KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Sergejus Topolovas

Grafiniu procesoriumi grįstas uždengtos geometrijos atrinkimo algoritmas

Magistro darbas

Darbo vadovas
dr. Tomas Blažauskas

Kaunas, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Sergejus Topolovas

Grafiniu procesoriumi grįstas uždengtos geometrijos atrinkimo algoritmas

Magistro darbas

Vadovas
dr. Tomas Blažauskas
2011-05

Atliko
IFM-9/2 gr. stud.
Sergejus Topolovas
2011-05-23

Recenzentas
dr. Armantas Ostreika
2011-05

Kaunas, 2011

Turinys

Grafiniu procesoriumi grįstas uždengtos geometrijos atrinkimo algoritmas

Santrauka

Uždengtos geometrijos atrinkimas – tai būdas nustatyti geometriją, kuri yra uždengta su kita geometrija ir dėl to gali būti nevaizduojama, nes neturės jokios įtakos vaizduojamam paveikslui. Tokios geometrijos nevaizdavimas didina vaizdavimo procedūros našumą.

Egzistuoja eilė uždengtos geometrijos nustatymo būdų, iš kurių vienas yra hierarchinis uždengtos geometrijos atrinkimo algoritmas. Šiame darbe yra analizuojami uždengtos geometrijos nustatymo būdai bei nagrinėjamos pasirinkto algoritmo veikimo spartinimo galimybės panaudojus DirectCompute technologiją. Ši technologija yra Microsoft DirectX 11 bibliotekų rinkinio dalis, kuri leidžia panaudoti grafinį procesorių bendro pobūdžio skaičiavimams. Darbe iškeltų tikslų pasiekimui yra realizuotos kelios bazinės algoritmo versijos modifikacijos, atliekami modifikuotų versijų veikimo laiko bei įvairių veikimo laiką įtakojančių faktorių tyrimai. Yra aptariami gauti rezultatai bei pateikiamos išvados.

Raktiniai žodžiai

Uždengtos geometrijos atrinkimas, grafinis procesorius, gylio žemėlapis, grafinio procesoriaus panaudojimas bendro pobūdžio skaičiavimams, vaizdavimo laiko mažinimas, DirectX 11, DirectCompute.

Graphics processor-based occlusion culling algorithm

Summary

Occlusion culling is a method, which task is to determine geometry occluded with other geometry. Rendering this geometry is useless because it wouldn't impact rendered picture in any way, so discarding it will improve render time.

There are various methods to determine occluded geometry and hierarchical occlusion culling is one of them. This document contains a short summary of these methods, but it's mainly focused on improving hierarchical occlusion culling algorithm performance by making use of DirectCompute technology. This technology is a part of Microsoft DirectX 11 API, which helps the developer to use graphics processor for general-purpose computation. Main goal is reached by performing in-depth analysis of implemented hierarchical occlusion culling algorithm modifications. This analysis consists of both general performance and various performance-related analyses. Further down the road conclusions and recommendations are given based on performed work and overall results.

Keywords

Occlusion culling, graphics processor, depth buffer, making use of graphics processor for general-purpose computation, improving rendering time, DirectX 11, DirectCompute.

1. TERMINŲ BEI SANTRUMPŲ ŽODYNAS

CPU (angl. *Central Processing Unit*) – centrinis procesorius.

GPU (angl. *Graphics Processing Unit*) – grafinis procesorius.

GPGPU (angl. *General-Purpose computation on Graphics Processing Unit*) – būdas, leidžiantis atlikti dažniausiai su centriniu procesoriumi atliekamus bendro pobūdžio skaičiavimus grafinio procesoriaus pagalba.

DirectX 11 – naujausias Microsoft kompanijos bibliotekų rinkinys, skirtas palengvinti vaizdavimą bei su juo susijusių uždavinių įgyvendinimą.

Vaizdavimas (angl. *rendering*) – procesas, kurio eigoje yra sudaromas modelio (-ių) atvaizdo paveikslas pagal pateiktus modelio (-ių) duomenis.

Rastrinis vaizdavimas (angl. *rasterization*) – metodas, kurio dėka vektoriniu formatu aprašyta geometrija yra konvertuojama į ją atitinkantį pikselių atvaizdą.

Vaizdavimo procedūra (angl. *rendering pipeline, graphics pipeline*) – rastrinio vaizdavimo principu paremtas 3D geometrijos vaizdavimo metodas naudojamas grafiniuose procesoriuose. Vaizdavimo procedūra yra sudaryta iš įvairių etapų, kai kurie iš jų yra programuojami pasitelkiant specialias grafines paprogrames.

Grafinė paprogramė (angl. *shader*) – vienos iš kelių galimų programuojamų vaizdavimo etapų instrukcijų seka.

Skaičiavimų paprogramė (angl. *compute shader*) – DirectCompute technologijos dalis; paprogramė, aprašanti bendro pobūdžio skaičiavimų etapą.

Verteksų paprogramė (angl. *vertex shader*) – paprogramė, aprašanti vaizdavimo procedūros etapą, kuriame yra dirbama su verteksais.

Geometrinė paprogramė (angl. *geometry shader*) – paprogramė, aprašanti vaizdavimo procedūros etapą, kuriame yra dirbama su geometrija (taškai, trikampiai). Šis etapas leidžia naikinti arba kurti naują geometriją esamos geometrijos apdorojimo eigoje.

Pikselių paprogramė (angl. *pixel shader*) – paprogramė, aprašanti vaizdavimo procedūros etapą, kuriame yra dirbama su pikseliais.

Uždengtos geometrijos atrinkimas (angl. *occlusion culling*) – metodas, leidžiantis išskaičiuoti geometriją, kuri ją atvaizdavus bus uždengta kita geometrija projektuojant vaizdą iš vartotojo apžvalgos taško perspektyvos.

Geometrijos, nepatenkančios į vartotojui matomą zoną, atrinkimas (angl. *frustum culling*) – metodas, leidžiantis išskaičiuoti geometriją, kuri ją atvaizdavus nepateks į vartotojui matomą zoną projektuojant vaizdą iš vartotojo apžvalgos taško perspektyvos.

Objektų gylio žemėlapis (angl. *depth map*) – tekstūra, kurioje vienos spalvos komponentė (dažniausiai – raudona) atvaizduoja vaizduojamoje vietovėje esančių objektų gylį. Arčiau vartotojo apžvalgos taško esančių objektų gylio reikšmė yra mažesnė negu toliau esančių.

2. ĮVADAS

Šiuolaikinei 3D vaizdavimą atliekančiai programinei įrangai yra keliami labai aukšti vaizduojamo paveikslo detalumo kokybės reikalavimai. To pasėkoje labai didelis dėmesys yra skiriamas vaizdavimo procedūros optimizavimui. Vienas iš esminių vaizdavimo optimizavimo principų – nevaizduoti to, kas nebus matoma vartotojui iš jo 3D aplinkos matymo taško. Tai yra įmanoma dviem atvejais: vaizduojamas objektas nepapuola į vartotojui matomą zoną arba vaizduojamas objektas yra pilnai uždengtas kitu vartotojui matomu objektu. Pastarajam spręsti yra išrasta eilė įvairiausių vaizduojamos vietovės hierarchija paremtų algoritmų, kurie yra išnagrinėti šio darbo 2 skyriuje. Šie algoritmai leidžia neatliekant pilno objektų geometrijos atvaizdavimo nustatyti kurie objektai yra nematomi vartotojui ir todėl neturi būti vaizduojami taip sutrumpinant bendrą vaizduojamos vietovės laiką.

Su kiekvienais metais vis labiau augantis grafinių (angl. *Graphics Processing Unit*; toliau – GPU) bei centrinių (angl. *Central Processing Unit*; toliau – CPU) procesorių pajėgumas verčia programinės įrangos kūrėjus ieškoti naujų būdų bei metodų jo efektyviam panaudojimui [19]. Grafinio bei centrinio procesorių našumo augimo nevienodumas ir vis labiau didėjantis grafinio procesoriaus našumo atotrūkis paskatino bendro pobūdžio skaičiavimų vykdymo grafinio procesoriaus pagalba (angl. *General-Purpose computing on Graphics Processing Unit*; toliau – GPGPU) technologijų atsiradimą. Šios technologijos, tokios kaip CUDA [14], OpenCL [8], Accelerated Parallel Processing (APP) [1], DirectCompute [2], ir kt., įgalina bei supaprastina išlygiagretinamų algoritmų vykdymą GPU pagalba.

GPGPU technologijos gali būti panaudotos sprendžiant tiek susijusius su grafikos atvaizdavimu tiek bendrinio pobūdžio uždavinius. Nors GPGPU vystymu ir buvo siekta užtikrint būtent pastarųjų spartinimą, tačiau dabar ne retai šios technologijos yra naudojamos kaip pagalbinės įvairaus pobūdžio 2D bei 3D grafikos vaizdavimu pagristoje programinėje įrangoje. Vienas iš geriausių ir plačiausiai sutinkamų tokios programinės įrangos pavyzdžių yra kompiuteriniai žaidimai. Būtent jie iš esmės ir stumia į priekį visą grafinių procesorių rinką, o tuo pačių ir su jais susijusias technologijas.

GPU pajėgumo augimas bei su tom susijusių programinės įrangos technologijų kaita verčia permastyti įvairių algoritmų realizacijos koncepcijas bei ieškoti naujų būdu esamų technologijų panaudojimui. Ne išimtis yra ir uždengtos geometrijos atrinkimo problema, ties kurios naujų sprendimo būdų paieška ir yra koncentruojamasi šiame darbe. Čia taipogi dėmesys

bei pirmenybe yra teikiamos GPGPU sprendimams bei bendram atliekamų skaičiavimų nukėlimo iš CPU į GPU klausimui. Manome, kad tai dabar yra ypatingai aktualus klausimas ir jo svarba bėgant laikui vis labiau augs.

Šio darbo pagrindiniai tikslai yra tokie:

- Ištirti GPGPU galimybių sutiekiamą naudą atliekant uždengtos vaizduojamų objektų geometrijos atrinkimą.
- Ištirti įvairių faktorių poveikį realizuoto algoritmo našumui siekiant pagerinti jo kiekybines bei kokybines charakteristikas su tiriama GPU architektūra.

Šiems tikslams įgyvendinti yra pasirinktas hierarchinis objektų gylio žemėlapiu (angl. *depth map*) paremtas uždengtos geometrijos atrinkimo algoritmas, kas ir yra šio darbo objektas. Šiame darbe yra nagrinėjamas iškeltos uždengtos geometrijos atrinkimo problemos sprendimo budai, yra siūlomos kelios pasirinkto algoritmo modifikacijos, realizuotos algoritmo versijos naudojančios tiek CPU, tiek GPU, bei atliekami realizuotų versijų tyrimai. Realizacijose yra siekiama maksimaliai į jas įtraukti grafinį procesorių. Darbo pabaigoje yra aptariami gauti rezultatai, pateikiamos bendros algoritmo naudojimo DirectCompute kontekste rekomendacijos ir pasiūlytų modifikacijų įvertinimas.

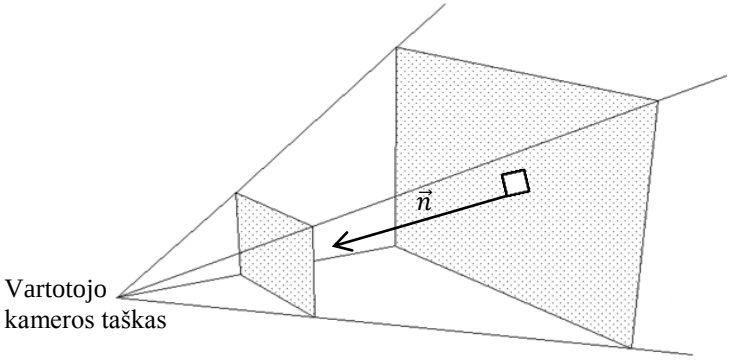
3. ANALITINĖ DALIS

3.1. Vaizdavimo procedūros spartinimas nematomos geometrijos dėka

Šiame skyriuje yra aptariami egzistuojantys būdai bei metodai skirti paspartinti vaizdavimo procedūrą atmetant nereikalingą vaizduojamos aplinkos geometriją.

3.1.1. Geometrijos, nepatenkančios į vartotojui matomą zoną, atrinkimas

1 paveiksle yra pateiktas 3D perspektyvinio projektavimo metu gaunamas erdvės regionas, kuriame esanti geometrija bus matoma vartotojo ekrane. Šis regionas sudaro figūrą, kuri dar yra vadinama nupjautine piramide (angl. *frustum box*). Visa geometrija bei objektai, kurie gali būti matomi vartotojui, turi papulti į figūros vidų.



1 pav. Vaizdavimo atlikimo metu ekrane matomo vaizdo zonos ribos.
(paimta iš http://paulbourke.net/miscellaneous/frustum/)

Nėra jokios prasmės vaizduoti tai, kas nesikerta su bent viena iš šią erdvę ribojančių plokštumų arba nėra šios erdvės viduje. Procedūra, kurios dėka yra filtruojama geometrija nepapuolanti į šią erdvę yra vadinama geometrijos, nepatenkančios į vartotojui matomą zoną, atmetimu (angl. *frustum culling*) [5].

Kiekvieną iš plokštumų galima aprašyti tokia lygtimi:

$$Ax + By + Cz + D = 0 \tag{1}$$

Čia *A*, *B*, *C* ir *D* – konstantos, kurios gali buti apskaičiuotos pagal bet kuriuos tris plokštumos taškus:

$$A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2) \tag{2}$$

$$B = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2) \tag{3}$$

$$C = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \tag{4}$$

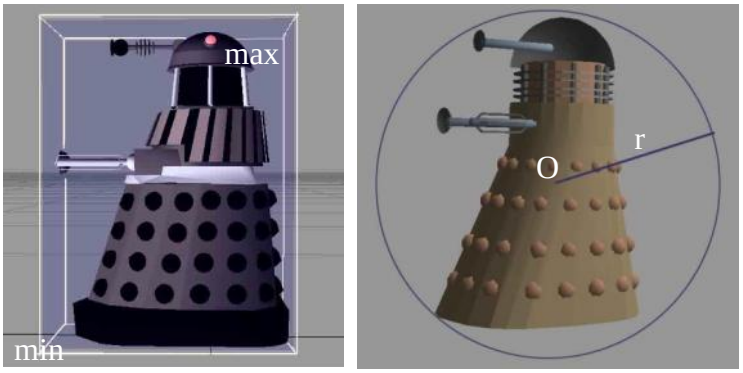
$$-D = x_1(y_2z_3 - y_3z_2) + x_2(y_3z_1 - y_1z_3) + x_3(y_1z_2 - y_2z_1) \tag{5}$$

(A, B, C) koordinatės atitinka normalės vektoriaus $\vec{n}$ pradžios tašką 1 paveiksle, o *D* – plokštumos atstumą nuo koordinačių pradžios taško normalės kryptimi. Taigi, norint patikrinti ar taškas yra projektuojamo vaizdo erdvės ribų viduje reikia tikrinamo taško koordinates įstatyti į kiekvienos iš 6 ribojančių plokštumų lygtis. Tokiu atveju galimi tris kiekvienos lygties rezultatų interpretavimo variantai:

- Rezultatas didesnis už 0: taškas yra teigiamoje plokštumos normalės vektoriaus $\vec{n}$ pusėje.
- Rezultatas mažesnis už 0: taškas yra neigiamoje plokštumos normalės vektoriaus $\vec{n}$ pusėje.
- Rezultatas lygus 0: taškas yra ant plokštumos.

Jeigu taškas yra teigiamoje visų plokštumų normalių vektorių pusėje – jis yra projektuojamos erdvės viduje. Bent vieno iš šešių didesnio už 0 rezultato atveju galima teigti, kad tas taškas bus matomas vartotojui, taigi jį reikia atvaizduoti.

Siekiant optimizuoti tikrinimą jis yra atliekamas ne kiekvieno vaizduojamo trikampio taškams, o trikampiais sudaryto objekto (angl. *mesh*) minimaliai dengiančiai figūrai. Dengianti figūra gali būti dėžės (angl. *bounding box*) arba sferos (angl. *bounding sphere*) formos. Minimali dengianti dėžė yra aprašoma *min* ir *max* kampų koordinatėmis, o minimali dengianti sfera – centro koordinatėmis *O* bei spinduliu *r*.



2 pav. Minimali dengianti dėžė (kairėje) ir minimali dengianti sfera (dešinėje).
(paimta iš http://www.toymaker.info/Games/html/collisions.html)

Naudojant minimalią dengiančią dėžę užtenka patikrinti kiekvieno arba bent kelių jos kampinių taškų patekimo į nupjautinės piramidės vidų sąlygą pagal aukščiau aprašytą metodą. Tuo tarpu siekiant patikrint ar sfera yra teigiamoje plokštumos normalės vektoriaus $\vec{n}$ pusėje reikia tikrinti tokią sąlygą [18]:

$$\overrightarrow{(A,B,C,D)} \cdot \overrightarrow{(Ox,Oy,Oz,1)} + r > 0 \quad (6)$$

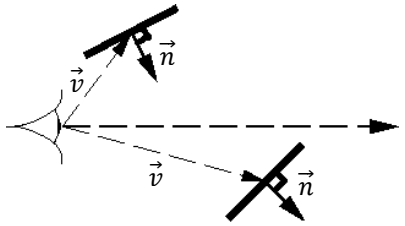
Jeigu ši sąlyga yra tenkinama bent vienai nupjautinės piramidės plokštumai – dengianti sfera kertasi su vartotojui matomą zoną, tad sfera apgaubtas objektas turi būti vaizduojamas ekrane.

3.1.2. Geometrijos su paviršiumi nenukreiptu į vartotojo pusę atrinkimas

Šitas metodas yra automatiškai vykdomas kiekvienam vaizduojamam trikampiui prieš vykdant rastrinio vaizdo formavimą vaizdavimo procedūros metu [17]. Siekiant apskaičiuoti kur yra nukreipta vaizduojamo trikampio plokštuma pakanka apskaičiuoti iš kameros taško į trikampio plokštumą nukreipto vektoriaus $\vec{v}$ bei trikampio normalės vektoriaus $\vec{n}$ skaliarinę sandaugą, kaip parodyta 7 formulėje.

$$\vec{v} \cdot \vec{n} < 0 \quad (7)$$

Jeigu yra tenkinama 7 formulėje pateikta sąlyga – trikampio plokštuma yra nukreipta į vartotojo pusę ir toks trikampis turi būti vaizduojamas. Tačiau kai gaunama reikšmė yra didesnė negu arba lygi 0 – tokių trikampių plokštumos yra nukreiptos ne į vartotojo kameros taško pusę ir nebus matomos vartotojui. Tai yra paremta trikampių viršūnių pateikimo eiliškumu (angl. *winding order*). Taigi tokius trikampius galima iškarto atmesti ir nevaizduoti.



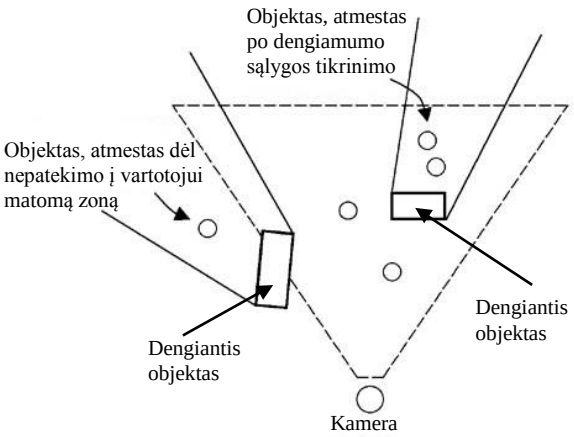
3 pav. Trikampių plokštumų vaizdas iš vartotojo perspektyvos.
(paimta iš http://omega.di.unipi.it/web/IUM/Waterloo/node66.html)

3.1.3. Uždengtos geometrijos atmetimas

Uždengtos geometrijos atrinkimas, dar vadinamas uždengto paviršiaus nustatymu (angl. *hidden surface determination*), – tai metodas, kurio tikslas yra apskaičiuoti kuri geometrija yra pilnai uždengta su kita geometrija žvengiant iš vartotojo kameros taško [20]. Šiam tikslui realizuoti egzistuoja eilė uždengtos geometrijos atmetimo sąlygos išskaičiavimo algoritmų (plačiau 3.1.3.1 – 3.1.3.5 skyriuose). Visi jie yra paremti tuo pačiu principu: jeigu geometrija, projektuojant ją iš vartotojo kameros taško, yra pilnai uždengta kita geometrija – jos vaizduoti nereikia, nes ji vartotojui yra nematoma. Siekiant optimizuoti šitą tikrinimą jis yra atliekamas ne

kiekvienam trikampiui, o visam objektui remiantis to objekto minimalia dengiančia dėže arba minimalia dengiančia sfera. Taip pat nėra prasmės tikrinti objektų dengiamumą tiems objektams, kurie nepraėjo patekimo į vartotojui matomą zoną patikrinimo (detaliau aprašytas 3.1.1. skyriuje). Jis dažniausiai yra atliekamas prieš vykdant bet koki detalesnį uždengtos geometrijos objektų aptikimą.

4 paveiksle yra pateiktas suplotas primityvus galimos vaizduojamos aplinkos vaizdas.



4 pav. Galimas suplotas vaizduojamos aplinkos vaidas su dengiančiais objektais.
(paimta iš http://www.gamecareerguide.com/features/222/book_excerpt_programming_a_.php)

Iš 4 paveikslo matome, kad keli mažesni objektai yra uždengti didesniais objektais ir todėl nėra matomi vartotojui. Tokie objektai neturi patekti į vaizdavimo procedūrą nes jie niekaip neįtakotų suformuoto vartotojui pateikiamo vaizdo. Juos atmetus yra sutaupomas vaizdavimo laikas. Plačiausiai naudojami uždengtos geometrijos išskaičiavimo metodai yra pateikiami 3.1.3.1 – 3.1.3.5 skyriuose.

3.1.3.1. Potencialiai matomos aibės metodas

Potencialiai matomos aibės (angl. *Potencially Visible Set – PVS*) metodas – tai vaizdavimo procedūros spartinimo nematomos geometrijos dėka metodų šeima. Visi čia papuolantys metodai veikia atliekant visos busimos vaizdavimo metu naudojamos aplinkos sudalinimą į blokus bei jų gylio informacijos apskaičiavimą prieš vykdant patį vaizdavimą. Vaizdavimo metu ši informacija yra naudojama tikrinime priklausomai nuo to, kokia vaizduojamos aplinkos zona turi būti matoma vartotojui. Siekiant dar labiau sumažint atliekamų tikrinimų kiekį prieš palyginimą dar yra atliekamas blokų pateikimo į vartotojui matomą zoną testas.

Egzistuoja kelios PVS metodų grupės, kiekviena su jai būdingais metodais [9]:

- Konservatyvus metodai: šie metodai mažina dengiančiuosius objektus siekiant sumažinti jų įtaką tikrinimui. Tokiu būdu yra gaunamas tikslus vaizdas tačiau dažniausiai yra atvaizduojamos nereikalingos vartotojui nematomos objektų detalės.
- Agresyvus metodai: šie metodai siekia maksimaliai sumažinti tikrinamos aibės dydį. Dėl to ne retai atsiranda atvejai kai objekto detalės kurios turi buti matomos yra neatvaizduojamos.
- Aproksimuojantys metodai: šie metodai gali sukelti tiek vaizdavimo netikslumus, tiek perdėtą vaizdavimą.
- Tikslus metodai: metodai kurie labai tiksliai apskaičiuoja dengiamumą. Šie metodai paprastai yra sunkiai realizuojami ir (arba) reikalauja daug procesoriaus laiko.

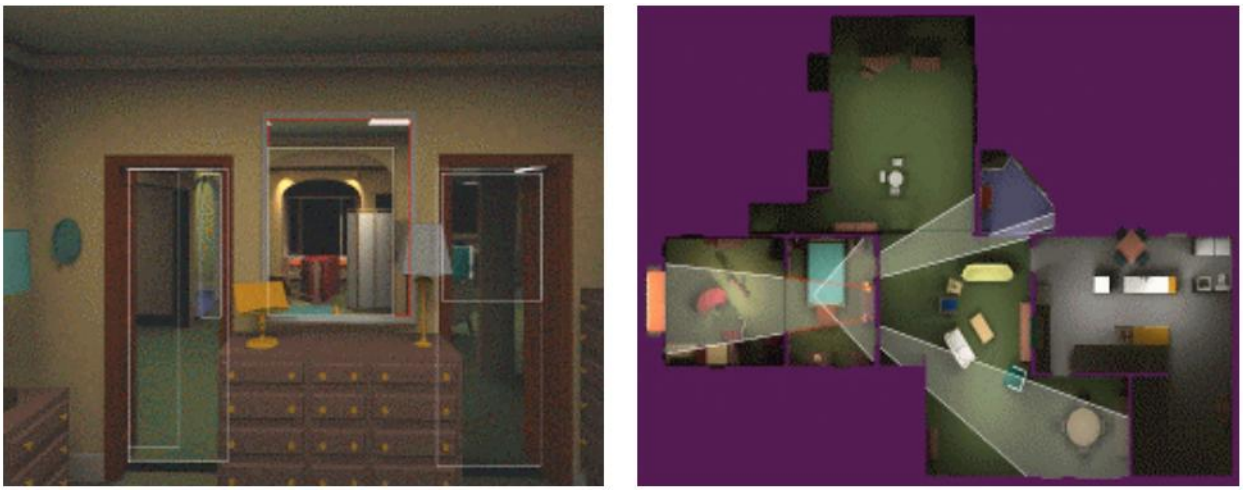
Visi šie metodai padeda greitai atmesti nereikalingą geometriją, tačiau dažniausiai jie gali būti taikomi tik statinei nekeičiančiai savo 3D padėties aplinkos vaizdavimo metu geometrijai. Didžiausias pliusas – nereikia atlikti informacijos perskaičiavimo kiekvieną kadrą ar kas kart pasikeitus vartotojo kameros pozicijai. Visgi visos šios informacijos saugojimas užimą papildomą vietą bei pats apskaičiavimo procesas gali trukti nemažai laiko dirbant su dideliu aplinkos modeliu arba kai pats modelis yra labai sudėtingas.

3.1.3.2. Portalais paremtas vaizdavimo metodas

Portalais paremtas vaizdavimas – tai dar vienas iš vaizdavimo procedūros spartinimo metodų. Šis metodas operuoja erdvės zonomis (angl. *cells*) bei jas jungiančiomis vietomis – portalais. Vaizduojamas pasaulis yra sudalinamas į stačiakampius regionus – erdvės zonas, kurios tarpusavyje jungiasi tik per portalus [9]. Tokiu būdu vartotojas negali būti daugiau negu vienoje erdvės zonoje vienu metu. Yra vaizduojami visi šioje zonoje esantys objektai, prieš tai atliekant tik objektų patekimo į vartotojui matomą zoną tikrinimą.

Per zonos, kurioje tuo metu yra vartotojo kameros taškas, portalus jam gali buti matoma tik tam tikra su šia erdvės zona sujungtų zonų dalis. Ji yra paskaičiuojama atlikus portalų projekciją į tikrinamos zonos erdvę. Su šia projekcija yra atliekamas erdvėje esančių objektų susikirtimo tikrinimas ir gaunamas rezultatas parodo kas turi būti vaizduojama, o kas ne.

Vizualizuotas šio algoritmo variantas yra pateiktas 5 paveiksle. Vartotojas yra kambaryje su dviejomis prieš jį esančiomis durimis bei veidrodžiu. Portalai kairiajame paveiksle atvaizduoti paryškintu kontūru. Dešiniajame paveiksle yra paryškintos portalų vaizdo projekcijos į kitas erdvės zonas. Čia kiekvienas atskiras kambarys sudaro kitą erdvės zoną.



5 pav. Portalų panaudojimo vizualizavimas. Kairėje - vartotojui matomas vaizdas, dešinėje - aplinkos vaizdas iš viršaus.
(paimta iš http://www.tml.tkk.fi/Opinnot/Tik-111.500/2003/paperit/MikkoLaakso.pdf, Figure 2)

Portalais paremtas vaizdavimo metodas žymiai supaprastina nematomos geometrijos atrinkimo bei išmetimo iš vaizdavimo procedūros funkcijos vykdymą. Visgi šitas metodas turi vieną esminį trukumą – dėl savo principo jis yra tinkamas tik uždarų erdvių vaizdavimui.

3.1.3.3. Spindulių trasavimu paremtas metodas

2000 – taisiais metais B. H. Pease pasiūlė uždengtos geometrijos aptikimo metodą, kuris yra paremtas spindulių trasavimų [15]. Norint atrinkti uždengtą geometriją šio metodo pagalba pirmiausiai erdvė yra rekursyviai sudalinama į blokus, kurie yra saugomi aštuntainiame medyje (angl. *oct tree*). Aštuntainis medis – tai medžio tipo duomenų struktūra, kurioje kiekviena viršūnė turi po 8 vaikines viršūnes arba lapus. Dalinimasis pradedamas nuo erdvės centro ir vyksta vis smulkinant turimą erdvę iki tam tikros nustatytos dydžio ribos.

Norint nustatyti kurie šios struktūros lapai yra matomi vartotojui iš vartotojo kameros taško yra „leidžiamas“ tam tikras fiksuotas spindulių kiekis. Pradžioje yra nustatomas oktetas, kuris kertas su vartotojo kameros tašku. Jis bus kiekvieno spindulio pradžios oktetu. Toliau kiekvienam spinduliui yra nustatomas sekantis to spindulio kryptimi einantis su spindulių besikertantis oktetas. Paskui jam sekantis ir t.t. Operacija kartojama kol yra susiduriama su kokiu

nors objektu arba kol spindulys pasiekia tikrinamos zonos ribas. Svarbu paminėti ir tai, kad prieš atliekant palyginimą oktetas turi būti pakreiptas su vartotojo kameros poziciją nusakančia matrica (angl. *traverse*) kad jo pozicijos reikšmė atitiktų vaizdą iš vartotojo kameros perspektyvos (angl. *view space*). Visi objektai, su kuriais susidūrė paleisti spinduliai privalo būti vaizduojami nes yra matomi iš vartotojo kameros taško. Atitinkamai jei spindulys kirto oktetą – visi jo lapuose esantys objektai taipogi turi būti vaizduojami.

3.1.3.4. Hierarchinis uždengtos geometrijos atrinkimo metodas

1998 metais Zhang H. pasiūlė efektyvų objektų hierarchija erdvėje paremtą uždengtos geometrijos apskaičiavimo metodą [20]. Šitas metodas remiasi dengiančių (angl. *occluders*) ir dengiamų (angl. *occludees*) objektų koncepcija. Dengiantys objektai – tai objektai, kurie gali uždengti kitus objektus dėl tam tikrų savo savybių, pavyzdžiui dydžio. Dengiami objektai – tai tokie objektai, kurie gali būti uždengti atvaizdavus dengiančiuosius iš vartotojo kameros taško.

Prieš vykdant uždengtos geometrijos atrinkimą yra labai svarbu nustatyti kurie objektai bus dengiančiaisiais. Nuo šito pasirinkimo priklauso atliekamo tikrinimo tikslumas. Šitas nustatymas yra atliekamas remiantis vienu arba keliais iš žemiau pateiktų variantų:

- Objektų didžiu 3D erdvėje.
- Objektų dydžiu, gautu atlikus šių objektų projekciją iš vartotojo kameros taško.
- Objektų atstumo iki vartotojo kameros taško ir jų dydžio santykine reikšme.
- ir kitais būdais.

Pačios dengimo sąlygos išskaičiavimas yra paremtas dengiančių objektų gylio žemėlapiu (angl. *depth map*), kuris yra dar vadinamas Z – buferiu. 6 paveiksle pavaizduotame vaizduojamos aplinkos gylio žemėlapyje objektų gylis yra atvaizduotas vienos spalvos komponentes pagalba. Galimos gylio reikšmės yra nuo 0 iki 1, kur mažesnės reikšmės reiškia jog ta vieta yra arčiau vartotojo kameros taško. Mažesnės reikšmės 6 paveiksle yra tamsesnės už didesnes, o tai reiškia, kad tamsesnės vietos yra arčiau vartotojo kameros taško nei šviesesnės.

6 pav. Vaizduojamos aplinkos gylio žemėlapis.

Objektų gylio žemėlapis yra formuojamas grafinio įrenginio pusėje ir yra plačiai taikomas ne tik su uždengimo apskaičiavimų susijusiuose uždaviniuose. Vienas iš bazinių ir naudojamų beveik kiekvieno 3D vaizdavimo metu taikymo pavyzdžių yra galutinės pikselių spalvos formavimas [16]. Yra svarbu, kad vaizdavimo metu pikselis įgautų arčiausiai kameros taško esančios geometrijos spalvą, tame praverčia vaizduojamos scenos gylio žemėlapis. Jis taip pat yra naudojamas eilėje įvairių vaizdavimo efektų.

Pasak Zhang H., vaizdavimas su uždengtos geometrijos atmetimu turi būti sudarytas iš šių žingsnių [9]:

1. Dengiančių objektų pasirinkimas ir jų gylio žemėlapio sudarymas.
    a. Objektų, tenkinančių pasirinktą dengiančių objektų sąlygą bei papuolančių į vartotojui matomą zoną, pasirinkimas.
    b. Gylio žemėlapio sudarymas pasirinktiems dengiantiesiems objektams.
    c. Gylio žemėlapio hierarchinės piramidės sudarymas.
2. Potencialiai dengiamų objektų dengiamumo sąlygos tikrinimas bei neatmestų objektų vaizdavimas.
    a. Dengiamų objektų patekimo į vartotojui matomą zoną tikrinimas.
    b. Praėjusių (neatmestų) 2a žingsnį objektų gylio palyginimas su 1 žingsnyje suformuotu gylio žemėlapiu. Jei objekto gylio reikšmė yra mažesnė negu dengiančiųjų objektų gylio žemėlapyje esanti reikšmė – tas objektas bus matomas vartotojui ir neturi būti atmestas.
    c. Objektų, praėjusių dengiamumo patikrinimą, vaizdavimas.

Hierarchinė gylio žemėlapio piramidė paminėta 1c žingsnyje – tai gylio žemėlapių grandinė, kurioje 0 lygyje yra pateiktas pats detaliausias gylio vaizdas. Kiekvienas sekantis lygis yra dvigubai mažesnis už praeitą, ir jo kiekvieno pikselio reikšmė yra apskaičiuojama kaip

didžiausia prieš tai einančio lygio 4 gretutinių pikselių reikšmė, kas parodyta 8 formulėje. Čia $p(x,y)_n$ – *n*-tojo hierarchinio gylio žemėlapio lygyje *(x, y)* tekstūros koordinatėse esantis pikselis. Šita formulė turi būti taikoma nuo pradedant nuo 1 lygio.

$$p(x,y)_n = \max\big(p_{n-1}(2x,2y), p_{n-1}(2x+1,2y), p_{n-1}(2x,2y+1), p_{n-1}(2x+1,2y+1)\big) \quad (8)$$

1b ir 1c – vieninteliai šio metodo žingsniai, kurie tuo metu buvo atliekami GPU pagalba. Taip buvo todėl, kad grafiniai procesoriai dar neturėjo pakankamai galimybių kažkam daugiau. Tačiau laikui bėgant situacija keitėsi. 2008 – tais metais AMD savo pristatyme parodė kaip įprastos GPU vaizdavimo procedūros dėka galima atlikti uždengtos geometrijos dengiamumo išskaičiavimą [3]. 2010 – tais metais kompanija Ubisoft pristatyme pademonstravo šio metodo veikimą bei naudą savo išleistame kompiuteriniame žaidime Splinter Cell Conviction [7]. Ubisoft taipogi atliko bazinį palyginimą tarp šio metodo ir aparatūrinėmis dengiamumo užklausomis (apie šį metodą plačiau 3.1.3.5 skyriuje).

Naujos technologijos, tokios kaip Microsoft DirectX 11 bibliotekų paketas kartu su DirectCompute technologija bei jos standartus atitinkančia aparatūrine įranga, leidžia šį metodą padaryti dar lankstesnių. To yra pasiekiama atliekant dengiamumo testą kaip ir bet kokius kitus bendro pobūdžio skaičiavimus grafinės skaičiavimų paprogramės dėka (angl. *compute shader*) [2]. Plačiau apie DirectCompute galimybes yra aprašyta 3.2 skyriuje. Naujos technologijos taipogi leidžia optimizuoti algoritmo 1c žingsnį, nes DirectX 10 arba naujesnės technologijos panaudojimas leidžia kurti skirtingos paskirties resursų vaizdus atskiriems gylio piramidės lygiams: skaitymo (angl. *shader resource view*) ir rašymo (angl. *render target view*) [13]. Tai leidžia vienu metu turėti skaitymo bei rašymo teises skirtingiems resurso lygiams (pavyzdžiui skaityt iš 0-io lygio ir rezultatą rašyt į 1-ą ir t.t.) taip panaikint būtinybę atlikti resursų kopijavimą bei kaitaliojimą tarp kelių gylio piramidžių. Tai buvo vienintelis šio žingsnio atlikimo būdas naudojant senesnes technologijas.

3.1.3.5. Aparatūrinių matomumo užklausų metodas

Aparatūrinės matomumo užklausos – tai DirectX API specialios funkcijos. Jų pagalba galima gaut detalią informaciją apie GPU darbo statistiką, patikrint ar baigėsi tam tikra paleista operacija ir t.t. Jos taipogi leidžia patikrinti ar tam tikra geometrija praėjo gylio žemėlapio patikrinimą [4]. Nors šis funkcionalumas egzistuoja jau senai, bet jis nėra plačiai naudojamas dėl dviejų priežasčių: užklausos iškvietimo kaina (kiekviena užklausa – papildomas vaizdavimo

kreipinys) ir užklausos rezultato laukimui sugaištamas laikas. Nors didelis vaizdavimo kreipinių skaičius mažina bendrą programos našumą, užklausos rezultato laukimas irgi yra labai svarbus faktorius. Taip yra todėl kad šios užklausos yra vykdomos asinchroniškai ir jų rezultatas tampa prieinamas vartotojui tik po tam tikro laiko intervalo [12]. Ne gana to, jeigu rezultato nėra laukiamą to paties kadro vaizdavimo metu (nes neefektyvu) – tai ir pati vaizduojama scena per šį laiką gali pasikeisti, kas gali sukelt įvairių vaizduojamo paveikslo netikslumų, tokių kaip iš niekur atsirandantys objektai ir t.t. Tačiau šis metodas yra vykdomas pilnai GPU pagalba, kas sutaupo nemažai CPU resursų.

Standartinis matomumo užklausos vykdymo algoritmas atrodo taip:

1. Objektų išrikiavimas atstumo nuo vartotojo kameros taško didėjimo tvarka.
2. Kiekvienam objektui:
    a. Užklausos objekto sukūrimas.
    b. Rašymo į vaizdavimo gylio buferį bei vaizdavimo rezultatų buferius atjungimas.
    c. Supaprastintos tikrinamos geometrijos figūros atvaizdavimas.
    d. Užklausos sustabdymas bei rezultatų laukimas.
    e. Jei gražintas praėjusių tikrinimą pikselių kiekis didesnis už 0 – objektas bus matomas iš vartotojo taško. Priešingu atvejų jį galimą atmest ir nevaizduoti.

Nors kiekvienas užklausos kreipinys neišvengiamai sunaudoja šiek tiek CPU laiko, pačios užklausos sukeliamas GPU laiko sąnaudas galima sumažint. Tai yra atliekama užklausos metu vaizduojant ne visa tikrinamą geometriją, o jos minimalią dengiančią figūrą (dėžę arba sferą). Norint padidinti tikslumą galimas vaizduojamo modelio sudalinimas į smulkesnius minimalius dengiančius blokus, kurių bendras atvaizdavimas vis tiek bus ne toks resursams imlus procesas kaip pilnas viso modelio vaizdavimas.

Siekiant sumažint užklausos rezultato laukimo įtaką per tą laiką galima vykdyti kitus skaičiavimus, t.y. užklausų laukimą yra rekomenduojama daryti asinchroniškai nuo vaizdavimo. Siekiant sumažinti bendrą vykdomų užklausų kieki galimas BSP ar kitokios medžio – tipo struktūros panaudojimas vaizduojamos aplinkos objektų struktūrizavimui. Tokiu atveju į 2 – ajį algoritmo žingsnį yra kreipiamasi rekursiškai, o užklausų vykdymas yra pradedamas šakniniame objekte. Tolimesnių vaikinių objektų užklausos vykdomos tik jei šakninė užklausa gražino teigiamą tikrinimo rezultatą. Tokiu būdu, kai to reikia, yra perbėgama per visą medį iki lapų,

tačiau bendras atliekamų užklausų kiekis ženkliai sumažėja. Šios optimizacijos efektyvumas tiesiogiai priklauso nuo vaizduojamos aplinkos.

3.2. DirectCompute metodologija

DirectCompute yra Microsoft DirectX 11 programinio bibliotekų paketo dalis, skirta bendro pobūdžio skaičiavimų vykdymui grafinio įrenginio pagalba [2]. Ši technologija yra ypatingai svarbi busimiems DirectX 11 technologija paremtos programinės įrangos kūrėjams, nes priešingai negu jos konkurentai leidžia resursų apsikeitimą tarp DirectX ir DirectCompute. Tai reiškia, kad pavyzdžiui vaizdavimo procedūroje naudotą arba dar tik planuojama naudoti resursą galima skaityti arba modifikuoti skaičiavimų paprogramėje (angl. *compute shader*). Tai suteikia eilę naujų galimybių vaizdavime naudojamų algoritmų modifikavimui arba naujų algoritmų kūrimui.

Programos, skirtos vykdymui su DirectCompute, kaip ir bet kokios kitos grafinės paprogramės atveju, turi būti parašyti kaip metodai aukšto lygio grafinėje kalboje (angl. *High Level Shading Language – HLSL*) [10]. Toks metodas ir yra vadinamas skaičiavimo paprograme. Sukompiliavus parašytą išeities tekstą yra gaunamas vykdomasis failas. Pastarasis vėliau vykdomas DirectX bibliotekų bei grafinio įrenginio pagalba. Kiekvienas toks metodas yra vykdomas kiekvienai paleistai grafinio procesoriaus gijai.

Gijos yra leidžiamos grupėmis. Gijos šiose grupėse gali būti sinchronizuojamos aprašant sinchronizavimo taškus (angl. *barriers*) arba keistis informacija bendro panaudojimo (angl. *shared*) atmintyje. Bendras paleidžiamų gijų grupių kiekis per viena paleidimą negali viršyti 65535 grupes. Gijų kiekis grupėse negali viršyti 1024 gijas (leidžiant jas ant DirectX 11 aparatūrinės įrangos) arba 768 gijas (leidžiant jas ant DirectX 10 aparatūrinės įrangos). Priklausomai nuo aparatūrinės įrangos versijos taipogi skiriasi ir prieinamas bendros atminties kiekis: DirectX 11 atveju jis yra lygus 32 KB, o DirectX 10 atveju – tik 16 KB.

Gijų grupės tarpusavyje negali būti sinchronizuojamos, tačiau paleidus jas darbui rezultatas gražinamas vartotojui tik tada, kai jos visos baigia darbą. Čia rezultatu yra skaitoma atminties buferiuose esanti informacija. DirectCompute gali naudoti kelių tipų buferius:

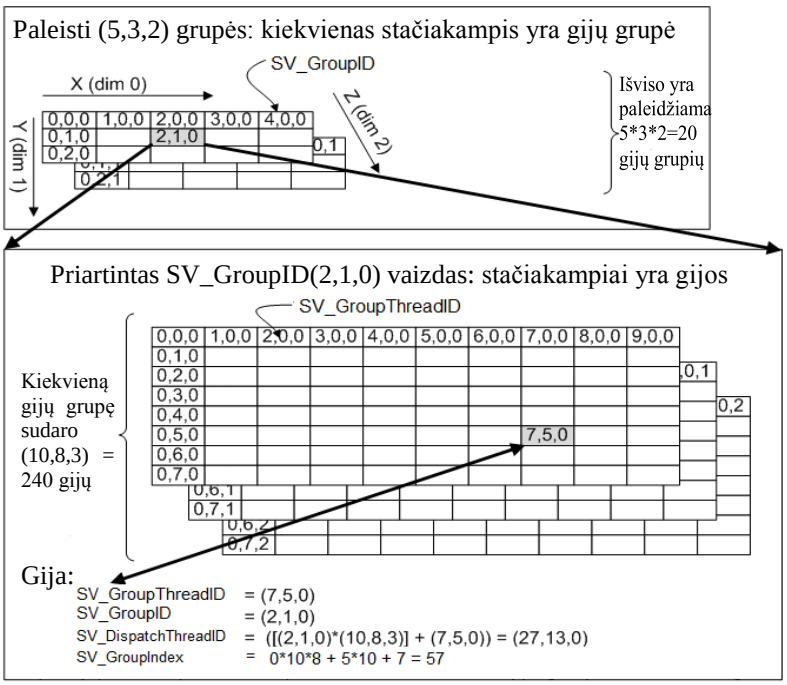
- Tik skaitymą (angl. *read-only*) palaikantys buferiai (paprasti, struktūriniai, vartojimo (angl. *consime*) ir kiti);
- Tik rašymą (angl. *write-only*) palaikantis pridėjimo (angl. *append*) buferis;

- Tiek rašymą, tiek skaitymą (angl *read-write*) palaikantys buferiai (paprasti, struktūriniai ir nerikiuojamos prieigos (angl. *unordered access*)).

Rašymą palaikantys atminties buferiai išlaiko savo informacija kol ant viršaus nėra užrašoma kita arba kol nėra pašalinama į tą vietą rodanti rodyklė. Tai yra svarbu, nes leidžia rašyti algoritmus, kurie savo darbą atlieka per kelis gijų grupių paleidimus. Skirtingai negu rašymą palaikantys buferiai, bendro panaudojimo gijų grupėms prieinama atmintis savo informacijos tarp kelių gijų grupių paleidimų neišlaiko. Tačiau bendro panaudojimo atmintis yra žymiai greitesnė, nes ji yra laikoma aparatūriniame keše.

Siekiant supaprastinti gijų valdymą, jų panaudojimą algoritmų sprendimuose bei suteikti vartotojui žinių apie tai, kuri gija tuo metų vykdo konkrečias instrukcijas, buvo realizuota gijų indeksavimo sistema [11]. Gijos grupėse, kaip ir pačios grupės, yra leidžiamos su trim indeksais: *X*, *Y* ir *Z*. Be to kiekvienai gijai dar yra prieinama eilė kintamųjų su semantikom *SV_GroupIndex*, *SV_DispatchThreadID*, *SV_GroupThreadID* ir *SV_GroupID*. Čia:

- *SV_GroupIndex* – gijų grupėje unikalus suplotas gijos numeris
- *SV_DispatchThreadID* – gijos numeris viso gijų grupių paleidimo atžvilgiu
- *SV_GroupThreadID* – gijos numeris gijų grupėje
- *SV_GroupID* – gijos grupės indeksas



7 pav. Gijų grupių ir gijų grupėse indeksavimo pavyzdys.
(paimta iš http://msdn.microsoft.com/en-us/library/ff476405%28v=vs.85%29.aspx)

7 paveiksle yra patektas šių indeksų bei gijų grupių paleidimo pavyzdys kai yra paleidžiamos gijų grupės su kreipiniu (5,3,2) ir gijos grupėse turi aprašą (10,8,3).

Remiantis grafinių procesorių gamintojų AMD ir NVIDIA rekomendacijomis, siekiant išgauti didžiausią DirectCompute našumą, reikia [6]:

- Pateikti pakankamai skaičiavimų darbo kad padengti duomenų perkėlimo iš CPU į GPU atminties zonas vėlinimus;
- Gijų grupės dydis privalo būti didesnis arba lygus aparatūriniam gijų procesorių bloko dydžiui (angl. *shader processor*), priešingu atveju aparatūriniai resursai yra naudojami neefektyviai. Šitas dydis priklauso nuo GPU architektūros;
- Stengtis kaip įmanoma labiau vienodai išdalinti darbą tarp gijų jų grupėse, nes tai leis efektyviau išnaudoti turimus resursus;
- Duomenų manipuliavimą surišti su skaliariniais (NVIDIA atveju) arba vektoriniais (AMD atveju) duomenų tipais. Tai leis pasiekti didesnį našumą darbe su duomenimis;
- Naudoti kuo mažiau atominių operacijų. Atominės operacijos sudaro gijų sinchronizacijos taškus;
- Mažinti kreipinių į atminties buferius ir bendros atminties zoną kiekius;
- Vengti bendros gijų atminties prieigų konfliktus. Prieigų kiekis yra lygus 32. Kiekvienas adresas, kuris yra per 32 DWORD vienetus nutolęs nuo kito naudoja tą patį prieigos tašką. Kai dvi ar daugiau gijų vienu metu naudoja tą patį prieigos tašką – atsiranda konfliktas, kuris mažina našumą.

3.3. Pasirinkto metodo pagrindimas

Tyrimui buvo pasirinktas Hierarchinis uždengtos geometrijos atrinkimo algoritmas. Šis sprendimas buvo padarytas remiantis šiais teiginiais:

1. Šio algoritmo dengiamumo sąlygos išskaičiavimo žingsnis yra lengvai lygiagretinamas, nes kiekvienas tikrinimas yra nepriklausomas nuo kitų.
2. Dengiamumo tikrinimui yra naudojamas aplinkos objektų gylio žemėlapis, kuris yra sukuriamas grafinio įrenginio pagalba.
3. Remiantis 1 ir 2 punktais galima teigti, kad šis algoritmas yra tinkamas vykdymui GPU pusėje, kas taipogi pašalins būtinybę perkelti aplinkos objektų gylio žemėlapio duomenis į CPU prieinamą atmintį.

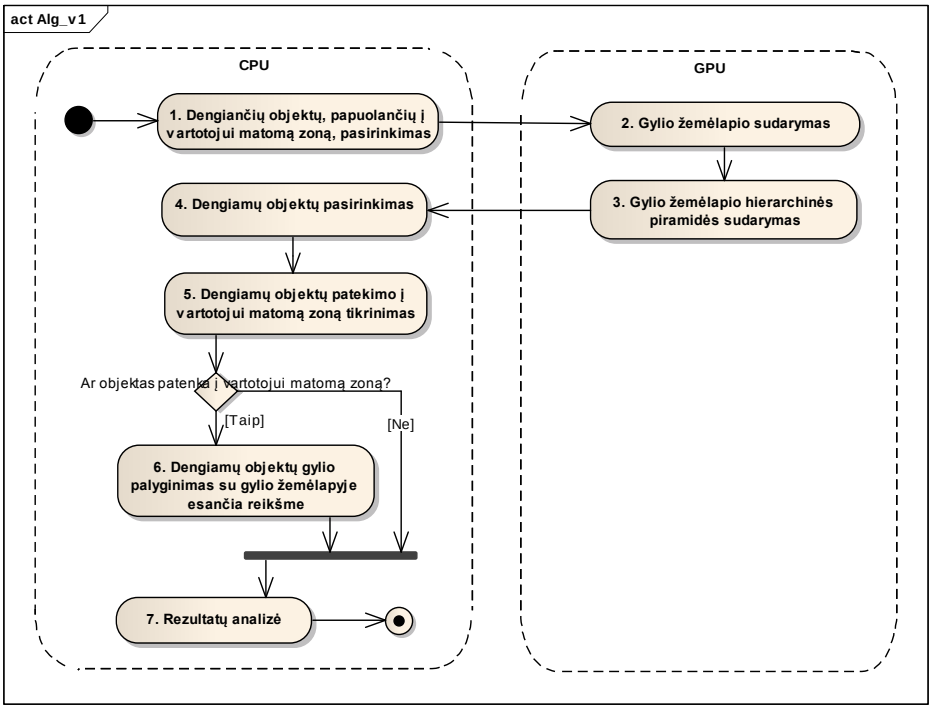
4. Šis algoritmas yra lengvai suprantamas ir jo realizacija nereikalauja didelių pastangų bei sudėtingų skaičiavimų.
5. Skaičiavimų perkėlimas į GPU pusę leis atlaisvinti CPU resursus kitiems skaičiavimams, vykdomiems šio algoritmo panaudojimo kontekste.

4. PROJEKTINĖ DALIS

Tiriamas algoritmas buvo realizuotas C# bei HLSL kalbose kaip magistro projektinio darbo grafinio variklio komponento dalis. Šiame skyriuje yra aptariamos realizuoto algoritmo bei siūlomų jo modifikacijų struktūra bei jo žingsnių realizacijos detalės.

4.1. Pradinio algoritmo struktūra

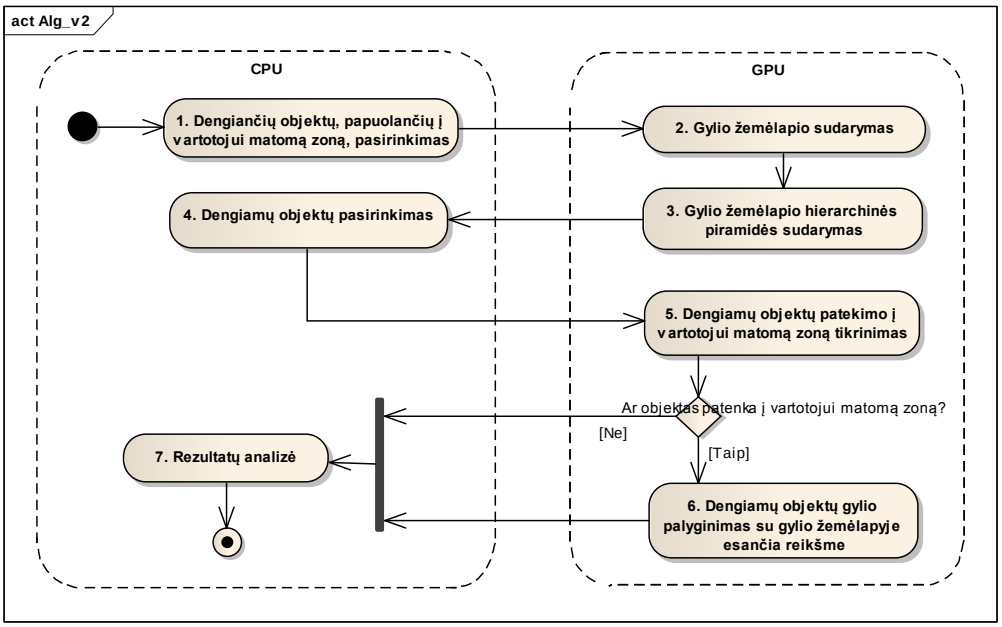
8 paveiksle yra pateiktas realizuoto pirminio algoritmo veiklos diagrama. Čia žingsniai yra atliekami remiantis 3.1.3.4 skyriuje pateikta metodologija. 4.2 skyriuje yra pateikiamos įvairių realizuoto algoritmo modifikacijų veiklos diagramos ir atliktų modifikacijų paaiškinimai.



8 pav. Pradinio algoritmo („v1“) veiksmų veiklos diagrama.

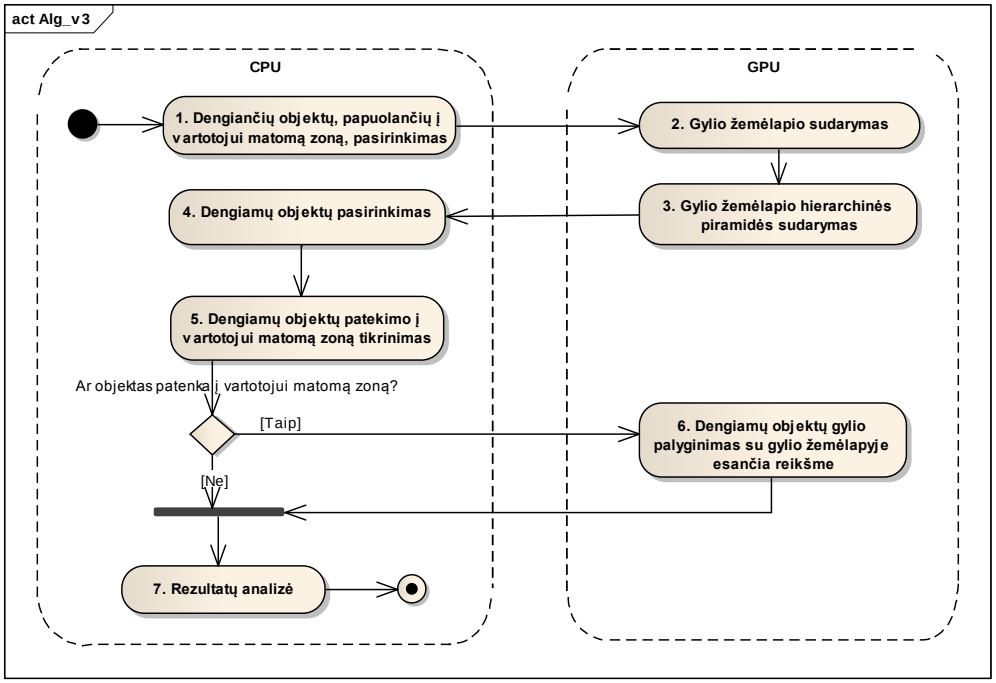
Šitas algoritmas didžiąją dalį savo veiksmų atlieka centrinio procesoriaus pagalba. Jam priskirsime pavadinimą „v1“ ir jis bus skaitomas kaip atsvaros taškas tyrimo rezultatų palyginimuose.

4.2. Algoritmo modifikacijų struktūra



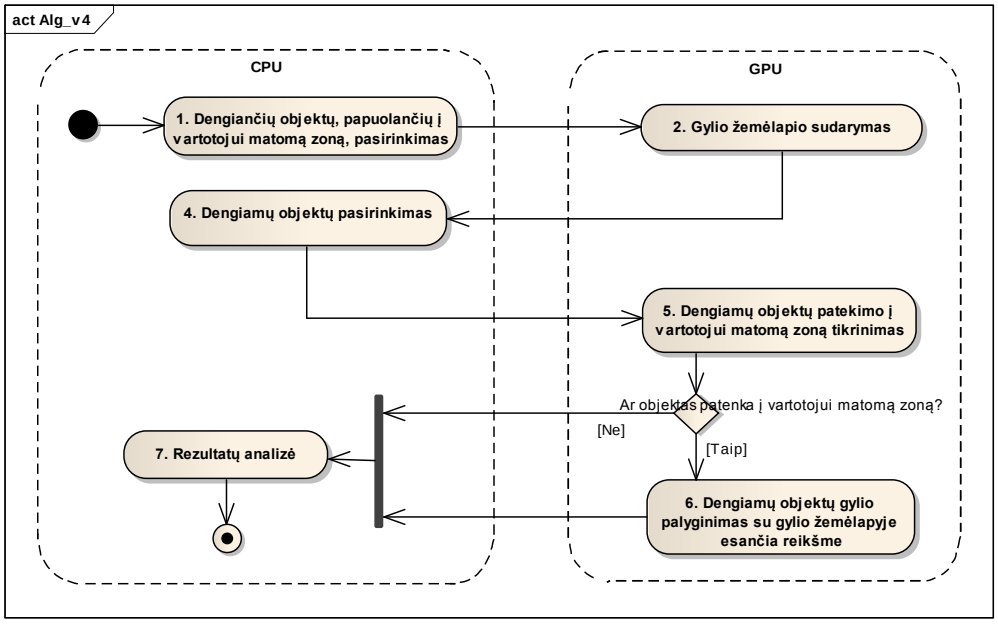
9 pav. Algoritmo modifikacijos („v2“) veiksmų veiklos diagrama.

Pirmoji algoritmo modifikacija, kuriai yra priskirtas pavadinimas „v2“, yra pateikta 9 paveiksle. Čia dengiamų objektų patekimo į vartotojui matomą zoną bei jų gylio apskaičiavimo ir palyginimo žingsniai yra atliekami GPU pusėje. Kiekvienam objektui yra skiriama atskira GPU gija.



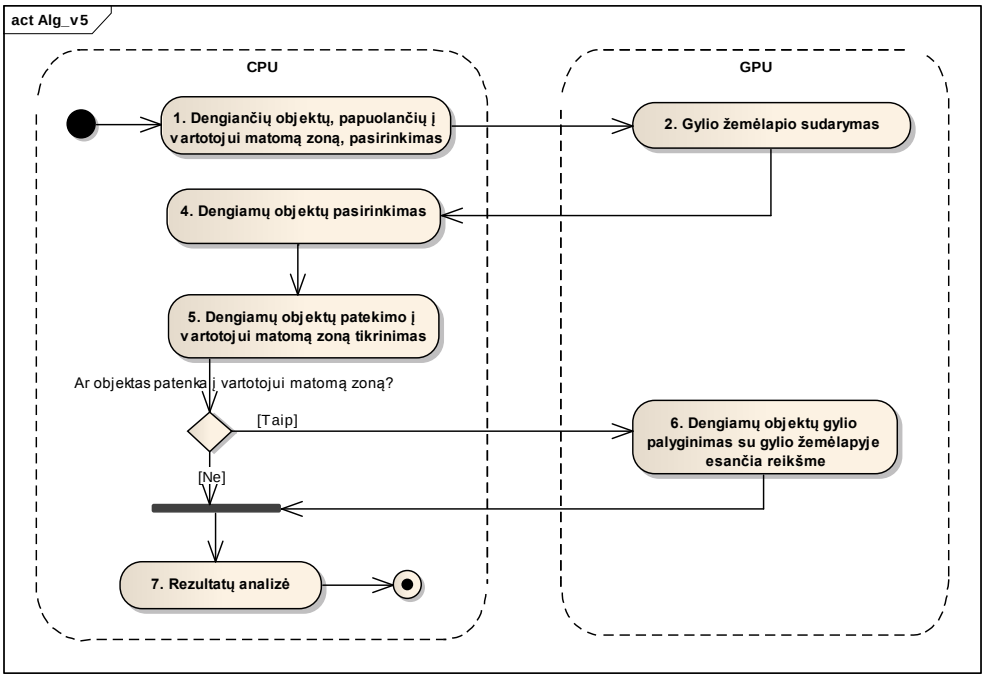
10 pav. Algoritmo modifikacijos („v3“) veiksmų veiklos diagrama.

Antroji algoritmo modifikacija, kuriai yra priskirtas pavadinimas „v3“, yra pateikta 10 paveiksle. Čia, lyginant su „v2“ modifikacija, dengiamų objektų patekimo į vartotojui matomą zoną patikrinimas yra atliekamas CPU pusėje. Taip yra sumažinamas leidžiamų GPU gijų kiekis.



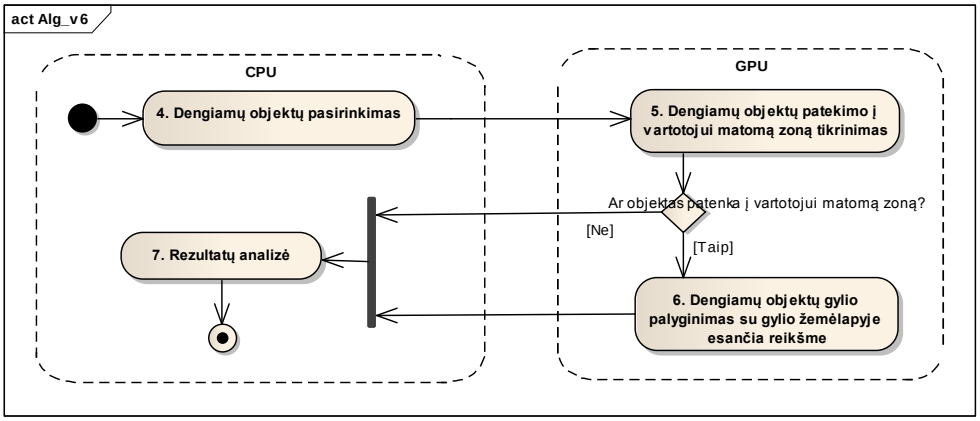
11 pav. Algoritmo modifikacijos („v4“) veiksmų veiklos diagrama.

Trečioji algoritmo modifikacija, kuriai yra priskirtas pavadinimas „v4“, yra pateikta 11 paveiksle. Čia, lyginant su „v2“ modifikacija, nėra konstruojama bei naudojama gylio žemėlapio hierarchinė piramidė.



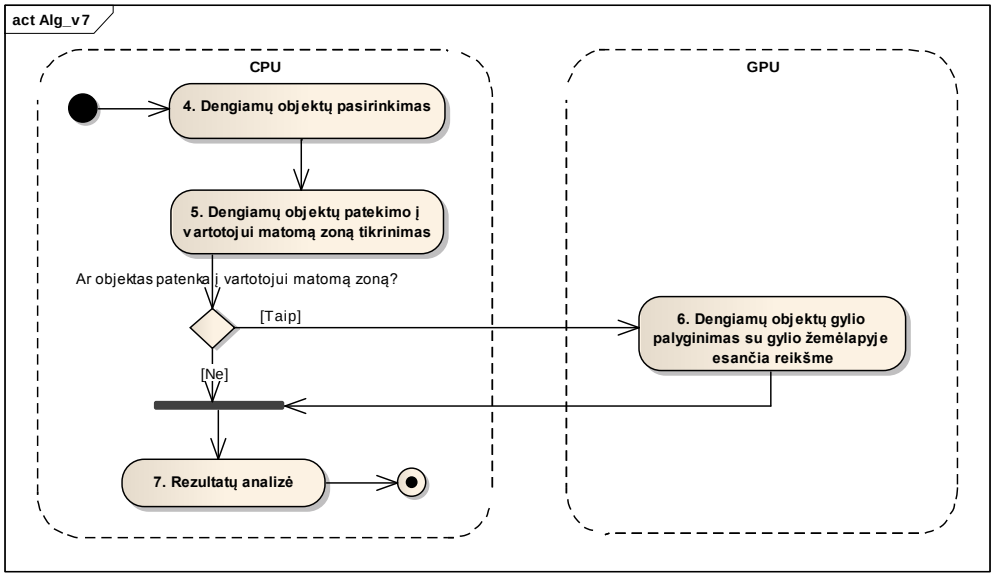
12 pav. Algoritmo modifikacijos („v5“) veiksmų veiklos diagrama.

Ketvirtoji algoritmo modifikacija, kuriai yra priskirtas pavadinimas „v5“, yra pateikta 12 paveiksle. Čia, lyginant su „v3“ modifikacija, nėra konstruojama bei naudojama gylio žemėlapio hierarchinė piramidė.



13 pav. Algoritmo modifikacijos („v6“) veiksmų veiklos diagrama.

Penktoji algoritmo modifikacija, kuriai yra priskirtas pavadinimas „v6“, yra pateikta 13 paveiksle. Čia, lyginant su „v2“ modifikacija, nėra konstruojamas atskiras gylio žemėlapis dengiantiems objektams. Vietoj jo yra naudojamas praeito kadro metu sukonstruotas vaizduojamos aplinkos gylio žemėlapis. Tokiu pačiu principu yra remiamasi ir šeštoje algoritmo modifikacijoje su pavadinimu „v7“, kuri yra pateikta 14 paveiksle.



14 pav. Algoritmo modifikacijos („v7“) veiksmų veiklos diagrama.

4.3. Algoritmo žingsnių realizacijos detalizavimas

4.3.1. Žingsnis „1. Dengiančių objektų, papuolančių į vartotojui matomą zoną, pasirinkimas“

Šiame žingsnyje yra atliekami šie veiksmai:

1. Dengiančių objektų pasirinkimas iš visų aplinkos objektų.
2. Pasirinktų objektų patekimo į vartotojui matomą zoną patikrinimas.

1 veiksmo tikslas yra atrinkti dengiamus objektus. Šie objektai yra nurodomi programuotojo, kuris tai padaro naudojantis kuriamos aplinkos informacija. 2 veiksmas yra atliekamas remiantis 3.1.1 skyriuje pateiktu metodu naudojant minimalias objektų dengiančias dėžės. Šios dengiančios dėžės, kaip ir visos algoritmo realizacijoje naudojamos dengiančios dėžės, yra lygiuotos pagal vaizduojamos aplinkos pasaulio koordinačių sistemos ašis (angl. *axis-aligned*).

4.3.2. Žingsnis „2. Gylio žemėlapio sudarymas“

Šiame žingsnyje GPU pusėje yra sukonstruojamas pasirinktų dengiančių objektų gylio žemėlapis. Gylis yra saugomas 32 bitų slenkančio kablelio formato buferyje (tekstūroje), kuriame reikšmės yra pateikiamos vartotojo kameros koordinačių sistemoje. Pagal nutylėjimą buferio dydis yra 256x128 pikseliai. Šis dydis buvo pasirinktas atlikus eilę bandymų siekiant išgauti optimalų buferio dydį atsižvelgus į darbo su buferiu našumą bei atliekamo tikrinimo tikslumą (3 ir 4 tyrimai).

Mes siūlome automatizuoti gylio žemėlapio sudarymą GPU pusėje tam panaudojant geometrijos paprogramę. Tokiu būdu yra galimas automatinis objekto minimalios dengiančios dėžės sukonstravimas remiantis paduotą minimalios objekto dengiančios dėžės informacija: jos *min* ir *max* kampų koordinatėmis.

Yra gaunama tokia vaizdavimo procedūros veiksmų seka:

1. Verteksų paprogramė (leidžiama kiekvienam dengiančiam objektui):
    a. Apskaičiuoja minimalios dengiančios dėžės *min* ir *max* kampų koordinates pasaulio koordinačių sistemoje. Jie yra perduodami geometrijos paprogramei.
2. Geometrijos paprogramė (leidžiama kiekvienam dengiančiam objektui):
    a. Paskaičiuoja bei išveda 36 verteksus remiantis minimalios dengiančios dėžės *min* ir *max* kampų koordinatėmis. Kiekvieni 3 vienas po kito išvesti verteksai yra traktuojami kaip naujo trikampio taškai. Verteksų koordinatės yra

išvedamos ekrano koordinačių sistemoje (angl. *screen space*). Su kiekvienu verteksų yra išvedama tos vietos gylio informacija vartotojo kameros koordinačių sistemoje. Šių trikampių padengiama erdvė vaizdavimo procedūroje yra paverčiama pikseliais ir paduodama pikselių paprogramei. Gylio informacija kiekvienam pikseliui yra gaunama atlikus tiesinį gylio reikšmių interpoliavimą (vykdomas automatiškai).

3. Pikselių paprogramė (leidžiama kiekvienam dengiančių objektų pikseliui):
    a. Išvedą tame pikselyje esančią gylio reikšmę į pajungtą tekstūrą. Prieš išvedimą reikšmė yra normalizuojama padalinus ją iš projektuojamo vaizdo maksimalios gylio reikšmės (angl. *z far*).

4.3.3. Žingsnis „3. Gylio žemėlapio hierarchinės piramidės sudarymas“

Šiame žingsnyje GPU pusėje yra sukonstruojama 2 žingsnyje gauto gylio žemėlapio hierarchinė piramidė. Ji yra daroma GPU pusėje 3.1.3.4 skyriuje aprašytu būdu.

4.3.4. Žingsnis „4. Dengiamų objektų pasirinkimas“

Šiame žingsnyje yra atrenkami objektai, kuriems bus atliekamas dengiamumo patikrinimas. Šiuos objektus sudaro visi vaizduojamos aplinkos objektai, išskyrus 1 žingsnyje atrinktus dengiančius objektus.

4.3.5. Žingsnis „5. Dengiamų objektų patekimo į vartotojui matomą zoną tikrinimas“

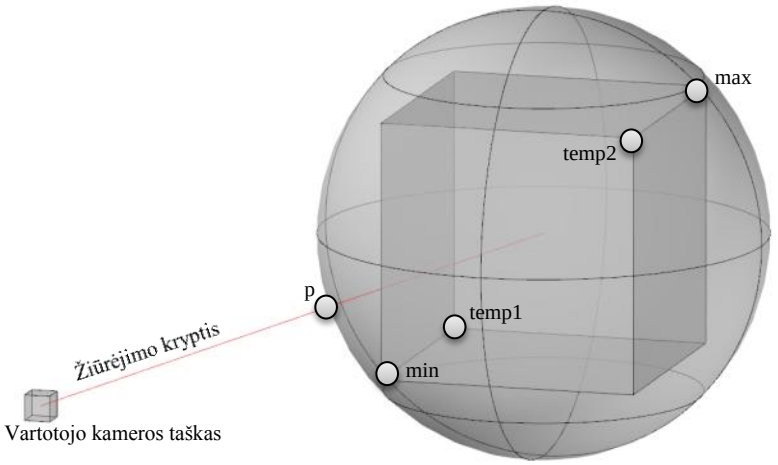
Šiame žingsnyje yra atliekamas objektų patekimo į vartotojui matomą zoną patikrinimas. Jis yra vykdomas 3.1.1 pateiktu metodu ir patikrinime naudoja objekto minimalią dengiančią sferą vietoj minimalios dengiančios dėžės. Taip buvo padaryta todėl, kad tokia tikrinimo operacija yra sudaryta iš mažesnio instrukcijų kiekio, nors ir tikrinimas nėra toks tikslus. Instrukcijų kiekis čia yra svarbus, nes šis žingsnis yra atliekamas ne tik CPU („v1“, „v3“, „v5“, „v7“ algoritmo versijos), bet ir GPU („v2“, „v4“, „v6“ algoritmo versijos) pusėje. Taipogi, kadangi dengiami objektai, priešingai negu dengiantys, dažniausiai yra pakankamai nedideli – sferos panaudojimas nesukels didelių susikirtimo su vartotojui matomą zoną paklaidų.

4.3.6. Žingsnis „6. Dengiamų objektų gylio palyginimas su gylio žemėlapyje esančia reikšme“

Šiame žingsnyje yra atliekamas objektų dengimo sąlygos patikrinimas, t.y. yra nustatoma ar objektas bus matomas vartotojui ar jis yra uždengtas su kita geometrija. Tikrinime yra naudojama dengiančių objektų gylio žemėlapio („v1-5“ algoritmo versijos) arba aplinkos gylio žemėlapio, sudaryto praeito kadro metu, („v6-7“ algoritmo versijos) informacija. Pastaruoju atveju aplinkos gylio žemėlapio dydis yra lygus atvaizduojamo paveikslo dydžiui.

Palyginimas yra pradedamas paimant 4 objekto minimalios dengiančios dėžės kampų koordinačių vietose esančias gylio reikšmes iš turimo gylio žemėlapio. Minimali dengianti dėžė yra aprašoma tik dviejų kampų (*min* ir *max*) koordinatėmis, tačiau čia yra išskaičiuojami dar du kampiniai taškai (*temp1 ir temp2*) siekiant pakelti atliekamo tikrinimo tikslumą. Yra randama bei išsaugoma šių gylio reikšmių maksimali reikšmė.

Gauta gylio maksimali reikšmė yra palyginama su apskaičiuotu arčiausiai vartotojo kameros taško ant objekto minimalios dengiančios sferos paviršiaus esančio taško *p* gylio reikšme. Jeigu šio ant sferos paviršiaus esančio taško gylio reikšmė yra didesnė negu gauta tikrinamo objekto maksimali gylio reikšmė – šis objektas bus uždengtas ir nematomas vartotojui. Remiantis 6 paveikslu šią sąlygą būtų galima perfrazuot šitaip: jeigu ant sferos paviršiaus esančio taško spalva yra tamsesnė už labiausiai sviesią 4 dengiančios dėžės kampų spalvą – šis objektas bus matomas vartotojui.



15 pav. Žingsnio logikos geometrinis paaiškinamasis vaizdas.

Šis žingsnis gražina objektų matomumą reprezentuojanti masyvą, kuriame objektui pagal jo indeksą priskirtoje vietoje yra įrašomas „0“ arba „1“. „0“ reiškia, jog objektas yra uždengtas

dengiančia geometrija ir neturi būti vaizduojamas. Atitinkamai „1“ atveju tarp vartotojo kameros taško ir objekto nebuvo rasta nieko, kas trukdytų jo matomumui.

4.3.7. Žingsnis „7. Rezultatų analizė“

Šiame žingsnyje yra analizuojami gauti rezultatai. Remiantis gražintomis reikšmėmis yra atžymima kuriuos objektus vaizduoti, o kuriuos ne.

5. TYRIMO IR EKSPEREMENTINĖ DALIS

5.1. Atliekamo tyrimo metodologija

Šiame darbe yra atliekami siūlomų algoritmo modifikacijų našumo tyrimai. Algoritmų našumas yra matuojamas nustatant jų pilną arba tik kai kurių žingsnių vykdymo laiką. Visi šiame darbe pateikti laiko matavimai yra išskaičiuojami kaip penkių to paties matavimo paleidimų vidurkis. Tyrimo metu minima matavimo užklausa – tai vienas duomenų rinkinys, kuriam yra norima gauti vieną rezultatą pasinaudojus vienu iš šiame darbe pateiktu algoritmu arba jo žingsniu. Visų matavimų metu yra išlaikoma ta pati matavimui naudojamos aparatūrinės bei programinės įrangos konfigūracija. Matavimai yra tiesiogiai priklausomi nuo matavimo įrangos, taigi jų atlikimas kitoje aplinkoje gali pateikti kitokius rezultatus ir (arba) privesti prie kitokių išvadų.

Šiame darbe yra nutarta vykdyti tokius tyrimus:

1. 2 žingsnio vykdymo laiko priklausomybė nuo dengiančių objektų kiekio ir vykdymo laiko palyginimas naudojant siūlomą automatinį dengiančios dėžės geometrijos kūrimą geometrijos paprogramės pagalba su statinės geometrijos atvaizdavimu.
2. Algoritmų, naudojančių GPU 5 bei 6 žingsnių vykdymui, gijų grupės dydžio įtaka našumui esant skirtingam užklausų kiekiui.
3. Gylio žemėlapio dydžio įtaka atliekamų dengiamumo sąlygos skaičiavimų tikslumui.
4. Algoritmų, naudojančių atskirą dengiančių objektų gylio žemėlapį („v1-5“ algoritmų versijos), gylio žemėlapio dydžio priklausomybė nuo vykdymo laiko esant skirtingam vykdomų užklausų kiekiui.
5. Visų algoritmų vykdymo laiko priklausomybė nuo vykdomų užklausų kiekio.
6. Vidutinis 5 ir 6 algoritmo žingsnių vykdymo skirtingų paskirties procesoriais (CPU ir GPU) metu suvartojamas CPU laikas esant skirtingam užklausų kiekiui.

5.2. Pasirinktų tyrimo metodų paskirtis

1 lentelė. Pasirinktų tyrimo metodų aprašas ir paskirtis

Tyrimo numeris	Paskirtis
1	Nustatyti siūlomo naujo dengiančios geometrijos gylio žemėlapio sudarymo metodo

Tyrimo numeris	Paskirtis
	vykdymo laiką esant skirtingam dengiančių objektų kiekiui ir palyginti jį su statiniu. Tai svarbu siekiant patikrinti siūlomo metodo veiksmingumą.
2	Nustatyti optimalų gijų grupės dydį esant skirtingam skaičiavimų užklausų kiekiui. Čia pakaks patikrinti šias algoritmo situacijas: 1. 5 ir 6 algoritmo žingsniai yra vykdomi GPU pusėje. 2. 6 algoritmo žingsnis yra vykdomas GPU pusėje. Šios situacijos yra būdingos kelioms algoritmo modifikacijoms ir jos sudaro visų galimų situacijų aibę, kurios algoritmai bus įtakojami gijų grupės dydžio.
3	Nustatyti optimalų gylio žemėlapio dydį remiantis atliekamų dengiamumo skaičiavimų metu gaunamu vizualiniu vaizdu ir jame matomų klaidingų rezultatų kiekiu. Klaidingu rezultatu yra skaitomas toks, kai objektui, kuris turi būti matomas vartotojui, yra išskaičiuojamas priešingas rezultatas. Šį rezultatą gausime vizualiai lygindami kelių gylio žemėlapio dydžiu panaudojimo metu gaunamus vaizdus.
4	Nustatyti gylio žemėlapio dydžio įtaką algoritmų našumui. Didesnis gylio žemėlapis padidina vykdomo tikrinimo tikslumą, tačiau jam reikia ilgesnės gylio žemėlapio piramidės (didesnės atminties sąnaudos). Čia pakaks patikrinti šias algoritmo situacijas: 1. CPU atlieka duomenų skaitymą iš gylio žemėlapio su hierarchine piramide. 2. GPU atlieka duomenų skaitymą iš gylio žemėlapio su hierarchine piramide. 3. GPU atlieka duomenų skaitymą iš gylio žemėlapio be hierarchinės piramidės. Šios situacijos yra būdingos kelioms algoritmo modifikacijoms ir jos sudaro visų galimų situacijų aibę, kurios algoritmai bus įtakojami gylio žemėlapio dydžiu.
5	Tarpusavyje palyginti siūlomo algoritmo modifikacijų našumą esant skirtingam vykdomų užklausų kiekiui.
6	Tarpusavyje palyginti 5 ir 6 žingsnių suvartojamo CPU laiko kiekį kai jie yra vykdomi CPU arba GPU pagalba. Čia pakaks patikrinti šias algoritmo situacijas: 1. 5 ir 6 žingsniai vykdomi CPU pusėje (atitikmuo – „v1“ algoritmas). 2. 5 žingsnis vykdomas CPU pusėje, 6 žingsnis – GPU pusėje (atitikmuo –

Tyrimo numeris	Paskirtis
	„v3“ algoritmas) 3. 5 ir 6 žingsniai vykdomi GPU pusėje (atitikmuo – „v2“ algoritmas). Šios situacijos sudaro visų galimų 5 ir 6 žingsnių paskirstymo tarp CPU ir GPU situacijų aibę.

5.3. Tyrimui naudojamos įrangos bei parametrų aprašas

Tyrimui yra naudojama tokia aparatūrinė bei programinė įranga:

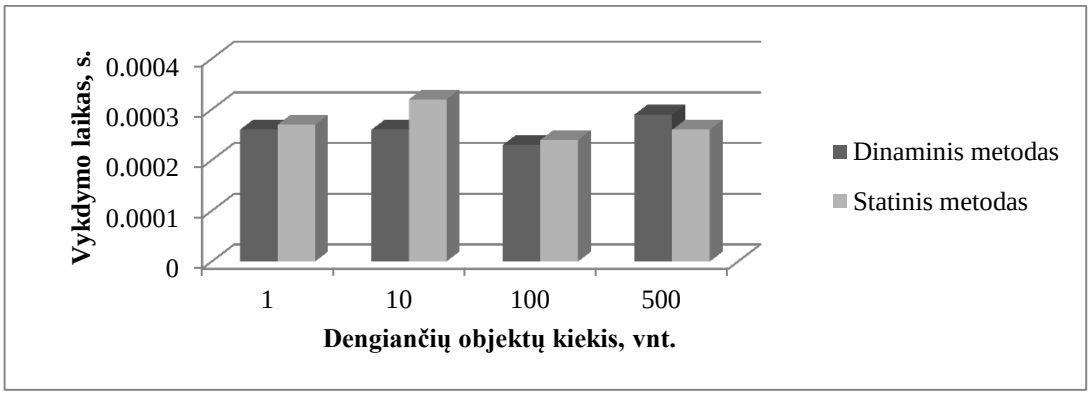
- Centrinis procesorius: Intel Core2 Duo E8400 @3.6 GHz.
- Operatyvioji atmintis: Kingston HyperX DDR2 4 GB @800 MHz.
- Grafinis procesorius: Gigabyte 450 GTS OC2.
- Grafinių tvarkyklių versija: 270.61
- Operacinė sistema: Microsoft Windows 7 64 bitų.

Visų tyrimų rezultatai yra tiesiogiai priklausomi nuo aukščiau aprašytos įrangos. Tai reiškia, jog kitos konfigūracijos atveju gali būti gauti šiek tiek kitokie arba net visai priešingi rezultatai.

Tyrimuose 3 – 6 gijų grupės dydis yra lygus 64. Tyrimuose 4 – 6 sudaromo gylio žemėlapio dydis yra lygus 256x128 pikseliams. 3 tyrime ekrano raškos dydis yra 1024x768 pikseliai. Tyrimuose 2, 4 – 6 dengiančių objektų kiekis yra lygus 10.

5.4. Tyrimo rezultatai

5.4.1. Tyrimas Nr. 1

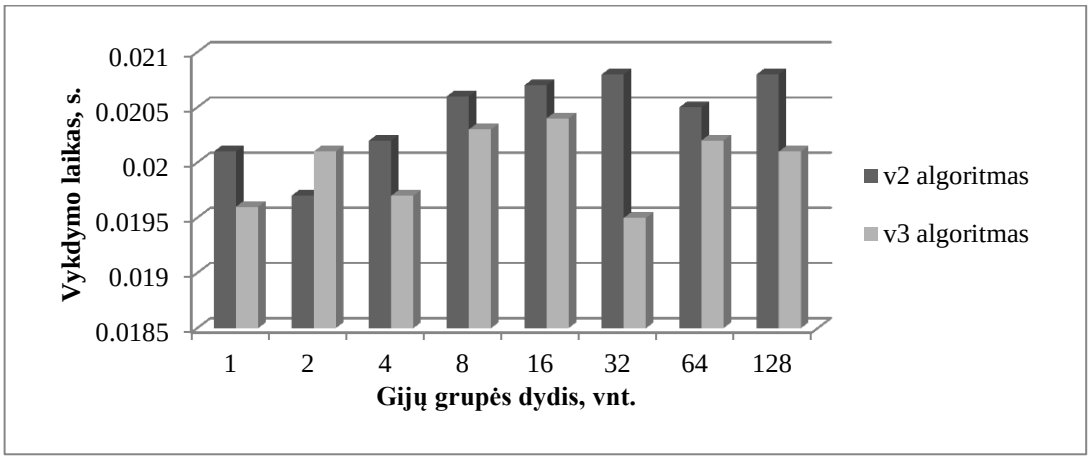


16 pav. Dengiančios geometrijos gylio žemėlapio sudarymo (2 žingsnis) vykdymo laiko priklausomybė nuo dengiančių objektų kiekio.

Iš 16 paveikslo matome, kad mūsų siūlomas dinaminis geometrijos paprograme paremtas gylio žemėlapio konstravimo metodo našumas yra apylygis statinės geometrijos vaizdavimu paremto metodo našumui. Remiantis šio tyrimo rezultatu galima teigti, jog dinaminio dengiančios geometrijos paviršiaus generavimo metodo panaudojimas yra pateisinamas situacijose kai yra norima sumažinti programinės įrangos atminties sąnaudas. Jos tampa mažesnės nes šio atveju atmintyje nereikia saugoti dengiančių dėžių paviršiaus geometrijos.

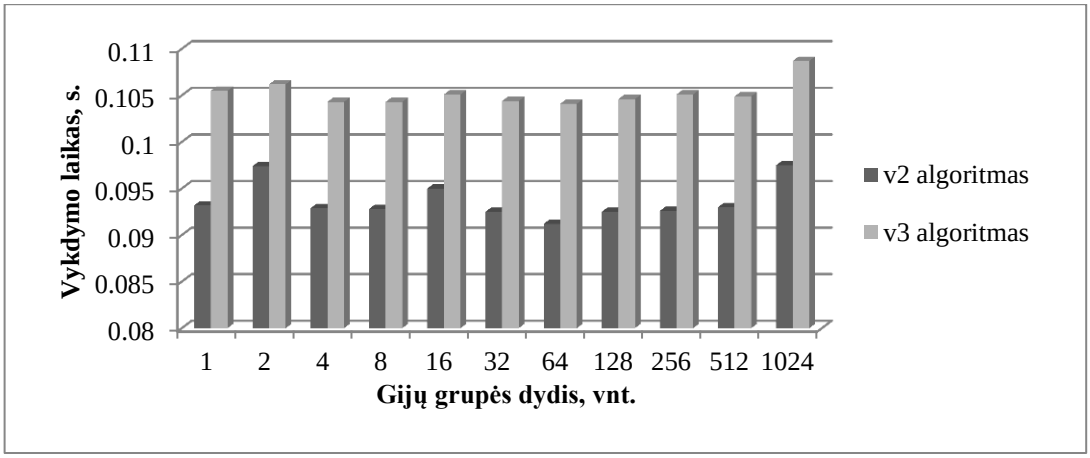
5.4.2. Tyrimas Nr. 2

Šiame tyrime mums užtenka patikrinti bet kokias dvi algoritmo versijas, kurios tenkina 1 lentelėje išsakytas antro tyrimo sąlygas. Jas atitinka „v2“ ir „v3“ algoritmo versijos. Tyrimas yra atliekamas su skirtingais užklausų kiekiais siekiant patikrinti įvairias situacijas.



17 pav. Algoritmų vykdymo laikas kintant gijų grupės dydžiui kai užklausų kiekis yra lygus 100.

Iš 17 paveikslo matome, jog gijų grupės dydis neturi ženklaus poveikio algoritmų našumui esant pakankamai nedideliam skaičiavimų kiekiui. Tačiau kai jis išauga (18 paveikslas) – didesnis našumas yra gaunamas su gijų grupės dydžiu lygiu 64. Remiantis šia informacija, 3 – 6 tyrimuose bus naudojamas būtent toks gijų grupės dydis.

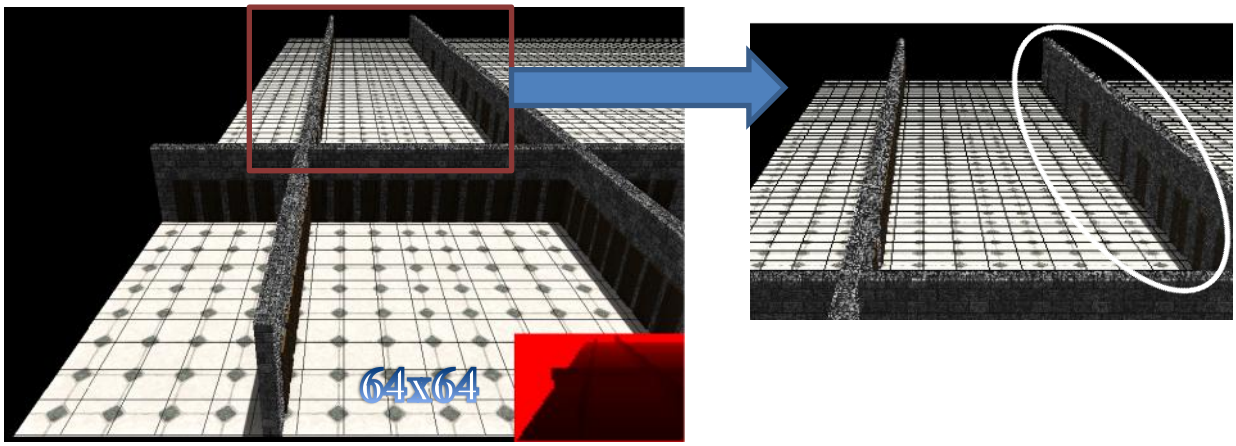


18 pav. Algoritmų vykdymo laikas kintant gijų grupės dydžiui kai užklausų kiekis yra lygus 100000.

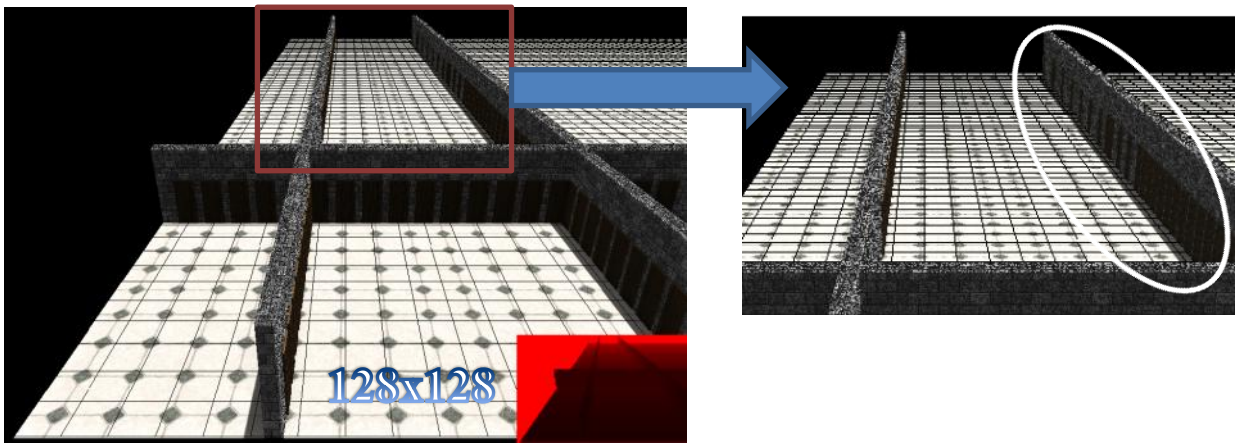
Šio tyrimo rezultatai taipogi parodo, kad 5 algoritmo žingsnio vykdymas GPU pusėje („v2“ algoritmas 18 paveiksle) duoda geresnį našumą esant dideliam užklausų kiekiui.

5.4.3. Tyrimas Nr. 3

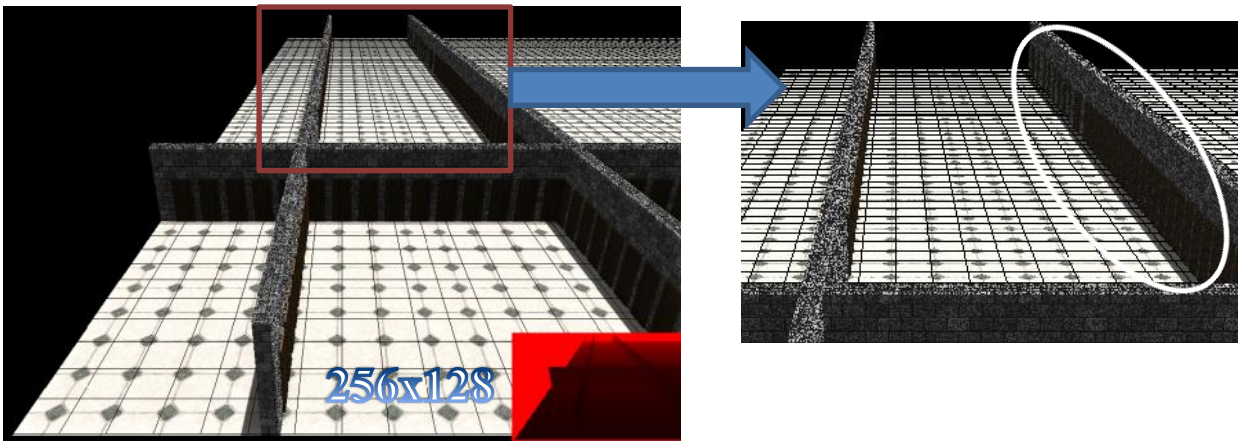
Šiame tyrime mes palyginsime vaizdus, gaunamus naudojant kelių dydžių dengiamų objektų gylio žemėlapius. Tyrimui panaudota aplinkos geometrija yra parinkta taip, kad išryškintų atliekamų skaičiavimų netikslumus. Netikslumai atsiranda vietose, kur gylio reikšmės artėjant prie 1, kas atitinka toliau nuo vartotojo kameros taško esančią geometriją. Jiems išryškinti buvo parinkti potencialiai dengiami objektai (šio atveju – durys), kurių geometrija artimai ribojasi su dengiančių objektų (šio atveju – sienų) geometrija. Tyrimo rezultatai yra pateikti 19 – 23 paveiksluose.



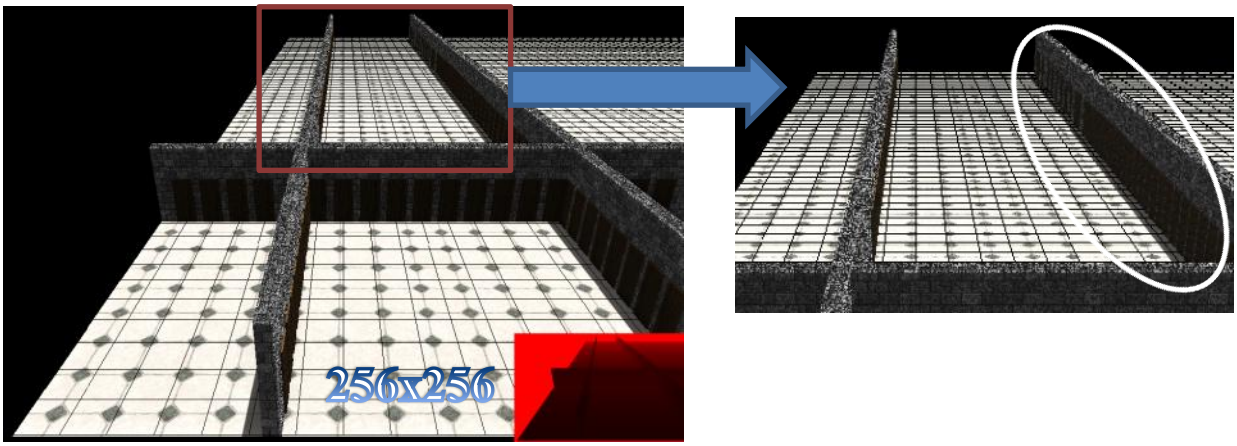
19 pav. Klaidingų rezultatų paieška kai gylio žemėlapio dydis yra lygus 64x64 pikseliams.



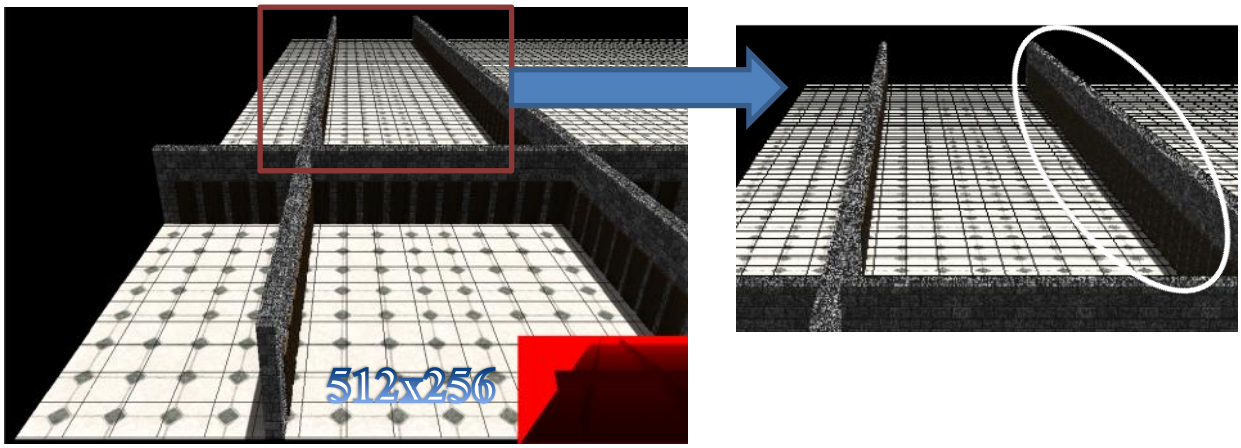
20 pav. Klaidingų rezultatų paieška kai gylio žemėlapio dydis yra lygus 128x128 pikseliams.



21 pav. Klaidingų rezultatų paieška kai gylio žemėlapio dydis yra lygus 256x128 pikseliams.



22 pav. Klaidingų rezultatų paieška kai gylio žemėlapio dydis yra lygus 256x256 pikseliams.



23 pav. Klaidingų rezultatų paieška kai gylio žemėlapio dydis yra lygus 512x256 pikseliams.

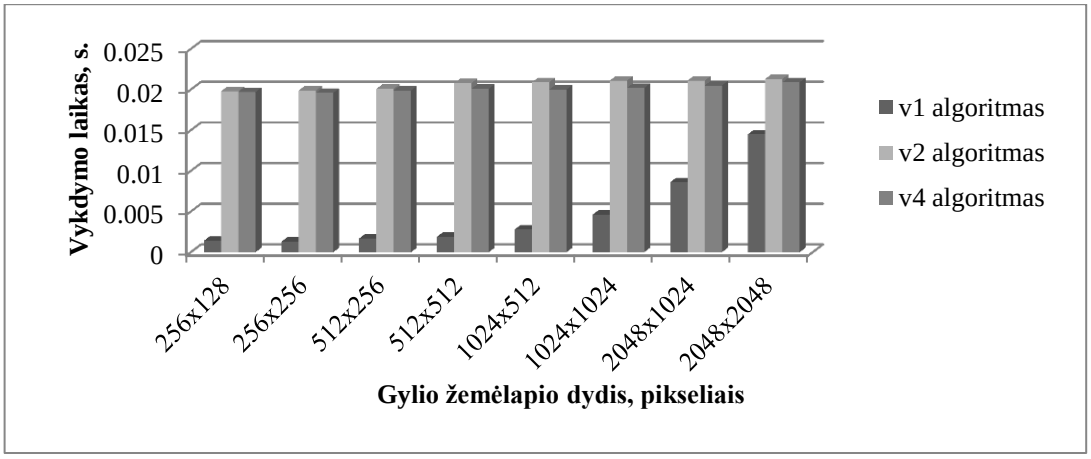
Kaip matome, mažėjant gylio žemėlapio dydžiui, klaidų kiekis (neatvaizduojamos durys) baltai pažymėtose paveikslų dešinėje esančiuose regionuose didėja. Kai gylio žemėlapio dydis yra lygus 512x256 pikseliams – klaidų visai nelieka. Kai gylio žemėlapio dydžiai lygus 256x128 ir 256x256 pikseliai – gaunamas rezultatas yra beveik identiškas.

Vertinant gautus rezultatus galima pabrėžti, jog idealaus paveikslo gavimas čia nėra būtinas, nes tolumoje esantys objektai yra sunkiai įžiūrimi vartotojui, tad jie mažai įtakoją bendrą gaunamą vaizdą. Remiantis tyrimo rezultatais bei čia išsakytais teiginiais galima padaryti išvadą, jog gylio žemėlapio dydis 256x128 pikselių yra pakankamas siekiant gauti ganėtinai tikslų vaizduojamos aplinkos vaizdą su tyrime naudota 1024x768 pikselių ekrano raiška.

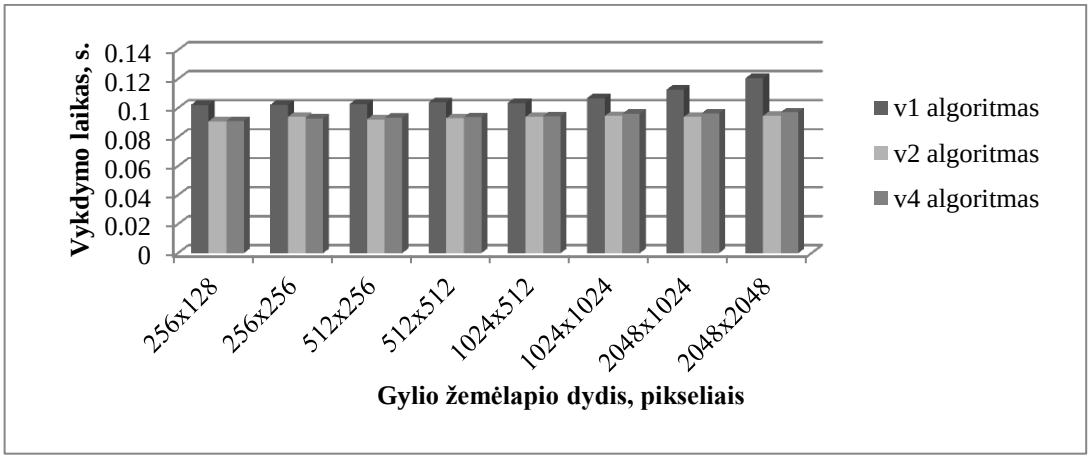
5.4.4. Tyrimas Nr. 4

Šiame tyrime mums užtenka patikrinti bet kokias tris algoritmo versijas, kurios tenkina 1 lentelėje išsakytas ketvirto tyrimo sąlygas. Jas atitinka „v1“, „v2“ ir „v4“ algoritmo versijos. Tyrimas yra atliekamas su keliais skirtingais gylio žemėlapio dydžiais siekiant patikrinti įvairias situacijas.

Šio tyrimo rezultatai, pateikti 24 ir 25 paveiksluose, parodo, jog gylio žemėlapio dydis neturi ypatingai didelės įtakos „v2“ ir „v4“ algoritmų našumui, tačiau mažesnis jo dydis visgi suteikia nedidelį našumo prieaugį. „v1“ algoritmo atveju skirtumas yra labiau matomas: augant gylio žemėlapio dydžiui jis ženkliai įtakoja algoritmo našumą. Tai ypač pastebima kai užklausų kiekis nėra didelis (24 paveikslas). Šią įtaką paaiškina tai, jog „v1“ algoritmo vykdymo metu gylio žemėlapį tenka perkelti iš grafinės į operatyviąją atmintį, o šios operacijos laikas tiesiogiai priklauso nuo perkeliamų duomenų dydžio. Dar vienas svarbus faktorius – mažesnis gylio žemėlapis taupo grafinę atmintį.



24 pav. Algoritmų vykdymo laikas kintant gylio žemėlapio dydžiui kai užklausų kiekis yra lygus 100.

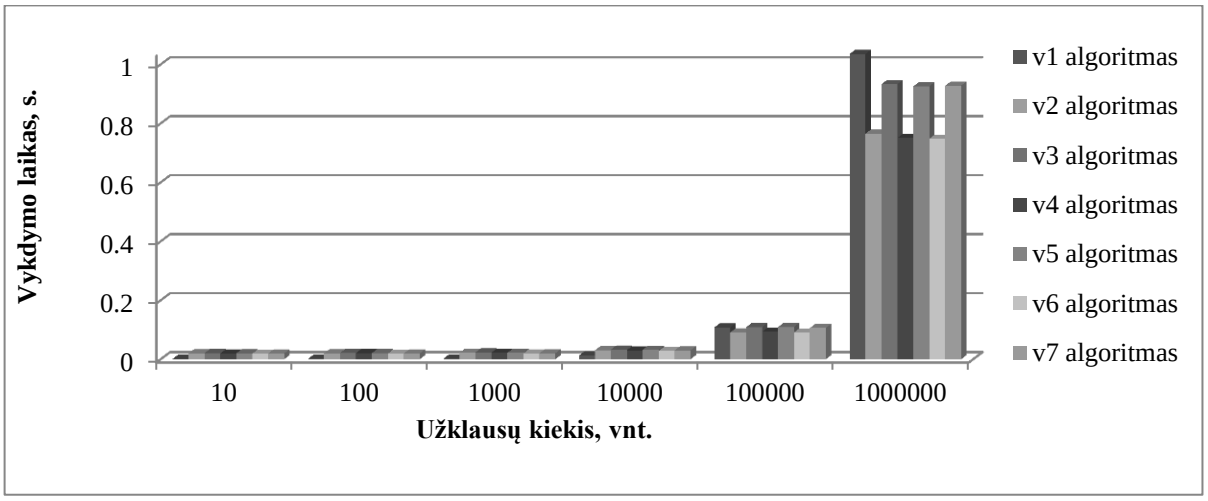


25 pav. Algoritmų vykdymo laikas kintant gylio žemėlapio dydžiui kai užklausų kiekis yra lygus 100000.

Gylio žemėlapio piramidės panaudojimas „v2“ algoritme lyginant su jos nebuvimu „v4“ algoritme neigiamai įtakoja našumą esant mažam skaičiavimų kiekiui (24 paveikslas). Čia jo sudarymo kaina neatperka suteikiamos naudos. Tačiau didėjant vykdomų užklausų kiekiui, ypač kai gylio žemėlapio dydis yra nemažas (didesnis negu 256x256 pikseliai, 25 paveikslas), jo panaudojimas atperka didelio gylio žemėlapio panaudojimo kainą GPU pusėje.

Remiantis šiais išsakytais teiginiais galima padaryti išvadą, jog sudaromo gylio žemėlapio dydį reikia stengtis laikyti kaip įmanoma mažesniu. Tokiu atveju gylio žemėlapio piramidės sudarymas ir panaudojimas tampa nebeaktualus.

5.4.5. Tyrimas Nr. 5



26 pav. Algoritmų vykdymo laikas esant skirtingam užklausų kiekiui.

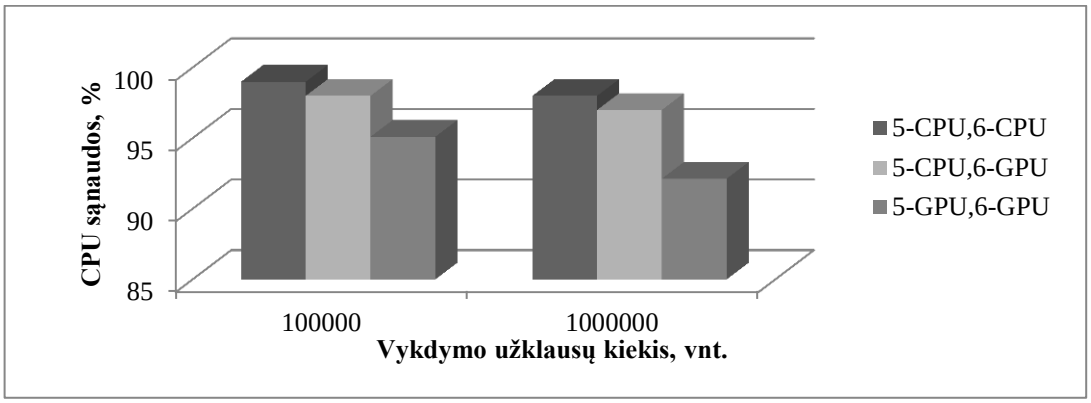
Analizuojant 26 paveiksle pateiktus šio tyrimo rezultatus galima išskirti šiuos svarbius momentus:

- Dengiamų objektų papuolimo į vartotojui matomą zoną (5 algoritmo žingsnis) tikrinimo atlikimas CPU pusėje tuo metu kai dengiamumo tikrinimas yra atliekamas GPU pusėje neduoda jokios naudos, ir netgi priešingai – neigiamai atsiliepia bendram algoritmų našumui („v3“,“ v5“ ir „v7“ algoritmo atvejai) kai užklausų kiekis yra didesnis arba lygus 10000. Nors šis atvejis ir mažina paleidžiamų GPU gijų kiekį, tačiau, kaip matome, GPU šį tikrinimą gali atlikti daug greičiau už CPU ir net didesnis gijų kiekis nedaro didelės įtakos GPU pusėje vykdomų algoritmų našumui.
- Didžiausią našumą iš GPU algoritmų demonstruoja „v6“ algoritmas. Dengiančių objektų gylio žemėlapio sudarymui (1 – 3 algoritmų žingsniai) skiriamas laikas neatstoja šio žemėlapio panaudojimu suteikiamo našumo prieaugio. Remiantis šiais rezultatais galime teigti, jog šių žingsnių galima atsisakyti ir naudoti praeito vaizdavimo kadro metu sugeneruotą aplinkos gylio žemėlapį. „v6“ algoritmas, lyginant su „v1“ algoritmu, parodo 19% našumo prieaugį kai užklausų kiekis yra lygus 100000 ir 38% našumo prieaugį kai užklausų kiekis yra lygus 1000000.
- Palyginus „v2“ ir „v4“ algoritmų našumą galima teigti, jog gylio žemėlapio piramidės sudarymas bei panaudojimas neatneša jokio ženklaus našumo prieaugio su naudotu 256x128 pikselių dydžio gylio žemėlapių. Priešingai, kai kuriais atvejais jos sudarymas net neigiamai įtakoja visą algoritmo našumą.

- Pirminis CPU pusėje veikiantis „v1“ algoritmas parodo gerus našumo rezultatus esant 10 – 10000 užklausų kiekiui. Be abejo svarbų vaidmenį čia vaidina tai, jog 5 ir 6 algoritmų žingsniuose yra pritaikytas lygiagretaus skaičiavimų vykdymo principas. Remiantis šia metodiką galima teigti, jog procesoriaus su didesnių branduolių kiekiu panaudojimas leistų šiai algoritmo versijai sumažinti atotrūkį iki GPU algoritmo versijų kai užklausų kiekis yra tarp 100000 ir 1000000 vienetų.

5.4.6. Tyrimas Nr. 6

Šiame tyrime yra matuojamas tik 5 ir 6 žingsnių vykdymo laikas kai jie yra vykdomi skirtingos paskirties procesoriais. Siekiant patikrinti įvairias situacijas buvo pasirinkti keli vykdymo užklausų kiekiai.



27 pav. Vidutinės CPU laiko sąnaudos algoritmų 5 ir 6 žingsnių vykdymo metu.

Kaip matome 27 paveiksle, 5 ir 6 algoritmų žingsnių vykdymas GPU pagalba leidžia sutaupyti CPU resursų. Nors nėra sutaupoma tiek ir daug (~6.37% kai užklausų kiekis lygus 1000000), tai vis tiek yra svarbu, nes GPU pajėgumas auga daug greičiau negu CPU. Tad yra tikslinga stengtis nukelti kaip įmanoma daugiau darbo į GPU pusę kai tik tai yra įmanoma ir palikti CPU resursus tiems skaičiavimams, kurių vykdymas GPU pusėje yra neįmanomas arba nėra prasmingas.

6. IŠVADOS

1. Dinaminio dengiančių objektų geometrijos formavimo geometrijos paprogramės pagalba našumo tyrimas (tyrimas Nr. 1, 34 psl.) parodė, kad mūsų siūlomo metodo našumas yra apylygis statiniam.
2. Gijų grupės dydžio įtakos algoritmų vykdymo laikui tyrimas (tyrimas Nr. 2, 35 psl.) parodė, kad esant mažam skaičiavimų kiekiui gijų grupės dydis neturi didelės įtakos realizuotų algoritmų našumui. Skaičiavimų kiekio didinimas pakeitė situaciją: esant dideliam užklausų kiekiui didžiausią našumą su mūsų algoritmais galimą pasiekti pasirinkus gijų grupės dydį lygu 64.
3. Gylio žemėlapio dydžio įtakos klaidingų rezultatų kiekiui tyrimas (tyrimas Nr. 3, 36 psl.) parodė, kad kai ekrano raiška yra lygi 1024x768 pikseliams, pakanka naudoti 256x128 pikselių dydžio gylio žemėlapį. Tokio gylio žemėlapio panaudojimas duoda pakankamai mažą klaidingų rezultatų kiekį kuris menkai įtakos vaizduojamo aplinkos vaizdo kokybę ir pakels algoritmų našumą (tyrimas Nr. 4, 38 psl.).
4. Gylio žemėlapio dydžio įtakos algoritmų našumui tyrimas (tyrimas Nr. 4, 38 psl.) parodė, jog gylio žemėlapio dydis neturi ypatingai didelės įtakos jį GPU pusėje naudojančių algoritmų našumui. Naudojant jį CPU pusėje esant nedideliam užklausų kiekiui situacija yra priešinga. Visumoje esant mažesniam gylio žemėlapio dydžiui yra gaunamas geresnis algoritmų našumas.
5. Gylio žemėlapio piramidės suteikiamos naudos esant skirtingiems gylio žemėlapio dydžiams tyrimas (tyrimas Nr .4, 38 psl.) parodė, kad kai gylio žemėlapio dydis yra lygus 256x128 pikseliams, gylio žemėlapio piramidės panaudojimas neturi jokios įtakos ištirtų algoritmų našumui. Šio atveju šito žingsnio galima atsisakyti. Gylio žemėlapio piramidės panaudojimas yra aktualus esant dideliam gylio žemėlapio dydžiui.
6. Objektų patekimo į vartotojui matomą zoną vykdymo (5 algoritmo žingsnis) CPU ir GPU pusėse palyginimo tyrimas (tyrimas Nr. 5, 40 psl.) parodė, jog šio žingsnio vykdymas CPU pusėje neigiamai įtakoją tirtų algoritmų našumą esant 10000 arba didesniam užklausų kiekiui tuo metu kai dengiamumo sąlygos tikrinimas (6 algoritmo žingsnis) yra atliekamas GPU pusėje.

7. CPU ir GPU algoritmų vykdymo laikų palyginimo tyrimas (tyrimas Nr. 5, 40 psl.) parodė, jog esant mažam (10 – 10000) vykdomų užklausų kiekiui šio uždavinio sprendimui geriau naudoti CPU, tačiau kai užklausų kiekis pasidaro didesnis – GPU panaudojimas atneša ženklų našumo prieaugį (19% kai užklausų kiekis lygus 100000 ir 38% kai užklausų kiekis lygus 1000000).
8. CPU ir GPU algoritmų vykdymo laikų palyginimo tyrimas (tyrimas Nr. 5, 40 psl.) parodė, jog gylio žemėlapio piramidės panaudojimas neatneša jokio ženklaus našumo prieaugio su tyrimui naudota įranga. Taipogi gylio žemėlapio, sugeneruoto praeito aplinkos vaizdavimo kadro metu, panaudojimas vietoje specialiai kuriamo duoda didžiausią našumą.
9. Objektų patikimo į vartotojui matomą zoną (5 algoritmo žingsnis) ir objektų dengiamumo sąlygos išskaičiavimo (6 algoritmo žingsnis) žingsnių vykdymo CPU bei GPU pagalba metu suvartojamo CPU laiko tyrimas (tyrimas Nr. 6, 41 psl.) parodė, jog šių žingsnių vykdymas GPU pagalba leidžia sumažinti algoritmo suvartojamą CPU laiką esant dideliam užklausų kiekiui.

7. PADĖKOS

Norime išreikšti padėkas:

- N. Darnell už savo tiklaraštyje, esančiame adresu http://www.nickdarnell.com/?p=942, pateikiamas mintis apie Uždengtos geometrijos atrinkimo algoritmo realizavimą naudojant DirectCompute bei DirectX 11 technologijas ir ten pateikiamus išeities teksto pavyzdžius.
- Kauno Technologijos Universiteto Programų Sistemų Inžinerijos katedros dr. Tomui Blažauskui už konsultacijas bei pagalba ruošiant šį darbą.

8. LITERATŪROS SARAŠAS

[1] **AMD**. AMD App Acceleration. *Advanced Micro Devices, Inc.* 2011 [žiūrėta 2011.04.20]. Prieiga per internetą: http://www.amd.com/us/products/technologies/amd-app/Pages/eyespeed.aspx

[2] **Boyd C**. DirectCompute: Capturing the Teraflop. *PDC09, Microsoft Corporation*. 2009 [žiūrėta 2011.04.19]. Prieiga per internetą: http://ecn.channel9.msdn.com/o9/pdc09/ppt/CL03.pptx

[3] **Barczak J., Oat C., Shopf J., Tatarchuk N.** March of the Froblins: Simulation and Rendering Massive Crowds of Intelligent and Detailed Creatures on GPU. Chaper 3.3: Character LOD Management. *Game Computing Applications Group, AMD Inc., SIGGRAPH 2008.* 2008 [žiūrėta 2011.04.21]. Prieiga per internetą: http://developer.amd.com/documentation/presentations/legacy/Chapter03-SBOT-March_of_The_Froblins.pdf.

[4] **Bittner J., Wimmer M.** GPU Gems 2. Part I – Geometric Complexity. Chapter 6: Hardware Occlusion Queries Made Useful *NVIDIA Corporation*. 2005 [žiūrėta 2011.04.21]. Prieiga per internetą: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter06.html

[5] **Bourke P.** Frustum Culling. *Bourke P.* November 2000 [žiūrėta 2011.04.22]. Prieiga per internetą: http://paulbourke.net/miscellaneous/frustum/

[6] **Cebenoyan C., Thibieroz N.** DirectCompute Performance on DX11 Hardware. *GDC 2010, NVIDIA, AMD.* 2010 [žiūrėta 2011.04.23]. Prieiga per internetą: http://developer.amd.com/gpu_assets/DirectCompute%20Performance.zip

[7] **Hill S.** Rendering with Conviction. *Ubisoft Montreal, GDC 2010.* 2010 [žiūrėta 2011.04.25]. Prieiga per internetą: http://www.selfshadow.com/talks/rwc_gdc2010_v1.pdf.p1-19

[8] **KHRONOS**. OpenCL Overview. *KHRONOS Group.* 2011 [žiūrėta 2011.04.23]. Prieiga per internetą: http://www.khronos.org/opencl/

[9] **Laakso M.** Potentially Visible Set (PVS).*Helsinki University of Technology.* 2003 [žiūrėta 2011.04.26]. Prieiga per internetą: http://www.tml.tkk.fi/Opinnot/Tik-111.500/2003/paperit/MikkoLaakso.pdf. p3-4, 6-8

[10] **MSDN**. HLSL. *Microsoft Corporation.* 2011 [žiūrėta 2011.04.10]. Prieiga per internetą: http://msdn.microsoft.com/en-us/library/bb509561%28v=vs.85%29.aspx

[11] **MSDN.** ID3D11DEVICECONTEXT::DISPATCH METHOD. *Microsoft Corporation.* 2011 [žiūrėta 2011.04.11]. Prieiga per internetą: http://msdn.microsoft.com/en-us/library/ff476405%28v=vs.85%29.aspx

[12] **MSDN.** Queries (DirectX 9). *Microsoft Corporation.* 2011 [žiūrėta 2011.04.10]. Prieiga per internetą: http://msdn.microsoft.com/en-us/library/bb147308%28v=vs.85%29.aspx

[13] **MSDN.** Resources (DirectX 10). *Microsoft Corporation.*2011 [žiūrėta 2011.04.05]. Prieiga per internetą: http://msdn.microsoft.com/en-us/library/bb205127%28v=vs.85%29.aspx

[14] **NVIDIA**. What is CUDA?. *NVIDIA Corporation.* 2011 [žiūrėta 2011.04.15]. Prieiga per internetą: http://www.nvidia.com/object/what_is_cuda_new.html

[15] **Piese B. H.** An Algorithm for Hidden Surface Complexity Reduction and Collision Detection Based on Oct Trees. *GDC 2000*. 2000 [žiūrėta 2011.03.03]. Prieiga per internetą: http://www.brentpease.com/GDCpaper1.html

[16] **Persson E.** Depth In-depth. *AMD.* 2007 [žiūrėta 2011.04.02]. Prieiga per internetą: http://developer.amd.com/media/gpu_assets/Depth_in-depth.pdf. d1-2

[17] **Pipenbrinck N.** Backface Culling in Object Space. *Pipenbrinck N.* June 1997 [žiūrėta 2011.04.03]. Prieiga per internetą: http://www.cubic.org/docs/backcull.htm

[18] **Ramires A.** View Frustum Culling Tutorial: Geometric Approach: Testing Points and spheres. *Lighthouse 3D.* 2011[žiūrėta 2011.04.02]. Prieiga per internetą: http://www.lighthouse3d.com/opengl/viewfrustum/index.php?gatest

[19] **Skadron K**. Thrends in Multicore Architecture. *ASPLOS '08*. 2008 [žiūrėta 2011.04.07]. Prieiga per internetą: http://gpgpu.org/static/asplos2008/ASPLOS08-2-manycore.pdf

[20] **Zhang H.** Effective Occlusion Culling for the Interactive Display of Arbitrary Models. *Department of Computer Science, UNC-Chapel Hill*. 1998 [žiūrėta 2011.04.08]. Prieiga per internetą: http://www.cs.unc.edu/~zhangh/dissertation.pdf

9. PRIEDAI

9.1. Publikuotas straipsnis

Šis straipsnis buvo publikuotas leidinyje „Informacinės technologijos. XVI tarpuniversitetinė magistrantų ir doktorantų konferencija“.

Skaičiavimų spartinimas panaudojus grafinį procesorių DirectCompute technologijos pagalba

Sergejus Topolovas[1], **Algis Pavasaris**[2]

Kauno Technologijos Universitetas, Programų inžinerijos katedra, Studentų g. 50-406, Kaunas, Lietuva

[1]*topolovas@gmail.com*, [2]*algis.pavasaris@meganet.lt*

Santrauka (abstract). Šiame straipsnyje nagrinėjamas grafinio procesoriaus panaudojimas bendro pobūdžio skaičiavimų atlikimui, tiriamas įvairių faktorių poveikis jų vykdymo našumui. Šiems faktoriams įvertinti buvo atlikti tyrimai panaudojant DirectCompute technologiją, bei du skirtingus algoritmus: A* kelio paieška ir Hierarchinis uždengtos geometrijos atrinkimas. Aptariami tyrimų rezultatai.

Raktiniai žodžiai: skaičiavimų spartinimas, grafinis procesorius, DirectCompute, lygiagretus skaičiavimai, trumpiausio kelio paieška, hierarchinis uždengtos geometrijos atrinkimas.

9.1.1. Įžanga

Kiekvienais metais augantis grafinių procesorių (angl. *Graphic Processing Unit*; toliau – GPU) pajėgumas verčia programinės įrangos kūrėjus bandyti išnaudoti šį potencialą, GPU pagalba atliekant ne vien su grafiniu apdorojimu susijusius skaičiavimus [7]. Šiuolaikiniai grafiniai procesoriai gali vienu metu vykdyti šimtus lygiagrečių veiksmų taip pranokdami esamas centrinių procesorių (angl. *Central Processing Unit*; toliau – CPU) galimybes. Skaičiavimų atlikimas GPU pagalba yra trumpai vadinamas GPGPU (angl. *General-Purpose computing on Graphical Processing Units*).

GPU panaudojimas bendro pobūdžio skaičiavimams nėra nauja idėja, tačiau tik nesenai ši idėja pradėta plačiau taikyti. Tam turėjo įtakos atsiradusi eilė programinių sąsajų (angl. *API*) ir technologijų, skirtų šiai procedūrai supaprastinti, tokių kaip CUDA [6], OpenCL [4], AMD Stream [1], DirectCompute [2], ir kt.. Anksčiau tokio tipo skaičiavimai buvo atliekami naudojant vaizdavimui (angl. *rendering*) skirtas procedūras (angl. *pipeline*) [5].

Šiame straipsnyje bus aptariama DirectCompute technologijos suteikiama nauda ir galimybės siekiant efektyviai išnaudoti grafinio procesoriaus resursus atliekant įvairaus pobūdžio skaičiavimus. DirectCompute yra DirectX programinio karkaso dalis [2]. Ši technologija buvo pasirinkta todėl, kad:

- DirectCompute palaiko tiek AMD, tiek NVIDIA firmų grafinius įrenginius. DirectCompute nepriklauso nuo konkretaus gamintojo.
- DirectCompute gali tiesiogiai naudoti DirectX pateikiamus resursus ir atvirkščiai, DirectX gali tiesiogiai naudoti DirectCompute pateikiamus resursus.

Šiame dokumente parodysime, kad GPU yra tinkamas ne vien 2D/3D grafikai atvaizduoti, bet ir padidinti lygiagretinamų algoritmų našumą. Visi algoritmai yra realizuoti C# kalboje pasinaudojant SlimDX biblioteką sąsajos su DirectX realizavimui [8].

9.1.2. Metodai

Siekiant įvertinti skirtingus algoritmų aspektus, pasirinkti trys algoritmai, kurie buvo realizuoti keliais būdais. Šiame skyriuje yra pateikti pasirinktų algoritmų aprašymai bei naudota tyrimų metodologija.

9.1.2.1. A* kelio paieškos algoritmas

Kompiuterių mokslų srityje A* algoritmas yra plačiai naudojamas trumpiausio kelio paieškai grafe. Algoritmas buvo plačiai aprašytas dar 1968 metais [3].

A* algoritmas naudoja geriausias pirmas (angl. *best - first*) paieškos taktiką ieškodamas trumpiausio kelio nuo pradinės viršūnės iki galinės. Grafo viršūnių apėjimo tvarką algoritme nusako euristinė funkcija, kuri yra dviejų papildomų funkcijų suma:

- Kelio funkcija, kuri aprašo kelio ilgį nuo pradinės viršūnės iki esamos viršūnės.
- Euristinė likusio kelio ilgio nuo esamos viršūnės iki galinės viršūnės įvertinimo funkcija.

Euristinė likusio kelio ilgio įvertinimo funkcija privalo nepervertinti kelio ilgio iki galinės viršūnes. Pagal šią funkciją apskaičiuota viršūnės vertė turi būti mažesnė nei faktinis kelio ilgis nuo tos konkrečios viršūnės iki galinės viršūnes.

Ieškant trumpiausio kelio, algoritmas A* parenka gretimą arčiausią viršūnę ir ją įsimena kaip trumpiausio kelio viršūnę. Tuo pačiu algoritmas įsimena nepasirinktų gretimų viršūnių prioritetinę eilę. Jei kuriuo nors metu nueitas kelias tampa ilgesnis, nei iki viršūnės esančios prioritetinėje eilėje, algoritmas tolimesnės viršūnės paiešką pradeda nuo viršūnės įsimintos prioritetinėje eilėje.

9.1.2.2. Hierarchinis uždengtos geometrijos atrinkimo algoritmas

Šis algoritmas yra skirtas 3D geometrijos vaizdavimo (angl. *rendering*) apdorojimo greičio optimizavimui. Algoritmas yra paremtas tuo, kad nėra prasmės atvaizduoti objektus, kurie yra uždengti ir yra nematomi vartotojui. Skaičiavimai yra atliekami remiantis vartotojui matomu hierarchiniu vaizduojamo pasaulio objektų gylio vaizdu (angl. *depth map*) [10].

Siekiant apskaičiuoti, kuriuos geometrijos objektus reik vaizduoti, o kurių ne, visi objektai yra padalinami į dvi grupes: uždengiantieji ir uždengtieji. Žemiau yra pateiktas hierarchinio uždengtos geometrijos atmetimo algoritmo žingsnių sąrašas:

1. Uždengiančiųjų objektų, papuolančių į vartotojui matomą zoną, išrinkimas (angl. *frustum culling*).
2. Uždengiančiųjų objektų gylio žemėlapio (angl. *depth map*) sudarymas.
3. Papuolančių į vartotojui matomą zoną potencialiai uždengtų objektų dengimo sąlygos išskaičiavimas.

Pirmasis žingsnis yra atliekamas CPU pagalba, o antrasis – GPU pagalba. Trečiasis žingsnis paprastai yra atliekamas CPU pagalba, nors gali būti atliktas ir GPU pagalba. Atliktų bandymų rezultatai pateikti 3.2 skyriuje.

9.1.3. Tyrimai

Šiame skyriuje yra pateiktos algoritmų vykdymo CPU ir GPU įrenginiuose laikinės charakteristikos. CPU pagalba vykdomi algoritmai yra priderinti lygiagrečiam vykdymui (angl. *multithreading*). Visi pateikti algoritmų vykdymų laikai yra apskaičiuoti kaip penkių bandymų vidurkis. Rezultatai užfiksuoti naudojantis tokia aparatūrine įranga:

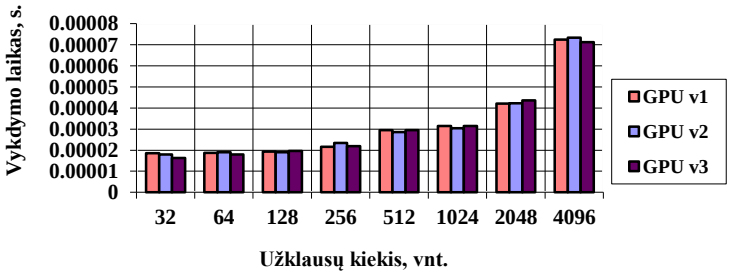
- Centrinis procesorius: Intel Core2 Duo E8400 @3.2 GHz.
- Operatyvioji atmintis: Kingston HyperX DDR2 4 GB @854 MHz.
- Grafinis procesorius: Gigabyte 450 GTS OC2.
- Operacinė sistema: Microsoft Windows 7 x64.

9.1.3.1. A* kelio paieškos algoritmas

Iš viso yra realizuotos trys GPU algoritmo ir viena CPU algoritmo versijos. „GPU v1“ algoritmas taupo GPU atmintį naudodamas vieno kintamojo bitus kelių kintamųjų informacijai saugoti. „GPU v2“ algoritmas yra toks pats kaip „GPU v1“ algoritmas, tačiau naudoja atskirą dar neištirtų viršūnių sąrašą (angl. *open node list*), kuriame nėra atliekamas kintamųjų informacijos suspaudimas. „GPU v3“ algoritmas nenaudoja visiškai jokio kintamųjų informacijos suspaudimo.

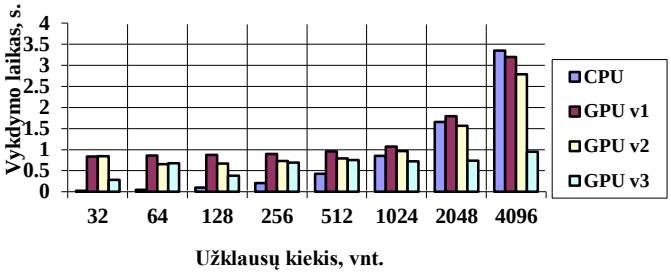
1 – 4 paveiksluose yra pateikti rezultatai atliekant skirtingus paieškos užklausų kiekius 64x64 viršūnių dydžio grafe. Grafo viduryje yra suformuota kliūtis, paliekanti tik siaurus praėjimus grafo kraštuose. Trumpiausio kelio paieška yra vykdoma iš (0,0) viršūnės į (63,63) viršūnę. Gijų grupės dydis yra 16.

1 paveiksle yra atvaizduotas duomenų perkėlimo iš CPU į GPU atminties buferius laikas. Kaip paaiškėjo, šie laikai kinta tam tikrose ribose, todėl pateikti 5 bandymų vidurkiai. Kaip matome, bendra laiko augimo, augant perkeliamų duomenų kiekiui, tendencija išlieka pastovi.



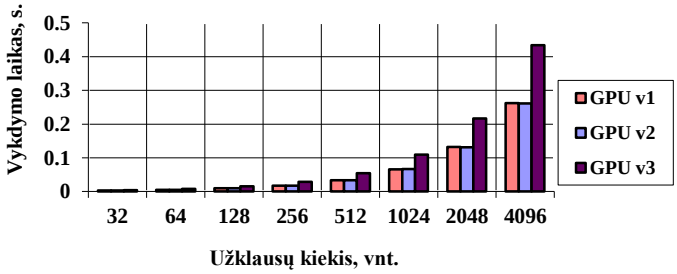
1 pav. Duomenų perkėlimo iš CPU į GPU atminties buferių laikas A* kelio paieškos algoritmo vykdymo metu.

2 paveiksle yra parodytas realizuotų CPU bei GPU algoritmų versijų skaičiavimų vykdymo laikų palyginimas. Suspaudžiant kintamųjų informaciją, saugojimo sąnaudas galime sumažinti nuo 20 iki 12 baitų, tačiau tai neigiamai įtakoja našumą. Lyginant skaičiavimų atlikimo laikus geriausiu atveju gauname ~3.52 kartų didesnį našumą naudojant GPU vietoj CPU, 4096 paieškos užklausų atveju.



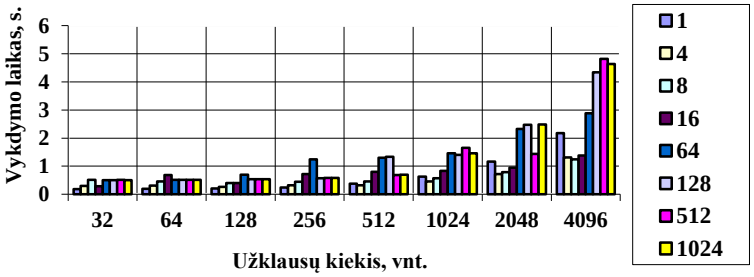
2 pav. A* kelio paieškos algoritmo versijų vykdymo laikų palyginimas.

3 paveiksle pateiktas duomenų perkėlimo iš GPU į CPU atminties buferių laikas yra pastebimai priklausomas nuo perkeliamos atminties dydžio.



3 pav. Duomenų perkėlimo iš GPU į CPU atminties buferių laikas A* kelio paieškos algoritmo vykdymo metu.

Gijų grupės dydis svarbus siekiant užtikrinti maksimalų algoritmo našumą. Tai aktualu yra ir tada, kai gijos grupėse tarpusavyje neatlieka jokių sinchronizavimo ar informacijos apsikeitimo veiksmų. Šie rezultatai tiesiogiai priklauso nuo GPU architektūros. Optimalus gijų grupės dydis pritaikytas prie paleidžiamų gijų grupių kiekio leidžia geriau išnaudoti bei pilniau užpildyti turimus aparatūrinius grafinio įrenginio resursus [9].

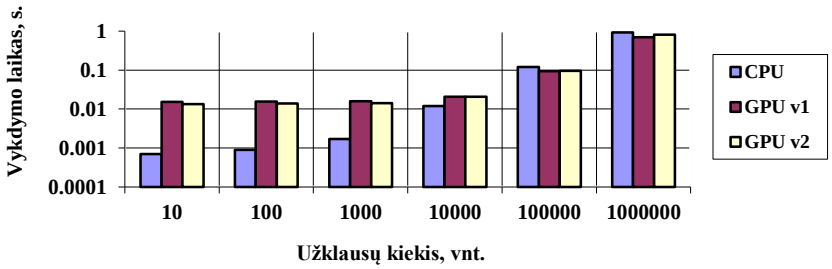


4 pav. Gijų grupės dydžio įtaka A* kelio paieškos algoritmo našumui.

4 paveiksle matome, kad esant skirtingam užklausų kiekiui reikia taikyti skirtingą gijų grupės dydį. Pavyzdžiui norint greičiau gauti 32 paieškos užklausų rezultatus optimalus gijų grupės dydis yra 1, tačiau kai paieškos užklausų kiekis yra 1024 – optimalus gijų grupės dydis tampa lygus 4. 64 ir didesnis gijų grupės dydis labai neigiamai įtakoja algoritmo našumą didėjant vykdomų užklausų kiekiui.

9.1.3.2. Hierarchinis uždengtos geometrijos atrinkimo algoritmas

Iš viso yra realizuotos trys algoritmo versijos. Pirmoji algoritmo versija 3-ią algoritmo žingsnį atlieka su CPU pagalba. „GPU v1“ algoritmo versija 3-ią algoritmo žingsnį atlieka su GPU pagalba. „GPU v2“ versija prieš atlikdama 3-ią žingsnį GPU pagalba, atlieka uždengtų objektų patekimo į vartotojui matomą zoną patikrinimą naudojant CPU. Gijų grupės dydis visai atvejais yra lygus 4. Objektai yra išdėstyti taip, kad dalis jų nepatenka į vartotojui matomą zoną.



5 pav. Hierarchinio uždengtos geometrijos atrinkimo algoritmo versijų vykdymo laikų palyginimas.

Tyrimas parodė, kad vaizdavimo atrinkimo testo skaičiavimų vykdymas pasitelkiant GPU naudingas tik tada, kai yra vykdomas didelis skaičiavimo užklausų kiekis. Atliekant skaičiavimus iki 100000 objektų, jų patekimo į vartotojui matomą zoną tikrinimą, yra efektyviau atlikti CPU pagalba. Šio tyrimo rezultatai parodo, kad algoritmams, kurie atlieka nedaug skaičiavimų, yra efektyviau naudoti CPU. Matome, kad situacija pasikeičia, kai skaičiavimų užklausų kiekis yra labai didelis.

9.1.4. Išvados

1. Protingas GPU resursų panaudojimas kartu su CPU suteikia nemažą našumo prieaugį skaičiavimui imliose užduotyse, kurių vykdymo algoritmas gali būti išlygiagretintas.
2. Mažai skaičiavimų atliekančių algoritmų vykdymas GPU pagalba naudingas tik vykdant didelį skaičiavimo užklausų kiekį.
3. Siekiant panaudoti GPU skaičiavimams atlikti, reikia įvertinti informacijos perkėlimo tarp CPU bei GPU atminties laikus. Būtent dėl to mažai skaičiavimų reikalaujantys veiksmai yra greičiau vykdomi CPU pagalba.
4. Gijų grupių dydis turi didelę įtaką GPU vykdomų algoritmų našumui. Tyrimų metu naudotos aparatūros atveju, optimalus gijų grupės dydis yra lygus 4, norint pasiekti didžiausią algoritmo našumo lygį.

Literatūros sąrašas

[1] **AMD**. ATI Stream Technology. *Advanced Micro Devices, Inc*. 2011 [žiūrėta 2011.03.03]. access via the Internet: http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/STREAM-TECHNOLOGY/Pages/stream-technology.aspx

[2] **Boyd C**. DirectCompute: Capturing the Teraflop. *PDC09, Microsoft Corporation*. 2009 [žiūrėta 2011.03.03]. access via the Internet: http://ecn.channel9.msdn.com/o9/pdc09/ppt/CL03.pptx

[3] **Hart P.E., Nilsson N.J., Raphael B.** A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE, Transactions on Systems Science and Cybernetics*. 1968, p.100–107.

[4] **KHRONOS**. OpenCL Overview. *KHRONOS Group*, 2011 [žiūrėta 2011.03.03]. access via the Internet: http://www.khronos.org/opencl/

[5] **Leubke D., Owens G**. A Survey of General-Purpose Computation on Graphics Hardware. *University of California, Davis, University of Virginia*. 2005 [žiūrėta 2011.03.03]. access via the Internet: www.seas.upenn.edu/~cis565/physicsCML.ppt

[6] **NVIDIA**. What is CUDA?. *NVIDIA Corporation*, 2011 [žiūrėta 2011.03.03]. access via the Internet: http://www.nvidia.com/object/what_is_cuda_new.html

[7] **Skadron K**. Thrends in Multicore Architecture. *ASPLOS '08*. 2008 [žiūrėta 2011.03.03]. access via the Internet: http://gpgpu.org/static/asplos2008/ASPLOS08-2-manycore.pdf

[8] **SlimDX**. What is SlimDX? *PDC09, SlimDX Group*. 2009-2011 [žiūrėta 2011.03.03]. access via the Internet: http://slimdx.org

[9] **Young E**. DirectCompute. Optimizations and Best Practices. *NVIDIA Corporations, GTC2010*. 2010 [žiūrėta 2011.03.03]. Access via the Internet: www.nvidia.com/content/GTC-2010/.../2260_GTC2010.pdf

[10] **Zhang H**. Effective Occlusion Culling for the Interactive Display of Arbitrary Models. *University of North Carolina, Chapel Hill*. 1998.

Improving calculation time by making use of graphical processor unit via DirectCompute technology

This paper describes how graphic processing unit can be used for general purpose computing and researches various factor impacts on its performance. DirectCompute technology was used to guide this research. Gathered data is based on two algorithms: A* path finding and Hierarchical occlusion.