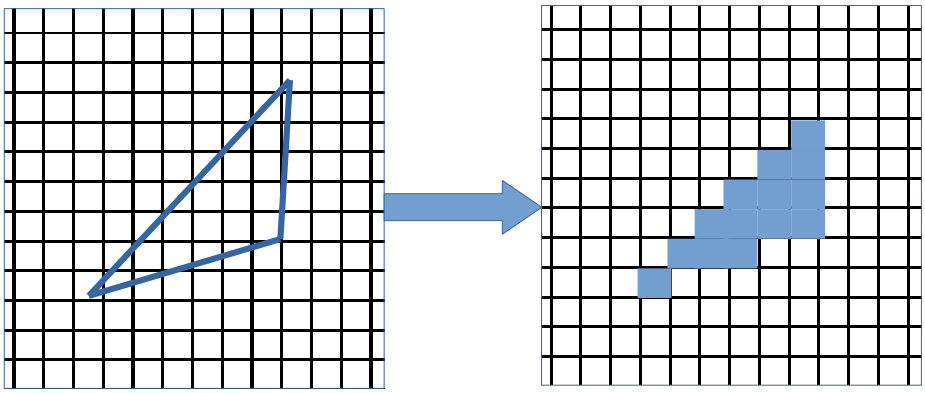


***pav. 2**: Grafikos atvaizdavimo procesas.*

Viršūnių ir pikselių duomenys yra perduodami iš pagrindinės programos į vaizdo plokštės atmintį. Viršūnių šeideris yra programa atsakinga transformuoti viršūnių duomenis (pozicijas, tekstūrų koordinates, spalvas, normalės vektorius ar kt.), tai reiškia, kad ši programa bus vykdoma tiek kartų kiek paduota viršūnių. Viršūnių duomenys dažniausiai transformuojami iš virtualaus pasaulio koordinačių sistemos į normalizuotas aparatinės įrangos koordinates, kurių aibė pagal OpenGL specifikaciją yra tokia: *x* (ilgio) ašis [-1;1], *y* (aukščio) ašis [-1;1], *z* (gylio) ašis [-1;1]. Ekranas yra dvimatis, tačiau naudojamos 3D koordinatės, nes gylio informacija yra labai naudinga norint turėti galimybę piešti objektus nepriklausoma eilės tvarka. Rodyklė iš tekstūrų laikmenos į viršūnių šeiderį pasako, kad šis šeideris gali skaityti tekstūras iš vaizdo plokštės atminties.

Rezultatai iš viršūnių šeiderio eina tiesiai į geometrijos šeiderį. Geometrijos šeideris taip pat atsakingas už viršūnių duomenų transformacijas, tačiau šiame etape galima sukurti naujas viršūnes arba atmesti esamas. Pavyzdžiui: iš atskiro taško erdvėje galima sukurti sudėtingesnę figūrą. Geometrijos šeideris nėra privalomas, tačiau piešimo metu būtina turėti viršūnių ir pikselių šeiderius.

Toliau eina rasterizacijos etapas. Šio proceso metu atliekamos operacijos kurios iš pozicijos taškų nustato dengiamus pikselius (žr. pav. 3), tada juos nuspalvina pikselių šeideris. Kadangi pikselių šeideris atsakingas už suteiktą spalvą, jame implementuojami algoritmai, kurie suteikia įvairius grafinius efektus: apšvietimas, tekstūravimas, gama-korekcija, ryškumo nustatymai, šviesių spalvų intensyvumas ir daugybė kitų. Šio etapo rezultatai įrašomi į pikselių masyvą, tačiau tai nebūtinai turi būti kompiuterio langas, tai gali būti kita tekstūra, kurią veliau gali naudoti, nuskaityti kiti šeideriai kito objekto ar efekto piešimo metu.



***pav. 3**: Rasterizacija*

Rasterizacija yra atliekama automatiškai vaizdo plokštės ir programuotojas prie šio proceso neturi prieigos ir jo keisti negali.

2.5 3D grafikos efektai

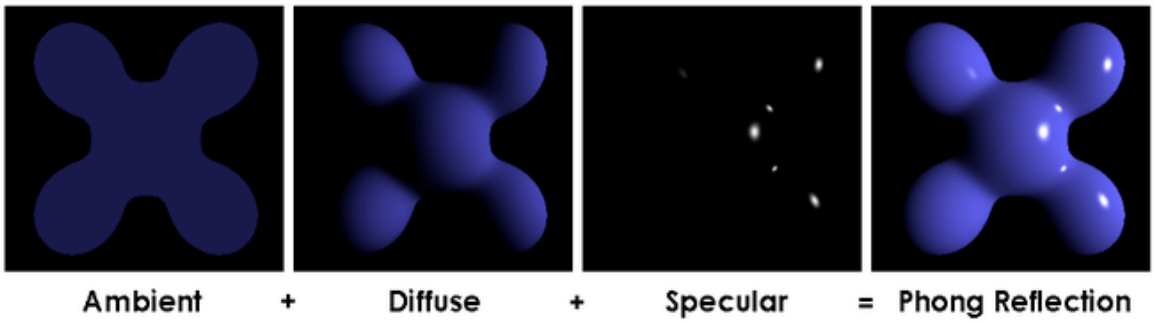
Visi grafikos specialieji efektai gali būti implementuojami transformuojant viršūnių ir pikselių parametrus. Efektyviausia tai atlikti sukuriant tinkamus šeiderius.

2.5.1 Apšvietimas

Apšvietimas yra vienas iš svarbiausių trimatės grafikos efektų. Jis suteikia objektams gylį, spalvą ir išryškina detales. Be apšvietimo būtų galima matyti tik objekto siluetą. Realiam pasaulyje šviesos šaltiniai skleidžia šviesą, dalį šios šviesos atspindi objektai, atsispindėjusi šviesa patenka į akis. Šį procesą reikia sumodeliuoti kompiuteryje. Galima sumodeliuoti šviesą, kaip fotonų srautą, paskaičiuoti fotonų trajektorijas, atsispindėjimą pagal objekto paviršiaus

savybes. Pasirenkant pakankamai didelį fotonų skaičių, būtų galima gauti fiziškai realistiškus rezultatus, tačiau tai reikalautų labai didelių skaičiavimų, kas yra visiškai nepraktiška, ypač realaus laiko grafinėse programose, kai vidutinė vieno kadro skaičiavimų trukmė yra 16 milisekundžių. Kad pasiekti tokią greitaveiką, reikia daryti tam tikras aproksimacijas.

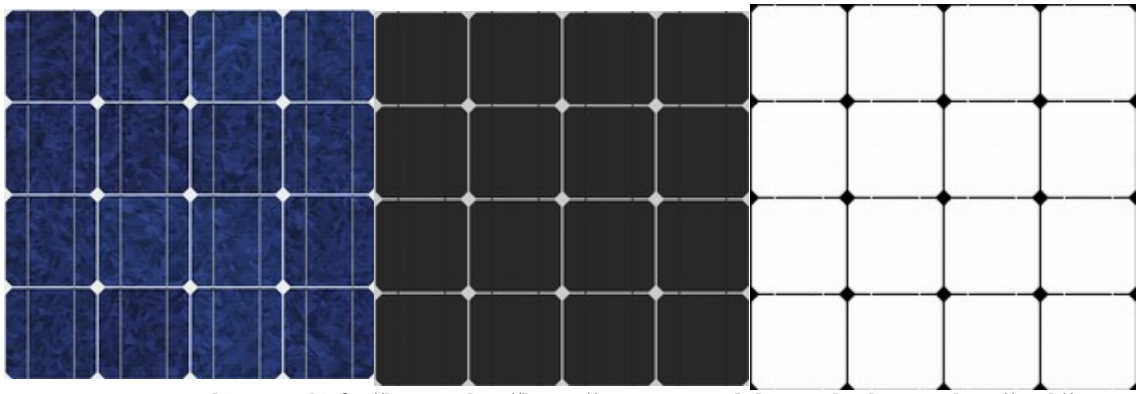
Vienas iš labiausiai paplitusių metodų yra „Phong“ apšvietimo modelis. Šis metodas išskaido šviesos skaičiavimus į tris pagrindines stadijas: aplinkos (angl. ambient), išskaidymo (angl. diffuse) ir spindesio (angl. specular). Aplinkos šviesa tai tiesiog aproksimuota spalvos konstanta kurią įgyja visi objektai scenoje. Išskaidymo šviesa suteikia pagrindinę spalvą objektui. Šios spalvos reikšmė taip pat priklauso nuo objekto paviršiaus šiurkštumo. Po šito skaičiavimo galima atskirti objekto formą, įdubimus ir šviesos šaltinio kryptį. Spindesio šviesa suteikia objektui spindėjimą priklausomai nuo paviršiaus normalės ir ateinančio šviesos spindulio krypties kampo. Šis efektas yra matomiausias kai šviesa nuo objekto paviršiaus atsispindi tiesiai į kamerą. Atlikus visus apšvietimo skaičiavimo etapus, galima rezultatus sudėti ir gauti apšviestą objektą (žr. pav. 4).



***pav. 4**: "Phong" apšvietimo modelis.*

Šie skaičiavimai gali būti atliekami viršūnių arba pikselių šeideriuose. Ankščiau, kai vaizdo plokštės, lyginant su šiuolaikinėmis, buvo ganėtinai silpnos, šie skaičiavimai buvo atliekami viršūnių šeideryje, nes objektas dažniausiai turi mažiau viršūnių, nei pikselių ekrane. Taigi apšvietimas atliekamas viršūnių šeideryje yra greitesnis, tačiau suteikia prastesnius rezultatus. Šiais laikais vaizdo plokštės yra gana galingos ir gali lengvai atlikti apšvietimo skaičiavimus pikselių šeideryje.

Kitas būdas, dažnai naudojamas aukštos kokybės kompiuterinėje grafikoje yra fiziškai realistiškas apšvietimo modelis (angl. physically based rendering). Šis būdas taip pat aproksimuoja realios šviesos fizišką veikimą, tačiau kitaip negu „Phong“ apšvietimas. Šiame apšvietimo modelyje objektų paviršius turi tris pagrindines savybes: spalva, šiurkštumą ir metališkumą. Šie parametrai dažniausiai yra paveikslėliai, kurie naudojami kaip tekstūros (žr. pav. 5). Kuo aukštesnės kokybės tekstūros tuo geresni rezultatai. Spalvos parametras suteikia objektui pradinę spalvą. Šiurkštumas nusako kaip šviesa atsispindėdama nuo objekto paviršiaus išsiskaido. Šiurkštesnis objektas turi platesnius ir labiau sulietus atspindžius nei lygus objekto paviršius. Metališkumas yra binarinis parametras, dažniausia tekstūrose laikoma kaip balta ar juoda spalva, tačiau kai kuriose situacijose naudojama pilka skalė. Šis apšvietimo būdas yra brangesnis nei ankščiau minėtas „Phong“ apšvietimo modelis, tačiau šiuolaikinė aparatinė įranga gali skaičiavimus atlikti pakankamai greitai net ir realaus laiko grafinėse programose.



pav. 5*: Saulės panelė fiziškai realistiško apšvietimo modelyje. Objekto spalva, šiurkštumas, metališkumas.*

Kartais vietoj šiurkštumo yra naudojama švelnumo tekstūra. Tada šiurkštumo parametras gaunamas iš vieneto atimant švelnumo reikšmę (šiurkštumas = 1.0 – švelnumas).

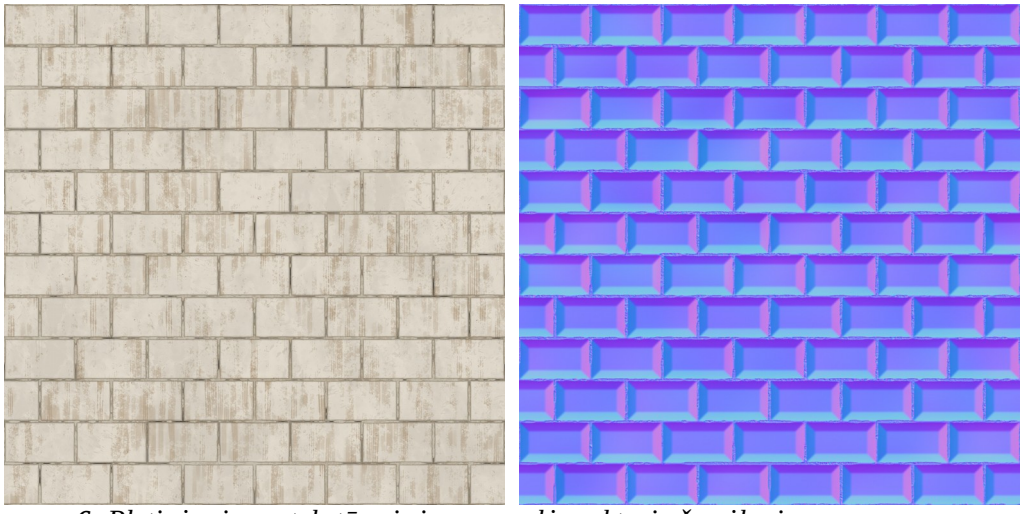
Lentelėje palyginami du prieš tai minėti apšvietimo modeliai.

„Phong“ apšvietimas		**Fiziškai tikslus apšvietimas**	
Privalumai	**Trūkumai**	**Privalumai**	**Trūkumai**
Paprasta implementacija			Sudėtingesnė implementacija
Greitas veikimas		Pakankamai greitas veikimas šiuolaikinėse vaizdo plokštėse	Lėtesnis veikimas
Paprasta aprašyti objekto medžiagos savybes			Medžiagai aprašyti naudojamos tekstūros
	Žemesnė atvaizdavimo kokybė	Atvaizduoti objektai atrodo įtikinamiau	

Įvertinant abiejų metodų privalumus ir trūkumus, pasirenkamas fiziškai tikslus apšvietimo modelis.

2.5.2 Normalės vektorių žemėlapiai

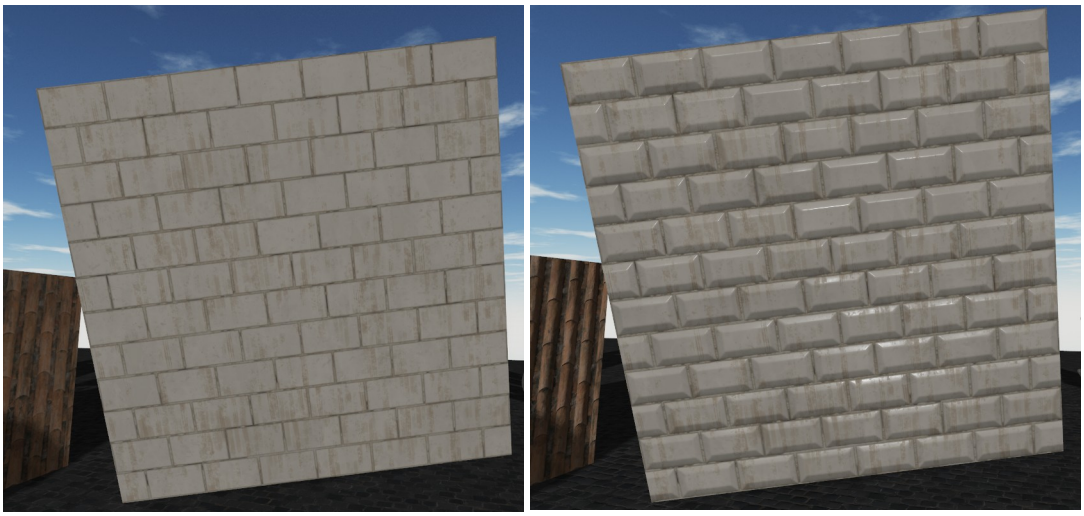
Normalės vektoriai yra labai svarbūs apšvietimo skaičiavimuose, nes pagal juos paskaičiuojamas šviesos atsispindėjimas. Objektai su klaidingais normalės vektoriais gali atrodyti netinkamai, vietos kurios turėtų atrodyti šviesesnės gali būti tamsios ir atvirkščiai. Paprastai normalės vektoriai yra sugeneruojami modeliavimo programose arba paskaičiuojami pačios grafinės programos pagal viršūnių pozicijas. Šis sprendimas nėra labai efektyvus, nes normalės vektoriai iš pradžių patenka į viršūnių šeiderį ir tik tada perduodami į pikselių šeiderį norimam apšvietimui apskaičiuoti. Kadangi viršūnių paprastai būna žymiai mažiau nei pikselių, kuriuos objektas užima ekrane, o apšvietimo skaičiavimai vyksta su kiekvienu pikseliu, tenka dalį duomenų sugeneruoti. Vektoriai tarp dviejų viršūnių yra sugeneruojami taip, kad pirmos viršūnės vektorius tolygiai keičiasi į antros viršūnės vektorių. Šis procesas vadinamas interpoliacija ir yra atliekamas automatiškai vaizdo plokštėse.



***pav. 6**: Plytinės sienos tekstūra ir jos normalės vektorių žemėlapis.*

Norint gauti tikslesnius rezultatus, reikia turėti daugiau viršūnių. Tai padidina atminties reikalavimus ir vaizdo plokštės apkrovą, nes kuo daugiau viršūnių tuo daugiau kartų viršūnių šeideris turi atlikti skaičiavimus. Ši problema išsprendžiama normalės vektorių žemėlapiais. Normalės vektorių žemėlapis yra paprasta tekstūra, kurios raudonos, žalios ir mėlynos komponenčių reikšmės atitinka normalės vektoriaus x, y ir z reikšmes (žr. pav. 6). Šitokiu būdu, kiekvienas objekto pikselis turi tikslią normalės vektoriaus reikšmę, o panaudojimo kaina yra labai maža – tekstūros nuskaitymas pikselių šeideryje.

Normalės vektorių žemėlapiai atrodo melsvai, nes z koordinatę eina tiesiai „iš ekrano“, o šią koordinatę atitinka mėlyna spalva tekstūroje. Scenos koordinačių sistema, kurioje atliekami apšvietimo skaičiavimai, yra kitokia negu normalės vektorių koordinačių sistema, todėl negalima tiesiog nuskaityti normalės žemėlapio reikšmės ir ją iškarto naudoti apšvietimo skaičiavimuose. Prieš tai reikia šį vektorių perkelti į tinkamą koordinačių sistemą. Tai atliekama naudojant matematines matricas. Ši matrica sukuriama pagal viršūnės normalės reikšmę ir tekstūros koordinates. Tokia matrica žinoma kaip tangentinės erdvės matrica (angl. Tangent Space Matrix). Padauginus nuskaitytą normalės vektorių iš tangentinės matricos gaunamas normalės vektorius teisingoje koordinačių sistemoje. Objektas atvaizduotas naudojant normalės žemėlapį (žr. pav. 7) turi žymiai daugiau detalių, įdubimų, nors pačios paviršiaus viršūnės yra vienoje plokštumoje.

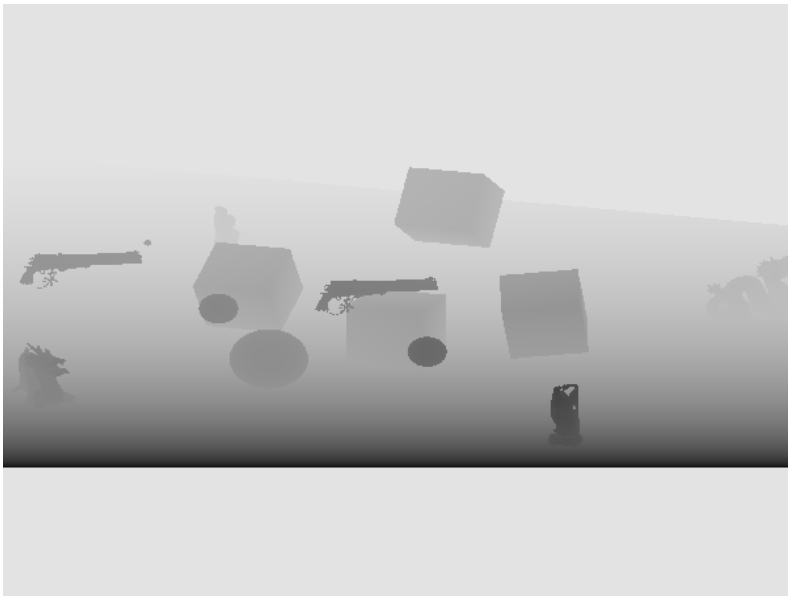


***pav. 7**: Objektas be ir su normalės vektorių žemėlapiu.*

Normalės vektorių žemėlapiai yra puikus būdas pagerinti trimatės scenos atvaizdavimo kokybę nepadidinant viršūnių skaičiaus. Reikalingas tik papildomas tekstūros nuskaitymas ir viena matricos ir vektoriaus daugyba.

2.5.3 Šešėlių žemėlapiai

Šešėliai kompiuterinėje grafikoje yra viena iš brangiausių operacijų, tačiau jie scenai suteikia nemažai realistiškumo. Yra labai daug šešėlių atvaizdavimo metodų, tačiau jie visi turi bendrą pagrindinį veikimą. Beveik visi šešėlių atvaizdavimo algoritmai nupiešia visa sceną iš šviesos šaltinio perspektyvos. Nupiešta scena yra laikoma gylio kadro buferyje. Tai pilkos skalės tekstūra laikanti atstumą (žr. pav. 8), nuo ekrano iki objekto. Gylio buferis pagrinde naudojimas patikrinti ar dabar piešiamas objektas yra užstojamas kito objekto. Tai leidžia piešti objektus bet kokia eilės tvarka. Gylio buferis šešėlių generavimo kontekste dar kartais vadinamas šešėlių žemėlapiu (angl. shadow map). Turint šešėlių žemėlapį piešiama pagrindinė scena, tačiau pikselių šeideryje patikrinama ar dabar spalvinamas pikselis yra uždengtas kito objekto (ar dabartinis pikselio gylis yra didesnis už šešėlių žemėlapyje esančią gylio reikšmę). Čia susiduriama su labai panašia problema, kaip ir normalės vektorių žemėlapio panaudojime. Dabar spalvinamas pikselis turi visai kitas koordinates, negu pikselis šešėlių žemėlapyje, nes scena piešiama iš visai kitos perspektyvos, todėl atskaitos taškai vėl nesutampa. Tai galima išspręsti apsirašant šviesos šaltinio matricą, kuri transformuotų esamą pikselį į šviesos šaltinio atskaitos tašką. Tik tada galima nuskaityti šešėlių žemėlapį ir gauti tinkamus rezultatus.



pav. 8*: Gylio buferis sukurtas piešiant sceną iš šviesos perspektyvos.*

Šešėlių kokybė priklauso nuo šešėlių žemėlapio raiškos ir reikšmių tikslumo. Šis efektas yra labai brangus, nes visą trimatę sceną reikia nupiešti bent du kartus. Pirmą kartą kai generuojamas šešėlių žemėlapis, o antrą kartą kai piešiama scena į pagrindinį kadro buferį (kompiuterio langą ar kitą tekstūrą). Kad šis procesas vyktų greičiau naudojamos įvairios optimizacijos. Vienas iš optimizacijos būdų yra turėti paprastesnes modelių versijas. Kadangi gylio buferyje matosi tik objektų siluetai, mažesnis viršūnių kiekis nesumažina kokybės, tačiau sumažina vaizdo plokštės apkrovą, nes viršūnių šeideris bus kviečiamas mažiau kartų. Jeigu scena didelė taip pat reikia tinkamai parinkti projekcijos matricą, naudojamą piešti objektus iš šviesos perspektyvos. Kadangi kompiuterinėje grafikoje dažniausia naudojamos 16 ar 32 bitų tikslumo slankiojančio kablelio reikšmės, svarbu, kad šešėlių žemėlapio reikšmės būtų mažame

intervale. Kitu atveju palyginimas ar pikselis yra šešėlyje gali tapti labai netikslus ir sukurti šešėliai gali tik pabloginti visos trimatės scenos vaizdą.

2.5.4 HDR kadro buferis

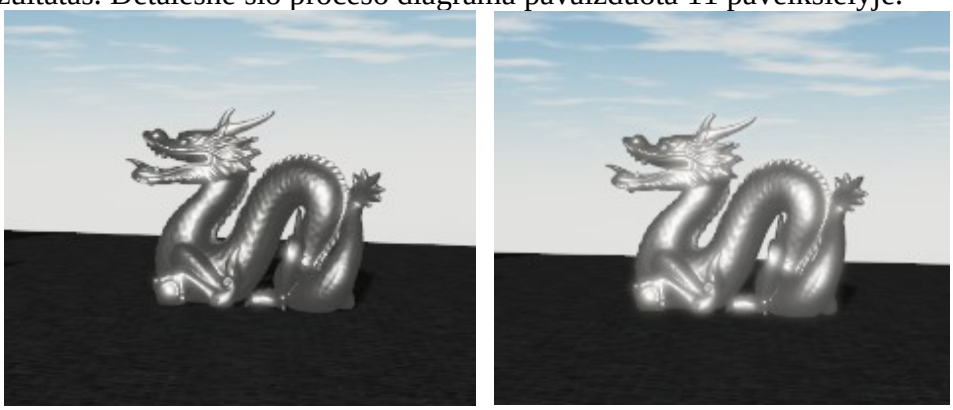
Paprastai ekrane ar tekstūroje kiekvienam pikseliui priskiriamos trys spalvos (raudona, žalia ir mėlyna) su 8 bitų natūralių skaičių tikslumu [0; 255]. HDR kadro buferis praplečia šį intervalą naudojant slankiojančio kablelio reikšmes. Šitokiu būdu galima išlaikyti daugiau detalių piešimo metu, tačiau galutinis rezultatas vis tiek turi būti konvertuojamas atgal. Procesas atliekantis šį veiksmą vadinamas „tone mapping“. Beveik visi šio proceso algoritmai turi išryškinimo (angl. exposure) parametrą. Šis parametras reguliuoja scenos ryškumą (žr. pav. 9). Egzistuoja algoritmų kurie dinamiškai pagal scenos ryškumą keičia išryškinimo parametrą. Tamsioje scenoje jis tampa didesnis, labai šviesioje – mažesnis. Šis veikimas labai panašus į žmogaus akį.

***pav. 9**: HDR „tone mapping“ su skirtingais išryškinimo parametrais.*

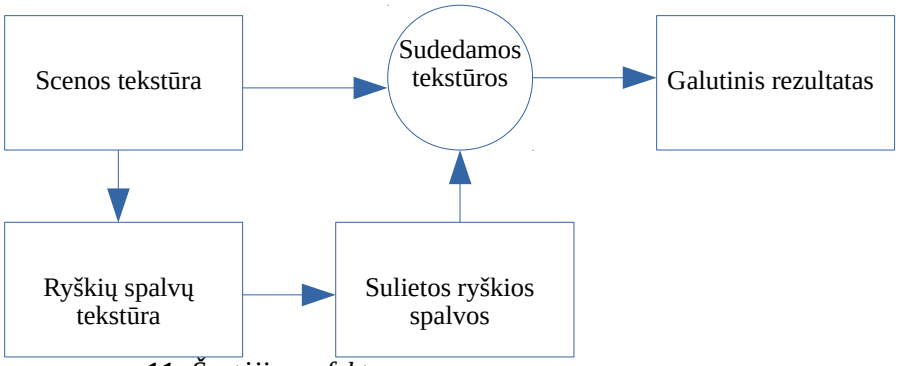
Šiuolaikinės vaizdo plokštės, net ir mobiliuose įrenginiuose palaiko slankiojančio kablelio reikšmių kadro buferį, o „tone mapping“ algoritmai nėra labai brangūs. Todėl beveik visos trimatės grafinės programos naudoja šį metodą, kad pagerinti scenos kokybę. Šis metodas taip pat suteikia galimybę kitiems efektams.

2.5.5 Švytėjimo efektas

Švytėjimo efektas (angl. bloom) yra vienas iš tų kuris puikiai gali pasinaudoti HDR suteiktomis galimybėmis. Šis efektas sušvelnina labai ryškius objektus arba jų dalis ir suteikia jiems švytėjimo (žr. pav. 10). Efekto algoritmas susideda iš dviejų pagrindinių etapų: nupiešiamas į atskirą tekstūrą, tada išrenkamos šviesios spalvos ir įrašomos į kitą tekstūrą, tekstūra turinti ryškias spalvas yra suliejama (dažniausiai naudojamas „Gaussian blur“ algoritmas), tada sulieta ryškių spalvų tekstūra yra pridedama prie originalios ir gaunamas galutinis rezultatas. Detalesnė šio proceso diagrama pavaizduota 11 paveikslėlyje.



***pav. 10**: Objektas atvaizduotas be ir su švytėjimo efektu.*

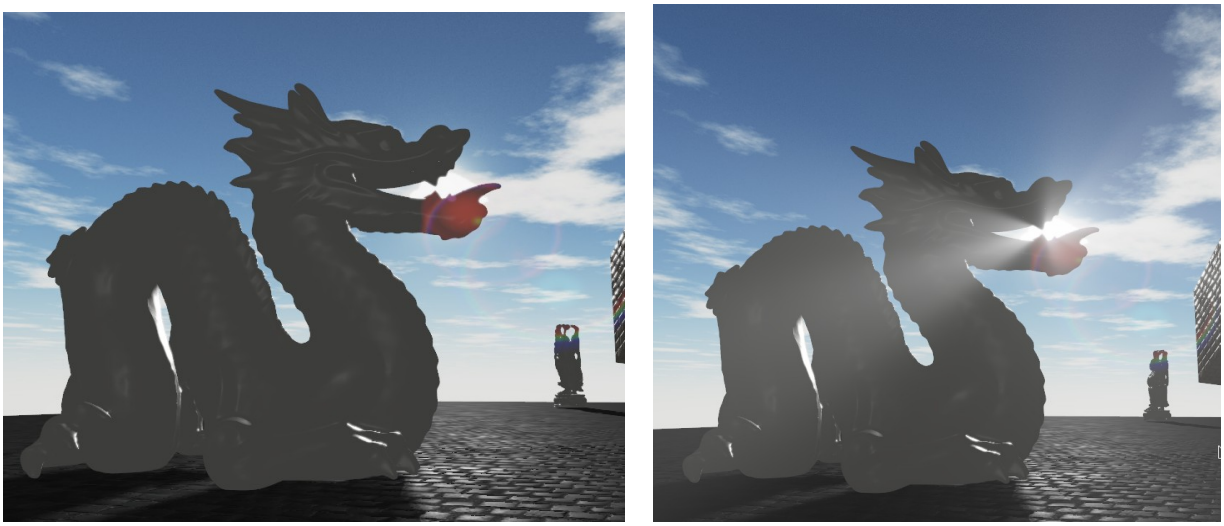


***pav. 11**: Švytėjimo efekto procesas.*

Švytėjimo efektas paprastai turi du pagrindinius parametrus: suliejimo koeficientą (kiek pikselių suliejama) ir sudėties koeficientą (kokia dalis sulietos tekstūros yra pridedama prie scenos). Aukštos kokybės švytėjimo efektai dažnai sulieja ryškių spalvų tekstūrą kelis kartus su skirtingais parametrais. Taip galima gauti dar švelnesnes spalvas arba stipresnį efektą atvaizduojant šviesos šaltinius.

2.5.6 Šviesos išsiskaidymas

Šviesos išskaidymo efektas atvaizduoja šviesos spindulius kai virtuali kamera žiūri į šviesos šaltinį (žr. pav. 12). Šio efekto veikimas labai panašus į prieš tai minėtą švytėjimo efektą. Scenoje esantys šviesos šaltiniai yra piešiami į atskirą tekstūrą, tada jie yra suliejami, kaip ir spindesio efekto metu. Sulieta tekstūra yra panaudojama pikselių šeideryje ir atliekamas šio efekto algoritmas.



***pav. 12**: Scena be ir su šviesos išskaidymo efektu.*

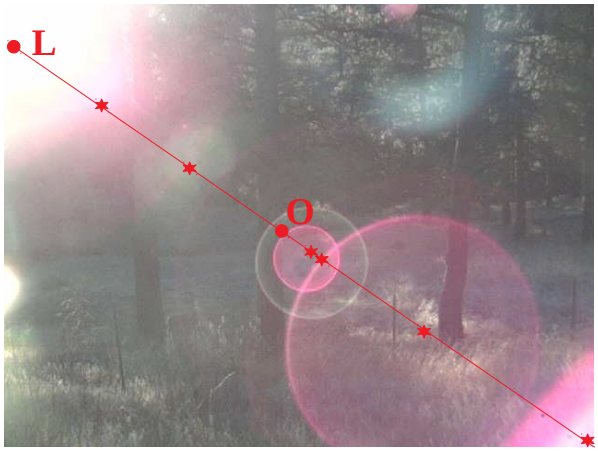
Šis efektas nėra labai brangus skaičiavimų atžvilgiu. Vaizdo plokštės apkrova priklauso nuo kompiuterio lango ar kadro buferio dydžio ir nuo algoritmo parametrų.

2.5.7 Lęšio žibėjimas

Stiprus šviesos šaltinis realioje kameroje sukelia lęšio žibėjimo efektą (angl. lens flare). Šis efektas atsiranda kai ryški šviesa kameros lęšyje išsiskaido dėl pačio lęšio medžiagos netobulumų. Lęšio žibėjimas virtualiai trimatei scenai gali suteikti daugiau realizmo ir dramatiškumo. Efekto pavyzdys realios kameros pavaizduotas 13 paveikslėlyje. Kad sukurti šį

efektą virtualioje trimatėje scenoje, iš pradžių reikia nupiešti sceną į atskirą tekstūrą ir išsisaugoti šviesos šaltinio poziciją ekrano koordinačių sistemoje. Toliau iš šviesos šaltinio pozicijos L (žr. pav. 14.) ir vidurio O sukuriama linija pagal kurią bus išdėstomos lęšio žibėjimo tekstūros. Atstumai tarp efekto tekstūrų gali būti dinamiškai koreguojami pagal atstumą nuo šviesos šaltinio iki ekrano centro.

***pav. 13**: Lęšio žibėjimo efektas realioje kameroje.*



***pav. 14**: Lęšio žibėjimo efekto kūrimas kompiuterinėje grafikoje.*

Kadangi šis efektas yra matomas tik tada, kai ryškus šviesos šaltinis yra ekrane ir jo pilnai neužstoja kiti objektai, reikia surasti būdą, kuris leistų nustatyti ar šviesos šaltinis yra matomas šiuo momentu. Tai padaryti galima pasinaudojant užklausas vaizdo plokštei. Šviesos šaltinio pozicijoje L piešiamas permatomas kvadratas šviesos šaltinio dydžio. Prieš pradedant piešimą pradedama gylio užklausa. Šios užklausos rezultatas gražina pikselių, kurie praėjo gylio patikrą (pikselių kurie yra ekrane), skaičių. Pagal šį rezultatą galima keisti efekto tekstūrų permatomumą ir gauti tinkamą veikimą. Jei šviesos šaltinis yra visiškai užstojamas, efekto tekstūros yra visiškai permatomos ir efekto nesimato.

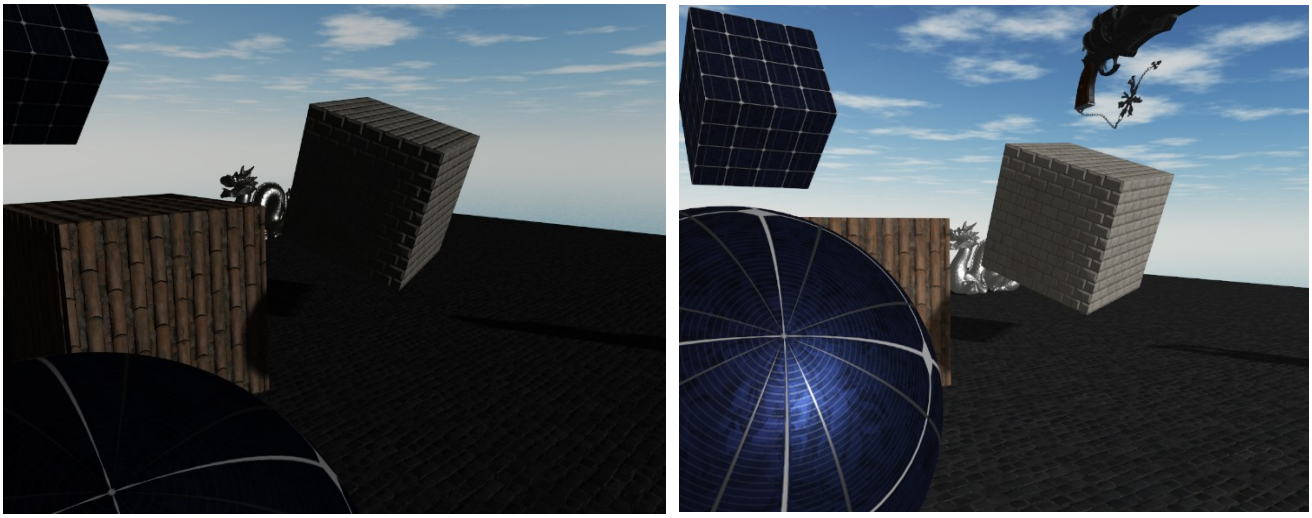
Šio efekto greitaveika priklauso nuo lęšio žibėjimo tekstūrų dydžių ir skaičiaus. Kadangi tekstūrų atvaizdavimas naudojant šiuolaikines vaizdo plokštes yra gana pigus, galima teigti, kad šis efektas didelės įtakos visos programos greitaveikai nedaro.

2.5.8 Gama korekcija

Žmogus šviesos intensyvumą mato netolygiai. Tai reiškia, kad dvigubai stipresnis šviesos srautas žmogui neatrodo dvigubai ryškesniu. Monitoriai naudoja sRGB (angl. standard Rend Green Blue) spalvų erdvę kuri atsižvelgia į šį faktą. Transformuojant spalvų reikšmes apšvietimo ar kitų efektų skaičiavimuose svarbu, kad spalvos butų tolygioje erdvėje. Tai reiškia, kad kiekvieną nuskaitytą tekstūros pikselį reikia konvertuoti į šią erdve ir tik tada atlikti skaičiavimus. Visos vaizdo plokštės nuo 2005 metų palaiko funkcionalumą, kuris automatiškai atlieka šią transformaciją. Keliant tekstūrą į vaizdo plokštės atmintį, tereikia paduoti papildomą argumentą, pasakantį kad ši tekstūra yra sRGB erdvėje. Ši transformacija vyksta vaizdo plokštėje ir praktiškai yra visiškai nemokama skaičiavimų atžvilgiu. Svarbu paminėti, kad ne visos tekstūros laiko spalvos informaciją. Normalės vektorių žemėlapiai, metališkumo, šiurkštumo, aukščio ar kitokios tekstūros jau yra tolygioje erdvėje ir jų reikšmių papildomai transformuoti nereikia.

Kad gauti gama korektišką rezultatą, reikia atlikti paskutinį veiksmą kuris gražintų rezultatą atgal į sRGB erdvę. Tai atliekama po visų kitų, objektų ir efektų piešimo. Kai visas kadras jau yra paruoštas atvaizdavimui ekrane, paleidžiamas pikselių šeideris kuris

transformuoja spalvas į teisingą formatą. Paveikslėlyje 15 palyginama scena, kuri atvaizduota su gama korekcija ir be jos.



***pav. 15**: Scena be gama korekcijos ir su ja.*

Iš paveikslėlio matosi, kaip gama korekcija pagerina visą vaizdą. Be šio efekto tamsesnės vietos atrodo per tamsios, šešėliai sunkiai matosi ir visa scena atrodo klaidingai.

2.6 **Egzistuojantys sprendimai**

Egzistuoja daugybė įvairių grafikos efektų implementacijų įvairiuose kompiuteriniuose žaidimuose, kitose realaus lauko grafinėse aplikacijose ar kompiuterių sugeneruotuose filmuose. Efektų implementacijos yra labai įvairios, jos gali būti sukonfigūruotos būtent specialiuose kontekstuose ir palyginimai tarp jų nėra labai prasmingi.

Grafikos efektus galima kurti naudojant jau sukurtus žaidimų variklius tokius kaip „Unity“ arba „Unreal“. Šie varikliai naudotojams suteikia galimybę rašyti šeiderius ir naudoti kitų žmonių sukurtus efektus. Šios programos dažniausiai naudoja šiek tiek aukštesnio lygio šeiderių programavimo kalbas, kurios vėliau būna paverčiamos į reikiamą GLSL ar HLSL formatą, priklausomai nuo operacinės sistemos. Šių programų naudojimas gali šiek tiek paprastinti šeiderių rašymą ir padaryti jį patogesnį programuotojui. Kai kurie kintamieji tokie kaip laikas, tekstūrų koordinatės ar kita naudinga informacija apie atvaizduojamą objektą yra automatiškai perduodama į šeiderius ir jie yra labai lengvai prieinami. Kuriant atvaizdavimo sistemą naudojant tik vaizdo plokščių programavimo technologijas išvardintas pirmame skyriuje, tokius duomenis į šeiderius reikia perduoti pačiam atskirai.

Nors šios programos suteikia galimybę implementuoti beveik visus grafinius efektus, jos paslepia daug detalių ir jų naudojimas atimtų didelę dalį šio darbo tikslo.

3 Projektas

Projektuojamos reikalingos sistemos analizėje minėtiems grafiniams efektams sukurti.

3.1 **Reikalavimai**

Funkciniai reikalavimai:

- Galimybė atvaizduoti trimatę sceną;
- Virtualios kameros valdymas;
- Fiziškai paremto apšvietimo modelis;
- Normalės vektorių žemėlapiai;
- Šešėlių žemėlapiai;
- HDR ir „tone mapping“ efektas;
- Švytėjimo efektas;
- Šviesos išskaidymo efektas;
- Lęšio žibėjimo efektas;

Nefunkciniai reikalavimai:

- Linux ir Windows operacinių sistemų palaikymas;
- Suderinamumas su vaizdo plokštėmis kurios palaiko OpenGL 4.3 versiją;
- 60-120 kadrų per sekundę;

3.2 **Naudojamos technologijos**

Pasirinkta vaizdo plokščių technologija yra OpenGL 4.3, nes ši sąsaja yra palaikoma ir Windows ir Linux operacinėse sistemose. Iš pirmos analizės dalies buvo nustatyta, kad OpenGL 4.3 versija turėtų palaikyti apytiksliai 90% visų kompiuterių konfigūracijų, turinčių vaizdo plokštę. Taip pat ši versija sumažina komunikaciją tarp šeiderio ir centrinio procesoriaus.

Pasirinkta programavimo kalba yra C++, nes ši kalbai yra visiškai suderinama su C bibliotekomis, o OpenGL specifikacija naudoja C kalbą. Grafikos programavimas reikalauja darbo su atmintimi, todėl kalba, kuri palaiko rodykles (angl. PowerPoint) yra labai priimtina. C++ taip pat leidžia kurti klases, užkloti operatorius, turi stipresnę tipų sistemą. Darbe kompiliatoriai skiriasi pagal platformą. Windows operacinėje sistemoje naudojamas Visual C++, o Linux aplinkoje – GCC.

Labai specifinėms problemoms spręsti naudojamos trys bibliotekos: SDL, GLAD, stb_image. SDL biblioteka suteikia vienodą sąsają komunikuoti su operacine sistema. Tai reiškia, kad suteikiamos vienodos funkcijos kompiuterinio lango sukūrimui, klaviatūros ar pelės nuskaitymui Linux ir Windows operacinėse sistemose. GLAD biblioteka suteikia reikiamas OpenGL konstantas ir funkcijų rodykles, o stb_image leidžia patogiai nuskaityti paveikslėlį iš failo.

3.3 **Vartotojo sąsaja**

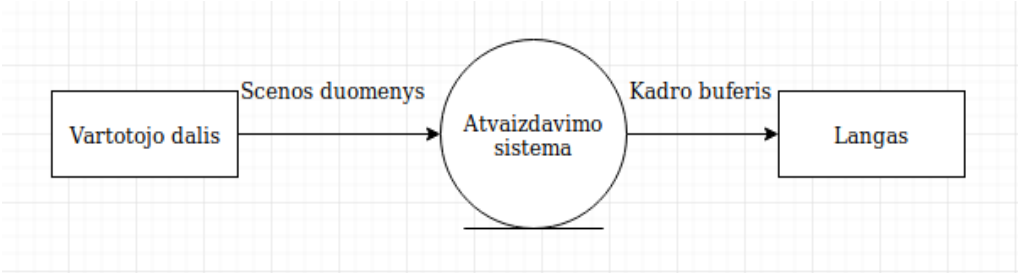
Kuriamas projektas yra realaus laiko trimatės erdvės ir grafinių efektų atvaizdavimo programa. Vartotojo sąsaja yra labai paprasta. Kompiuteryje sukuriamas langas kuriame atvaizduojama sukurta scena. Vartotojas virtualią kamerą valdo klaviatūra ir pele. Galima keisti virtualios kameros orientaciją, poziciją ir judėjimo greitį. Lango pavadinime galima matyti scenos atnaujinimo dažnį kadrais per sekundę. Taip pat yra galimybė matyti kiekvieno kadro skaičiavimo laiką centrinio ir vaizdo procesorių mikrosekundėmis.

3.4 **Atvaizdavimo sistema**

Visą programą galima suskaidyti į dvi dalis:

- Vartotojo dalis – užpildo trimatės scenos reprezentacijos duomenų struktūras, paima įvestis iš klaviatūros ir pelės, valdo virtualią kamerą, iškviečia atvaizdavimo sistemą;
- Atvaizdavimo sistema – priima trimatės scenos reprezentaciją ir atvaizduoja ją ekrane;

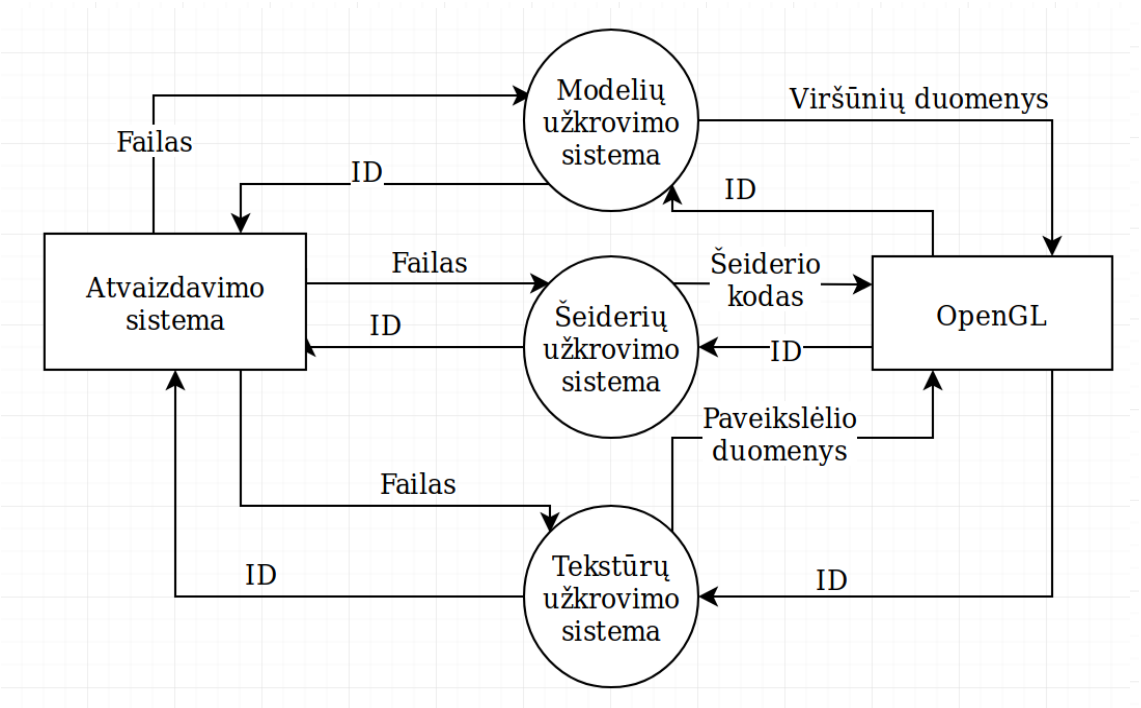
Labai abstrakti programos konteksto diagrama pavaizduota 16 paveikslėlyje.



***pav. 16**: Visos programos konteksto diagrama.*

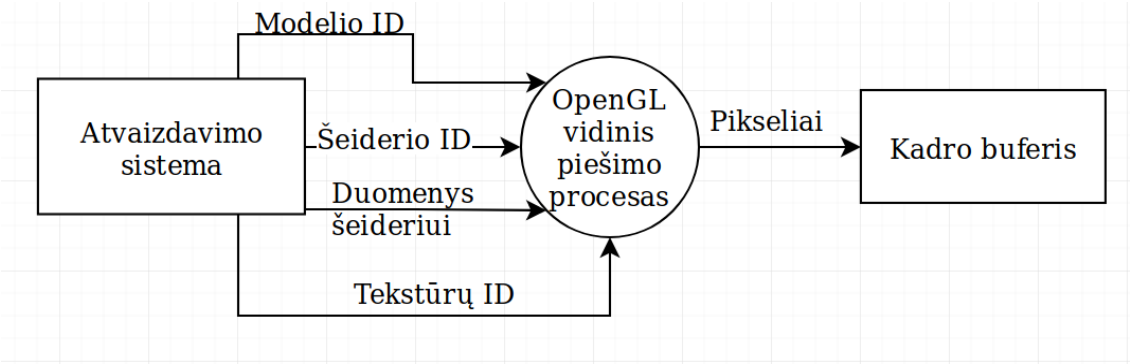
Pavaizduotoje diagramoje „vartotojo dalis“ yra iš programuotojo perspektyvos. Programuotojas naudoja atvaizdavimo sistemą, kaip biblioteką. Vartotojo dalyje tereikia užpildyti sceną reprezentuojančią duomenų struktūrą norimais duomenimis ir ją paduoti atvaizdavimo sistemai, kuri praeina pro visus piešiamus objektus ir juos nupiešia į kadro buferį su visais reikiamais grafiniais efektais.

Sudėtingiausia visos programos dalis yra atvaizdavimo sistema. Ji atsakinga už trimačių modelių, tekstūrų, šeiderių užkrovimą, duomenų į vaizdo plokštę nusiuntimą, šeiderių parametrų nustatymą ir visų reikalaujamų efektų atlikimą. Ši sistema yra pakankamai plati, todėl jos konteksto diagramos išskaidomos į inicializacijos (žr. pav. 17) ir veikimo būsenos (žr. pav.18).



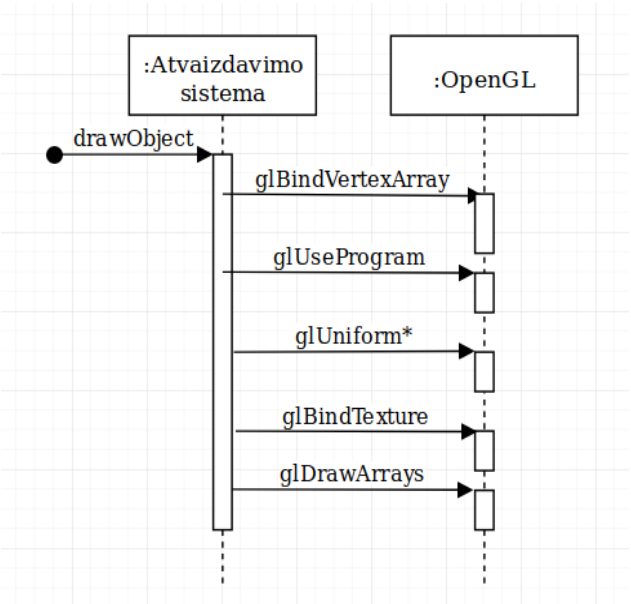
***pav. 17**: Konteksto diagrama atvaizdavimo sistemos inicializacijos metu.*

OpenGL pav. 17 atitinka vaizdo plokštę, nes tai yra sąsaja su ja per įrangos tvarkyklę. Atvaizdavimo sistema paduoda failus modelių, tekstūrų ir šeiderių užkrovimo sistemoms. Jos nuskaito šiuos failus ir reikiamus duomenis per OpenGL sąsają nusiunčia į vaizdo plokštę. Modelių atveju siunčiami viršūnių duomenys: pozicijos, normalės vektoriai, tekstūrų koordinatės. Vaizdo plokštės sąsaja gražina identifikatorių „ID“ su kuriuo bus galima naudoti įkeltus duomenis. Šeiderių atveju nusiunčiamas nuskaitytas GLSL kodas, šis kodas yra sukompiliuojamas pačios aparatinės įrangos tvarkyklės ir gaunamas „ID“ kuris bus naudojamas reikiamam šeideriui aktyvuoti piešimo metu. Tekstūrų atveju nusiunčiami paveikslėlių duomenys RGBA (angl. red, green, blue, alpha) formatu ir gaunamas identifikatorius.



pav. 18: Konteksto diagrama atvaizdavimo sistemos piešimo metu.

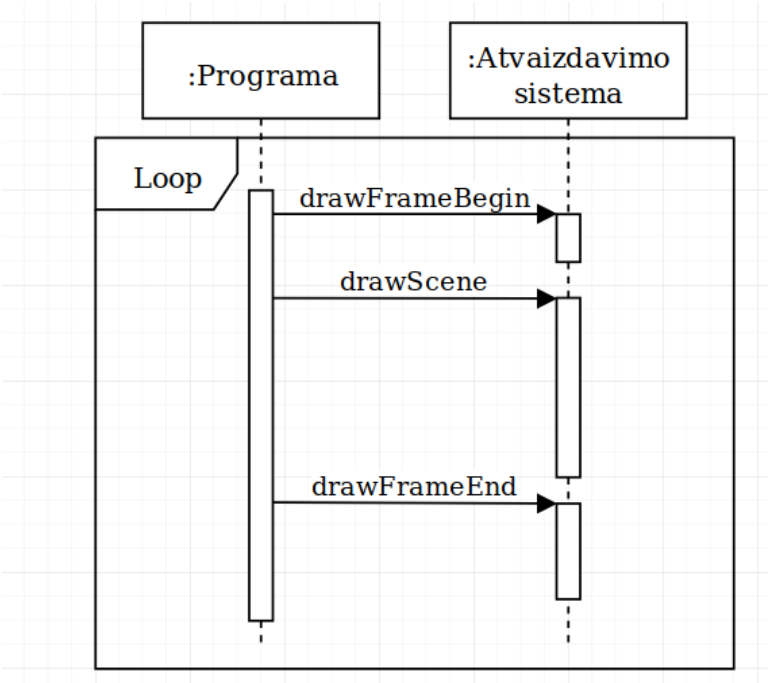
Kai reikiami duomenys yra vaizdo plokštės atmintyje, galima juos panaudoti ir atvaizduoti figūrą. Kad tai padaryti, reikia nustatyti kokius duomenis turės naudoti vaizdo plokštė. Tai atliekama paduodant gautus identifikatorius „ID“. Šeideriai viduje gali turėti kintamuosius kurie yra paduodami iš centrinio procesoriaus prieš kiekvieną piešimą. Šį funkcionalumą parodo „duomenys šeideriui“. Šie duomenys dažniausiai susideda iš modelio transformacijos matricos, šviesos informacijos bei kitų, specialiems efektams būdingų parametrų. Kadro buferyje gaunami rezultatai. Šis buferis gali būti kompiuterio langas arba atskira tekstūra. Pikseliai kuriais užpildytas kadro buferis yra nuspalvinti pikselių šeideriu. OpenGL vidinis piešimo procesas atitinka analizėje minėtą atvaizdavimo procesą (žr. pav. 2). Atvaizdavimo sistemos komunikacija su OpenGL, objekto piešimo metu, pavaizduota sekų diagramoje (žr. pav. 19).



***pav. 19**: Atvaizdavimo sistemos piešiant objektą sekų diagrama.*

Sekų diagramoje parodytos reikiamos OpenGL funkcijos ir jų iškvietimo tvarka norint nupiešti objektą naudojant vaizdo plokštę. Funkcijos *glBindVertexArray*, *glUseProgram*, *glBindTexture*, parametruose reikalauja modelio, šeiderio ir tekstūros ID atitinkamai. Duomenis iš operatyviosios atminties (RAM) į šeiderio atmintį galima perduoti naudojant funkcijas *glUniform**. Kadangi OpenGL turi C sąsają, nėra funkcijų užklojimo, todėl kiekvienam tipui perduoti egzistuoja atskira funkcija. Pasirinkus visus norimus resursus galima iškviesti paskutinę funkciją *glDrawArrays* kuri paleis visą vidinį atvaizdavimo procesą.

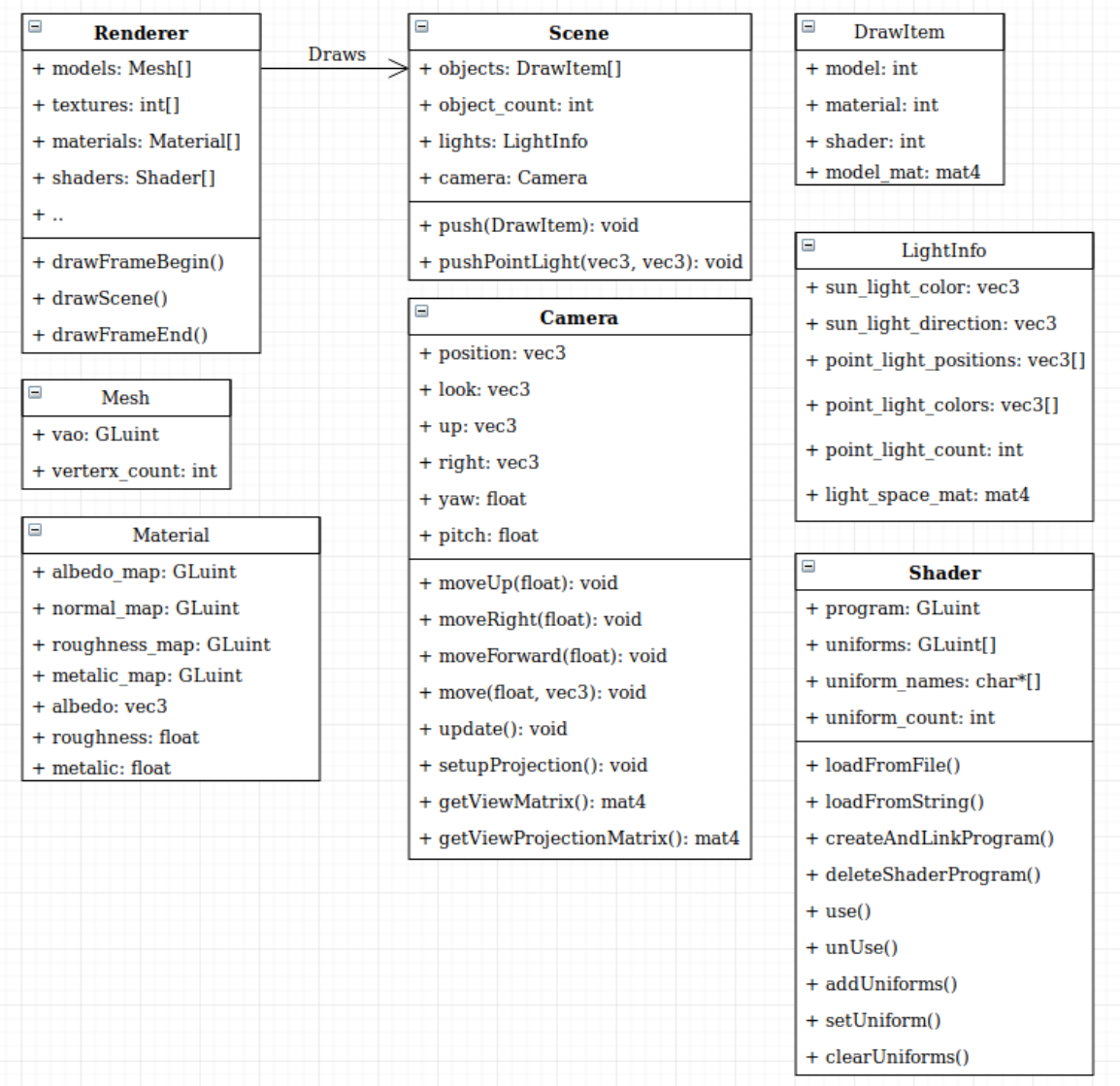
Grafiniai efektai yra suskirstomi į dvi dalis: efektai atliekami objekto piešimo metu ir efektai atliekami kadro pabaigoje. Pastarieji efektai žinomi kaip „post-process“ efektai. Atvaizdavimo sistemos ir jos naudojimas išskaidomas į tris pagrindinius žingsnius (žr. pav. 20).



***pav. 20**: Atvaizdavimo sistemos veiksmų ciklo sekų diagrama.*

Pavaizduotoje sekų diagramoje parodytas atvaizdavimo sistemos naudojimas. Pirmame veiksme (*drawFrameBegin*) sistema nustato piešimo kadro buferį, ištrina praėjusio kadro gylio ir pikselių duomenis, iš naujo užkrauna šeiderius iš failo, jeigu jie pasikeitė ir pasiruošia priimti sceną reprezentuojantį objektą. Šis etapas yra trumpiausias iš visų. Antroje dalyje (*drawScene*) nupiešiama visa scena pagal pav.19 parodytą sekų diagramą. Ši dalis užtrunka daugiausia laiko, nes joje praeinami visi piešiami objektai ir atliekami apšvietimo efektai. Paskutiniame etape (*drawFrameEnd*) atliekami „post-process“ efektai ir rezultatas atvaizduojamas kompiuterio lange.

Viena iš svarbiausių sistemos dalių yra trimatę sceną reprezentuojanti klasė. Šios klasės tikslas laikyti visus reikiamus duomenis apie sceną: objektų pozicijas, medžiagas, dydžius, taškinių šviesų pozicijas, jų spalvas ir intensyvumus, saulės spalvą ir kryptį bei virtualios kameros duomenis. Scena turi du pagrindinius metodus kurie leidžia į sceną įdėti piešiamą objektą ar šviesą. Pagrindiniai metodai ir klasės pavaizduotos 21 paveikslėlyje.



***pav. 21**: Atvaizdavimo sistemos klasių diagrama.*

Atvaizdavimo sistema (*Renderer*) yra didelė, monolitinė klasė turinti daug OpenGL specifinių kintamųjų, todėl diagramoje pavaizduoti tik aktualiausi kintamieji ir metodai. Klasių diagramoje galima matyti tris nestandartinius tipus: *Gluint, vec3, mat4*. Tipai su prefiksu „GL“ yra specifiniai OpenGL tipai, kurie dažniausiai atitinka standartinius tipus, tačiau jais galima pasiekti vidinius OpenGL resursu tokius (įkeltus modelius, tekstūras, šeiderius), *vec3* yra trimatis vektorius sudarytas iš trijų *float* tipo kintamųjų, *mat4* yra 4x4 matrica sudaryta iš šešiolikos *float* tipo kintamųjų. Atvaizdavimo sistemos klasė pagrinde turi medžiagų (*Material*), šeiderių (*Shader*) ir modelių (*Mesh*) tipų masyvus. DrawItem klasės nariai (model, material, shader) yra indeksai į šiuos masyvus. Scenos klasė (*Scene*) turi masyvą *DrawItem* tipo objektų.

Medžiagos (*Material*) klasė reprezentuoja fiziškai tikslaus apšvietimo modelio (PBR) medžiagą. Klasė turi spalvos (*albedo_map*), normalių (*normal_map*), šiurkštumo (*roughness_map*) ir metališkumo (*metalic_map*) žemėlapius. Kadangi kartais yra naudinga nupiešti objektą be tekstūrų, egzistuoja ir paprasti šių parametrų aprašymai (*albedo, roughness, metalic*). Šitoks piešimo būdas suteikia mažiau detalių, nes visam objektui suteikiama viena spalva, šiurkštumas ir metališkumas, o normalės vektorius paimamas iš viršūnių duomenų.

Modelio (*Mesh*) struktūra laiko OpenGL viršūnių masyvo identifikatorių (*vao, glVertexArray*) ir viršūnių skaičių. Šis skaičius reikalingas, nes kartais yra naudinga piešti ne visą modelį, o dalį jo.

Šeiderio (*Shader*) klasė yra maža abstrakcija visos šeiderio programos. Suteikiami metodai, kurie padeda užkrauti GLSL kodą, nuskaityti arba nustatyti kintamuosius. Kintamieji kurie gali būti nustatomi iš centrinio procesoriaus į šeiderio programą vadinami „*Uniform*“ kintamieji. Paprastai GLSL kodas yra rašomas kiekvienam šeideriui atskirai – viršūnių, geometrijos ir pikselių šeideriai atskiruose failuose. Kadangi šie šeideriai perduoda duomenis vienas kitam, yra labai patogu viską rašyti viename faile. Taip kaip daroma HLSL, kai naudojama Direct3D technologija. Tam pasiekti parašytas labai paprastas preprocesorius, kuris ieško atitinkamų žodžių ir vieną failą išskaido į atskirus (žr. pav. 22).

```
#VERTEX
#version 430 core

layout(location = 0) in vec3 v_position;
layout(location = 1) in vec3 v_normal;

uniform mat4 mvp_mat;
void main()
{
    gl_Position = mvp_mat*vec4(v_position, 1
}

#FRAGMENT
#version 430 core

void main()
{
    gl_Color = vec4(1, 0, 0, 1);
}
```

***pav. 22**: GLSL kodo pavyzdys.*

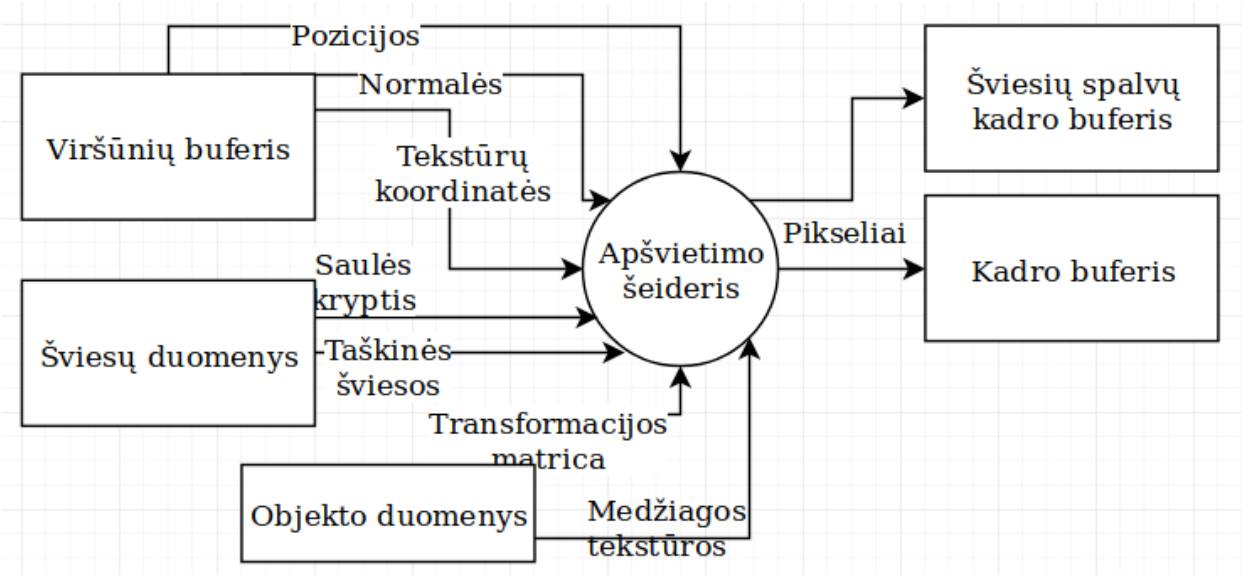
Paveikslėlyje parodytas GLSL kodo pavyzdys. Šis šeideris paima pozicijos koordinates iš viršūnių buferio, kuris buvo užpildytas ankščiau. Kiekviena viršūnė šiame buferyje yra sudauginam su MVP (model-view-projection) matrica, kuri šias pozicijas iš virtualaus pasaulio erdvės paverčia į dvimates ekrano koordinates, o pikseliai nuspalvinti raudona spalva. Ši GLSL kalba yra labai paprastai praplėsta naudojant *#VERTEX* ir *#FRAGMENT* žymėjimus kode, kad būtų galima atskirti viršūnių ir pikselių šeiderius atskirai iš vieno failo, nes to reikalauja OpenGL specifikacija.

Kameros klasė yra atsakinga už virtualios kameros sistemą. Tokio dalyko kaip kamera, vaizdo plokščių programavimo technologijose nėra. Kad sukurti kameros efektą, reikia visą sceną transformuoti priešingai. Tai reiškia, kad kameros sistema yra atsakinga už matricų, kurios galėtų transformuoti viršūnes iš virtualaus pasaulio erdvės į ekrano erdvę. Virtuali kamera turi dvi pagrindines matricas šiam efektui atlikti. Pirma yra vaizdo (angl. „View“) matrica, kuri pagal kameros poziciją ir orientaciją perkelia visus objektus į tinkamą koordinačių sistemą. Antroji yra projekcijos (angl. „Projection“) matrica, kuri paima vaizdo matricos rezultatus ir juos perkelia į ekrano koordinačių sistemą. Projekcijos matricos gali būti įvairios, tačiau dažniausiai naudojamos ortografinės arba perspektyvos. Perspektyvos projekcijoje tolimesni objektai atrodo mažesni, todėl šio tipo projekcijos matrica labai tinka trimačiai virtualiai scenai atvaizduoti. Kadangi matricas galima kombinuoti jas sudauginant, dažnai kameros sistemos turi galimybę gražinti šią kombinuotą matricą. Ši matrica (pav. 21) kameros klasėje gaunama metodu *getViewProjectionMatrix()*.

3.5 Efektų projektavimas

3.5.1 Apšvietimas

Su sukurta atvaizdavimo sistema galima pradėti implementuoti grafinius efektus. Apšvietimo efektams reikalingi scenos apšvietimo duomenys: saulės pozicija ir kryptis, taškinių šviesų pozicijos, kryptys, spalvos ir jų intensyvumai. Šiuos parametrus galima paduoti kaip „Uniform“ kintamuosius šeideriams. Pagrindinio apšvietimo ir šešėlių efektas atvaizduotas konteksto diagramoje (žr. pav. 23).

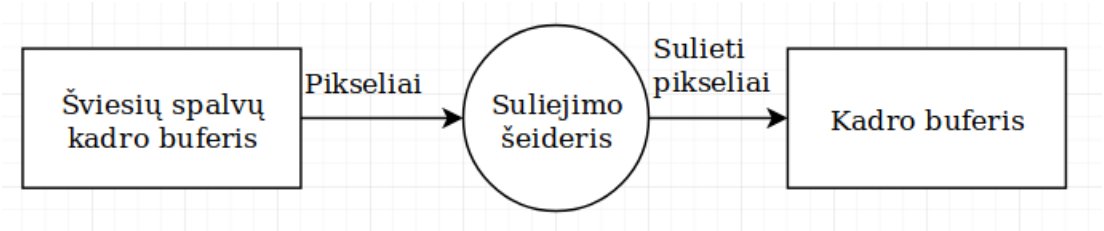


***pav. 23**: Apšvietimo sistemos konteksto diagrama.*

Apšveitimo šeideris paima visus reikiamus duomenis iš viršūnių buferio (*Mesh*), šviesų duomenų (*LightInfo*) ir objekto (*DrawItem*), atlieka skaičiavimus ir rezultatus įrašo į kadro buferį. Gauti kadro buferiai gali būti panaudojami tolimesniems „post-process“ efektams realizuoti.

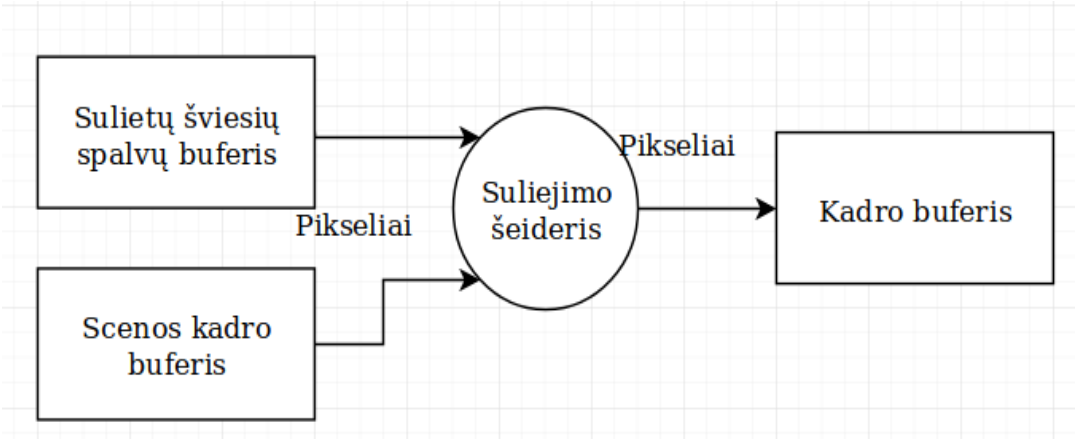
3.5.2 Švytėjimas

Švytėjimo efektui reikalingi du šeideriai. Pirmasis šeideris sulieja šviesias scenos spalvas (žr. pav. 24), o antrasis jas sudeda su esama scenos tekstūra (žr. pav. 25).



***pav. 24**: Suliejimo šeiderio konteksto diagrama.*

Šviesių spalvų kadro buferis yra suliejamas „Gaussian blur“ algoritmu.



***pav. 25**: Švytėjimo efekto šeiderio konteksto diagrama.*

Sulietos šviesios spalvos ir scenos kadro buferis sudedami į galutinį efekto rezultatą.

3.5.3 HDR

Nenormalizuotos HDR kadro buferio reikšmės yra perkeliamos į [0.0;1.0] reikšmių aibę. Darbe implementuojami du algoritmai.

$$c = 1 - e^{(-c_h \cdot x)}; \quad (1)$$

čia c – normalizuota spalva;
c_h – HDR reikšmė;
x – išryškinimo parametras;

$$c = \frac{c_h}{(c_h + 1)}; \quad (2)$$

čia c – normalizuota spalva;
c_h – HDR reikšmė;

Antroji formulė yra paprastesnė ir nesuteikia jokios kontrolės, o antrosios efektyvumas reguliuojamas išryškinimo parametru x. Darbe sukurtas šeideris leidžia pasirinkti norimą algoritmą.

3.5.4 Gama korekcija

Gama korekcija atliekama dviem veiksmais. Visų pirma užkraunamos tekstūros perkeliamos į tolygią erdvę, tada po visų skaičiavimų gražinama atgal į sRGB erdvę. Pirmoji dalis atliekama paduodant atitinkamą parametrą užkraunant tekstūrą (žr. pav. 26.).

```
45  GLuint texture2DLoad(Image* image, int flags)
46  {
47      GLuint texture;
48      GLCALL(glGenTextures(1, &texture));
49      GLCALL(glBindTexture(GL_TEXTURE_2D, texture));
50      GLenum internal_format = GL_RGBA;
51      if(flags & TEXTURE_SRGB)
52          internal_format = GL_SRGB8_ALPHA8;
53      GLCALL(glTexImage2D(GL_TEXTURE_2D, 0, internal_format, image->w, image->h, 0,
54              GL_RGBA, GL_UNSIGNED_BYTE, image->data));
55
56      setTextureParams2D(flags);
57      return texture;
58  }
```

***pav. 26**: Tekstūros užkrovimo funkcija.*

Sukurta funkcija paima *Image* struktūros rodyklę, kuri laiko pikselių duomenis reikiamu formatu atmintyje ir vėliavėlių parametrą *flags* kuris parodo ar dabar užkraunama tekstūra yra sRBG erdvėje. OpenGL funkcija, kuri atlieka užkrovimą į vaizdo plokštės atmintį vadinama *glTexImage2D.*

Galiausiai lieka pikselių šeideryje perkelti spalvos reikšmę atgal į sRGB erdvę. Tai padaroma pasinaudojant formule:

$$c = c^{(\frac{1}{g})}; \quad (3)$$

čia c – spalva;
g – gama konstanta;
Gama konstanta (g) beveik visada yra lygi 2.2, tačiau kartais naudinga leisti šią konstantą modifikuoti, taip leidžiant naudotojui pakoreguoti spalvas.

Visi „post-process“ efektai yra kombinuojami ir implementuojami viename pikselių šeideryje (žr. pav. 27).

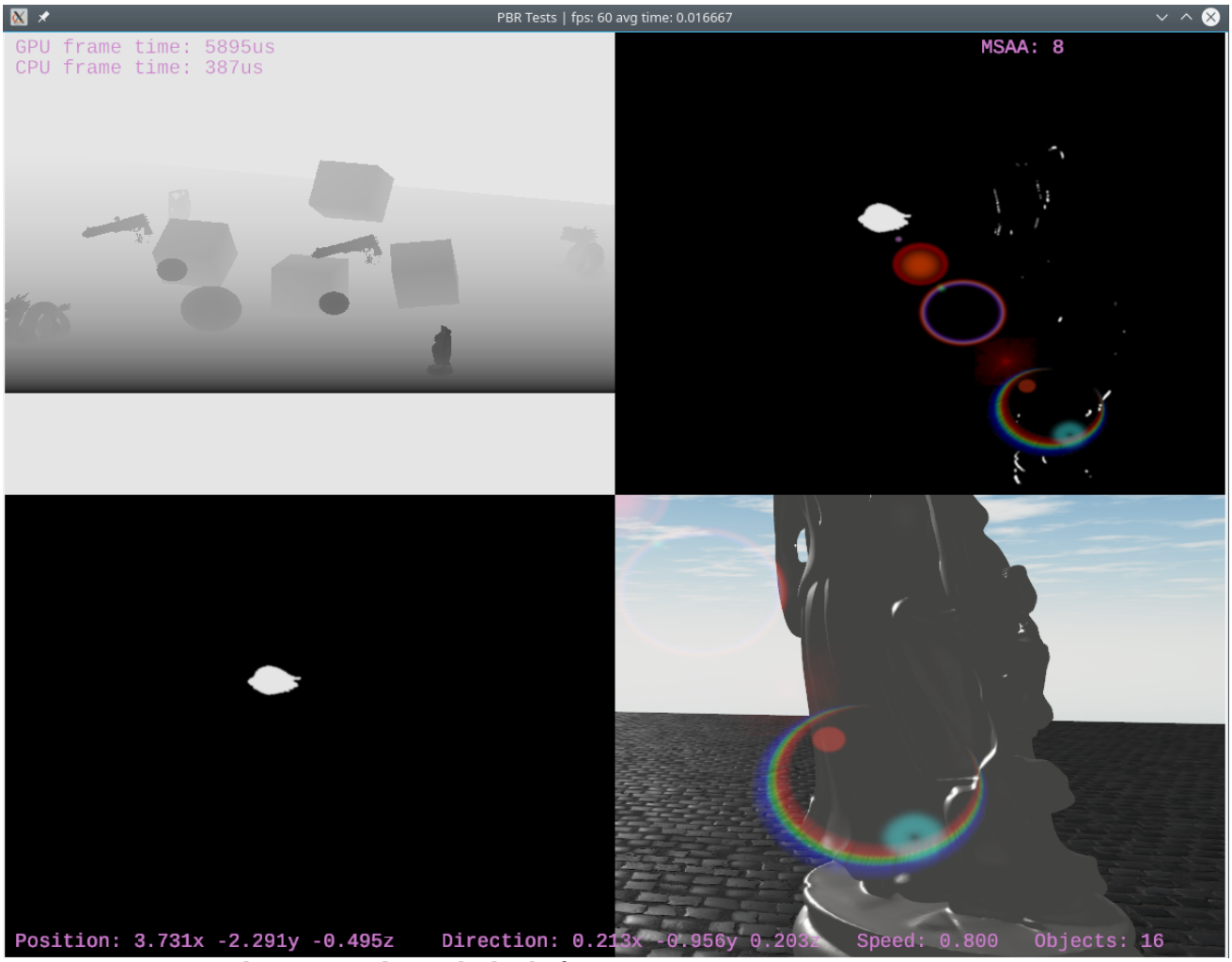
```
75  void main()
76  {
77      if(false){
78          vec4 color = texture(textureMap, uv);
79          color.rgb = pow(color.rgb, vec3(1.0/gamma));
80          out_color = color;
81      }else if(false){
82          vec3 hdrcolor = texture(textureMap, uv).rgb;
83          vec3 mapped = hdrcolor / (hdrcolor + vec3(1.0));
84          mapped = pow(mapped, vec3(1.0 / gamma));
85          out_color = vec4(mapped, 1);
86      }else if(true){
87          vec3 hdrcolor = texture(textureMap, uv).rgb;
88          // hdrcolor = texture(bloom_map, uv).rgb;
89          //if(use_bloom)
90          if(true)
91          {
92          vec3 bloom_color = texture(bloom_map, uv).rgb;
93          hdrcolor += bloom_color*0.85 + scattering().rgb;
94          }
95          vec3 mapped = vec3(1) - exp(-hdrcolor * exposure);
96          mapped = pow(mapped, vec3(1.0 / gamma));
97          out_color = vec4(mapped, 1);
98      }
99  }
```

***pav. 27**: „Post-process“ efektų šeideris.*

Šiame šeideryje atliekami visi „post-process“ efektai: HDR, švytėjimo (*bloom*), šviesos išsiskaidymo (*scattering*) ir gama korekcija (96 eilutė).

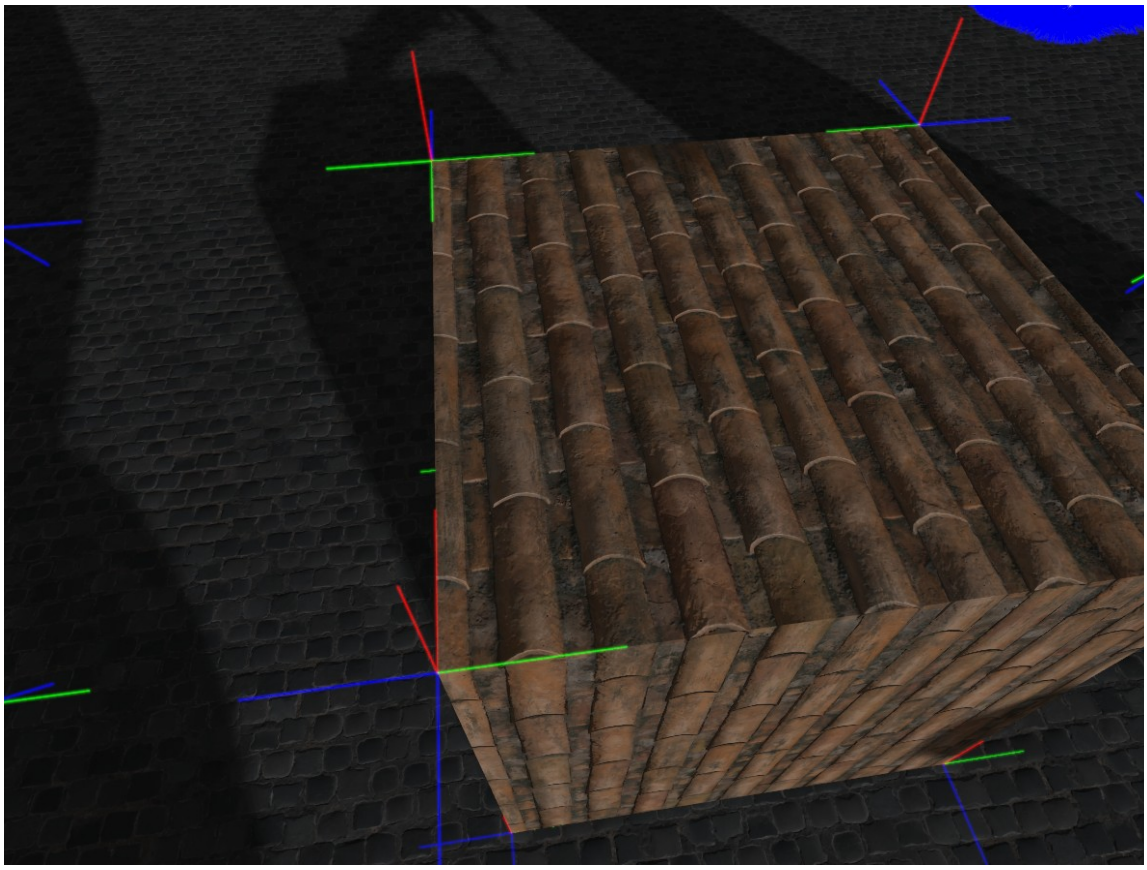
4 Realizacija ir testavimas

Realaus laiko grafinių programų rezultatai yra dvimačiai pikselių masyvai, kurie yra atvaizduoti kompiuterio ekrane. Šie rezultatai nėra nuspėjami iš anksto ir automatiškai jų patikrinti neįmanoma, tačiau galima patikrinti svarbiausius duomenis, pagal kuriuos atliekami skaičiavimai ir implementuojami grafiniai efektai. Juos patikrinti galima parašyti atskirą šeiderį, kuris atvaizduotų visus kadro buferius (žr. pav. 28) ir normalės vektorius (žr. pav. 29).



***pav. 28**: Atvaizduojami naudojami kadro buferiai.*

Paveikslėlyje parodytas vaizdas kuriama atvaizduojami trys papildomi kadro buferiai ant pagrindinės scenos vaizdo. Pirmasis kadro buferis turi scenos gylio informaciją nupieštą iš šviesos šaltinio (saulės) perspektyvos. Šis buferis reikalingas šešėliams atvaizduoti. Antrajame kadro buferyje galima matyti šviesias scenos spalvas (šviesos šaltinis, lęšio žibėjimo tekstūros, šviesūs atsispindėjimai nuo skulptūros). Šis buferis reikalingas švytėjimo efektui sukurti. Trečiasis buferis laiko tik šviesos šaltinius, pagal kuriuos sukuriamas šviesos išsiskaidymo efektas. Likusi lango dalis parodo atvaizduojamos scenos dalį.



pav. 29*: Normalės vektorių erdvės atvaizdavimas.*

Paveikslėlyje pavaizduotas objektas ir jo normalės, tangentinės ir bitangentinės erdvės vektoriai. Šie vektoriai turi sudaryti ortonormuotą vektorių bazę. Tai reiškia, kad šie vektoriai yra statmeni vienas kitam ir normalizuoti.

5 Dokumentacija naudotojui

6 Rezultatų apibendrinimas ir išvados

1. Išanalizuotos visos grafikos programavimo technologijos ir nustatyta, kad OpenGL labiausia atitiko darbo reikalavimus.
2. Sukurta atvaizdavimo sistema leidžia patogiai sukurti trimatę sceną ir ją atvaizduoti, taip pat suteikia galimybę keisti šeiderius ir matyti rezultatus realiu laiku kas padeda derinti algoritmų nustatymus ir pasiekti norimus efektus.
3. Implementuoti visi reikalauti grafiniai efektai ir nustatyta, kad šešėlių efektas yra brangiausias greitaveikos atžvilgiu.
4. Ištestuoti grafiniai efektai, tačiau viena programinės ir aparatinės įrangos konfigūracija klaidingai atvaizduoja šešėlių efektą.

1. *išanalizuoti technologijas naudojamas 3D vaizdo efektų kūrimui;*
2. *sukurti 3D atvaizdavimo sistemą;*
3. *implementuoti pasirinktus grafinius efektus;*
4. *ištestuoti sukurtą sistemą ir efektus;*

Išvados

1. <Atlikus esamų sprendimų analizę galima teigti, jog..>
2. <Atlikus technologijų analizę,... >
3. <Projektavimo metu buvo ..., kas leido... >
4. <Realizuojant sistemą .. >
5. <Atliekant sistemos testavimą.. >

7 Literatūros sąrašas