KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Algis Pavasaris

Kelio paieškos algoritmo A* tyrimas skirtingų paskirčių procesoriuose

Magistro darbas

Darbo vadovas
doc. dr. Tomas Blažauskas

Kaunas, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Algis Pavasaris

Kelio paieškos algoritmo A* tyrimas skirtingų paskirčių procesoriuose

Magistro darbas

Vadovas
doc. dr. Tomas Blažauskas
2011-05

Atliko
IFM-9/2 gr. stud.
Algis Pavasaris
2011-05-25

Recenzentas
asist. dr. Algirdas Noreika
2011-05

Kaunas, 2011

Turinys

Kelio paieškos algoritmo A* tyrimas skirtingų paskirčių procesoriuose

Santrauka

Kelio paieška – tai trumpiausio maršruto tarp dviejų taškų radimas. Praktikoje kelio paieška taikoma šiose srityse: minių judėjimo modeliavimas, infrastruktūros planavimas, procesų modeliavimas, logistika ir kt.

Egzistuoja eilė kelio paieškos algoritmų, iš kurių vienas yra A* - trumpiausio kelio paieškos algoritmas. Šiame darbe yra tiriamas šio algoritmo veikimas skirtingų paskirčių procesoriuose, analizuojami kelio paieškos 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. Darbo pabaigoje aptariami gauti rezultatai bei pateikiamos išvados.

Raktiniai žodžiai

Trumpiausio kelio paieška, A* algoritmas, Grafinio procesoriaus panaudojimas bendro pobūdžio skaičiavimams, skaičiavimų spartinimas, lygiagretus skaičiavimai, DirectX 11, DirectCompute.

Analysis of A* path finding algorithm operation on different purpose processors

Summary

Path finding – it is the search of the shortest route between two points. In practice path finding is used in areas such as: crowd movement modeling, infrastructure planning, process modeling, logistics, etc.

There are various path finding algorithms and one of them is A* path finding algorithm. This document contains analysis of A* path finding algorithm operation on different purpose processors. In this document we provide a short summary of path finding algorithms, but its main focus is on improving A* path finding algorithm overall performance by making use of DirectCompute technology. This technology is a part of Microsoft DirectX 11 API, which helps the use of graphics processor for general-purpose computation. Main research goal is achieved by performing in-depth analysis of implemented A* path finding algorithm modifications. This analysis consists of both general performance and various performance affecting factors analyses. At the end of the document conclusions and recommendations are given based on performed work and overall results.

Keywords

Path finding, A* algorithm, using graphics processor for general-purpose computation, speeding up computations, parallel computing, 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-Purspose 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 atvaizdo paveikslas pagal pateiktus modelio duomenis.

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.

Agentas (angl. *agent*) – savarankiška esybė, kuri stebi bei renka duomenis apie aplinką, kurioje ji yra, ir elgiasi atitinkamai prisitaikant prie tos aplinkos pokyčių.

Daugiaagentinė sistema (angl. *multi – agent system*) – sistema, sudaryta iš daugybės tarpusavyje sąveikaujančių intelektualių agentų.

DirectCompute – programinė sąsaja, kuri leidžia panaudoti stipriai lygiagretinamą grafinio procesoriaus pajėgumą įprastos programinės įrangos skaičiavimams atlikti. DirectCompute yra Microsoft DirectX 11 programinių sąsajų rinkinio dalis.

Įpėdinio funkcija (angl. *successor function*) – Įpėdinio funkcija aprašo visus leistinus tolimesnius veiksmus esant konkrečioje būsenoje.

Būsenų erdvė (angl. *state space*) – tai aibė visų galimų būsenų, pasiekiamų iš pradinės viršūnės.

Kelio svorio funkcija (angl. *path cost*). Kelio svorio funkcija priskiria skaitinę reikšmę kiekvienam egzistuojančiam keliui. Kelio svorio funkcija yra aprašanti skaičiuojamą parametrą (pvz.: ilgis, srautas, ir t.t.).

AVS – atvirų viršūnių sąrašas. Kelio paieškos algoritme naudojamas saugoti potencialiai tinkamas kelio viršūnes.

UVS – uždarų viršūnių sąrašas. Kelio paieškos algoritme naudojamas saugoti peržiūrėtoms ir pilnai ištirtoms viršūnėms.

SLI (angl. *Scalable Link Interface*) – Nvidia kompanijos brandas, nusakantis aparatūrinį sprendimą, leidžiantį apjungti dviejų arba daugiau vaizdo plokščių pajėgumus bendram darbo našumui didinti.

CrossfireX – AMD kompanijos brandas, nusakantis aparatūrinį sprendimą, leidžiantį apjungti dviejų arba daugiau vaizdo plokščių pajėgumus bendram darbo našumui didinti.

2. ĮVADAS

Kelio paieška – grafų teorijos problema, bendru atveju formuluojama kaip trumpiausio maršruto tarp dviejų taškų radimas. Kelio paieškos terminas dažniausiai yra naudojamas elgsenos modeliavimo, logistikos ir kt. srityse. Praktikoje kelio paieška daugiau ar mažiau naudojama beveik visose srityse, tačiau kelio paieškos uždavinio sprendimo radimas yra skaičiavimams reiklus procesas.

Šiais laikais kelio paieška yra atliekama kompiuterio pagalba, bet augant kelio paieškos problemos sudėtingumui įprastai naudojamų kompiuterio resursų gali ir neužtekti. Pastaruoju metu tampa populiaru ne tik kelio paieškos problemą, bet ir kitus daug skaičiavimų reikalaujančius uždavinius spręsti papildomų kompiuterio resursų pagalba.

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 jų efektyviam panaudojimui [33]. Grafinio bei centrinio procesorių našumo augimo nevienodi tempai 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 [27], OpenCL [18], Accelerated Parallel Processing (APP) [1], DirectCompute [2], ir kt., įgalina bei supaprastina lygiagretinamų algoritmų atlikimą GPU pagalba. GPGPU technologijos gali būti panaudotos sprendžiant bendrinio pobūdžio uždavinius. GPGPU vystymu buvo siekta užtikrinti būtent pastarųjų uždavinių spartinimą, tačiau GPGPU technologijų išpopuliarėjimą labiausiai paskatino įvairaus pobūdžio 2D bei 3D grafikos vaizdavimu pagrįsta programinė įranga. Vienas iš plačiausiai ir geriausiai žinomų tokios programinės įrangos pavyzdžių yra kompiuteriniai žaidimai. Būtent juose pradėtas naudoti GPU daug skaičiavimų reikalaujantiems uždaviniams spręsti.

GPU pajėgumo augimas bei su tuo susijusių programinės įrangos technologijų kaita verčia permąstyti įvairių algoritmų realizacijos koncepcijas bei ieškoti naujų būdų esamų technologijų panaudojimui.

Ne išimtis yra ir trumpiausio kelio paieškos problema, kuri yra plačiau nagrinėjama skirtingų paskirčių procesorių panaudojimo kontekste. Dėmesys yra teikiamas GPGPU sprendimams ir kartu iškylančiai bendram atliekamų skaičiavimų nukėlimo iš CPU į GPU

problemai spręsti. Tai yra itin aktuali problema ir vis geresnių sprendimų radimo svarba šiame kontekste didės.

Šio darbo pagrindiniai tikslai yra:

- Ištirti GPGPU galimybių suteikiamą naudą atliekant trumpiausio kelio paiešką.
- Ištirti įvairių faktorių poveikį realizuoto kelio paieškos algoritmo našumui. Pagerinti algoritmo kiekybines ir kokybines charakteristikas su tiriama GPU architektūra.

Šiems tikslams įgyvendinti yra pasirinktas A* kelio paieškos algoritmas. Šiame darbe yra nagrinėjami trumpiausio kelio paieškos uždaviniai ir šios problemos sprendimo būdai. Taip pat yra pasiūlomos kelios algoritmo modifikacijos ir realizuojamos algoritmo versijos naudojančios tiek CPU, tiek GPU, bei atliekami realizuotų versijų tyrimai. Algoritmų realizacijose yra siekiama maksimaliai įtraukti grafinio procesoriaus panaudojimą. Darbo pabaigoje yra aptariami gauti rezultatai, pateikiamos bendros algoritmo naudojimo DirectCompute kontekste rekomendacijos ir siūlomų modifikacijų įvertinimas.

3. ANALITINĖ DALIS

3.1. Kelio paieškos problema

Formaliai kelio paieškos problema gali būti aprašyta keturiais komponentais:

- Pradinė būsena (angl. *initial state*). Pradinė būsena nurodo skaičiavimų pradžios tašką.
- Galimų veiksmų sąrašas (angl. *action list*). Dažniausiai naudojama įpėdinio (angl. *successor*) funkcijos sąvoka. Kiekvienai konkrečiai būsenai *x*, įpėdinio funkcija grąžina surikiuotą aibę (veiksmas, įpėdinis) porų, kuriose yra aprašomi visi leistini veiksmai esant būsenoje *x* ir visos įpėdinio būsenos, kurios gali būti pasiektos atlikus konkretų veiksmą. Pradinė būsena kartu su galimų veiksmų sąrašu apibrėžia kelio paieškos būsenų erdvę (angl. *state space*). Būsenų erdvė – tai aibė visų galimų būsenų, pasiekiamų iš pradinės viršūnės. Būsenų erdvėje kelias apibrėžiamas kaip būsenų seka, sujungta galimų veiksmų seka.
- Tikslo patikrinimo funkcija (angl. *goal test*). Tikslo patikrinimo funkcija skirta patikrinti ar esama būsena yra galinė būsena. Dažniausiai pasitaiko atvejai, kai galinių būsenų aibė yra griežtai apibrėžta, tačiau kartais galinė būsena gali būti apibrėžta kokiu tai parametru (pvz.: bendras kelio ilgis, bendras kelio svoris ir t.t.)
- Kelio svorio funkcija (angl. *path cost*). Kelio svorio funkcija priskiria skaitinę reikšmę kiekvienam egzistuojančiam keliui. Kelio svorio funkcija yra parenkama tokia, kad aprašytų siekiamą skaičiuoti parametrą (pvz.: kelio ilgis, kelio maksimalus srautas, ir t.t.).

Aukščiau išvardinti keturi komponentai pilnai apibūdina kelio paieškos problemą ir apjungti kartu į vientisą duomenų struktūrą gali būti panaudojami kaip problemos sprendimo algoritmo įėjimo duomenys. Algoritmui sėkmingai išsprendus užduotį gaunamas rezultatas, kuris yra kelias nuo pradinės iki galinės būsenos. Sprendimo kokybė yra matuojama kelio svorio funkcijos pagalba, todėl optimalus sprendimas yra mažiausią kelio svorį turintis sprendimas.

Realiame pasaulyje kelio paieška yra sudėtingesnis procesas, todėl norint išspręsti šią problemą tenka palikti tik aktualius, tiesiogiai su problema susijusius, duomenis. Procesas, kai šalinama perdėtina, neaktuali informacija, vadinamas abstrakcija. Kelio paieškos problemos atveju tenka abstraktuoti ne tik pačias būsenas, tačiau ir galimų veiksmų aibę.

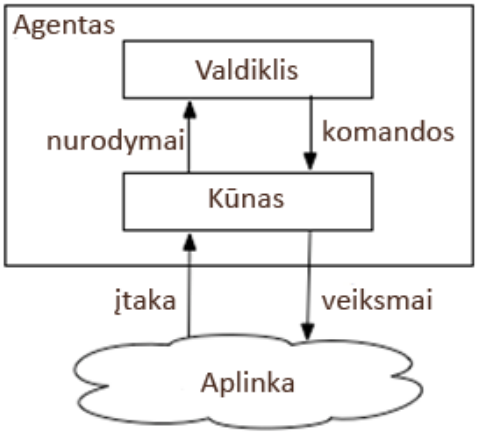
Konkrečiame paieškos algoritme tokia abstraktuota problema yra traktuojama kaip algoritmo įėjimo duomenys. Paieškos algoritmo išeitis yra sprendimas, kuris dažniausiai būna

konkrečių veiksmų seka. Kai sprendimas būna rastas, toliau gali būti atliekami įvairūs tarpiniai veiksmai.

Kelio paieškos problema dažniausiai sprendžiama dviem būdais: panaudojant agentines sistemas arba fizinės sąveikos sistemas. Todėl kelio paieškos procesas yra daugumos sumanių agentinių sistemų centrinis komponentas [29].

3.2. Agentinės sistemos

Agentas yra autonominė esybė, kuri stebi ir reaguoja į ją supančią aplinką ir atitinkamai keičia savo veiklą, norėdama pasiekti savo užsibrėžtus tikslus. Sumanūs agentai papildomai turi gebėjimą pasimokyti arba panaudoti anksčiau sukauptas žinias siekdami savo tikslų.



1 pav. Agentinė sistema ir jos komponentai.

Kaip matome iš 1 paveikslo agentinė sistema yra sudaryta iš agento ir jį supančios aplinkos. Agentas yra veikiamas aplinkos ir atlieka tam tikrus nustatytus veiksmus joje. Agentas yra sudarytas iš kūno ir valdiklių. Valdikliai gauna impulsus iš kūno ir siunčia kūnui komandas. Kūnui priklauso jutikliai, kurie paverčia aplinkos veiksnius į suvokiamą informaciją [9].

Paprasčiausiu atveju agento mąstysenai aprašyti, tai ką jis turi ar neturi daryti, agentas turi turėti būsenomis paremtą ir aprašytą aplinkos modelį. Tai gali būti plokščia, vieno lygio hierarchinė struktūra [22]. Tokiu atveju agentas gali pasiekti savo tikslus ieškodamas šiame modelyje savo esamos būsenos ir kelio iki tikslo būsenos. Jei yra randamas kelias, vadinasi agentas galės įvykdyti esamą užduotį.

Ši paieškos problema gali būti supaprastinta iki kelio paieškos grafe nuo pradinės viršūnės iki galinės viršūnės. Dauguma agentinių užduočių gali būti supaprastintos iki tokio aprašymo. Paieškos idėja yra tiesmuka: agentas suranda kelis dalinius galimus sprendimus esamai užduočiai

ir jas patikrina. Paieška tęsiama pakartotinai tikrinant kiekvieną dalinį sprendimą ir sustojant, kai pasiekiamas tikslas.

Dažniausiai agentui yra duodamas tik užduoties aprašymas, kuris leidžia vėliau atpažinti sprendimą, o ne algoritmas kaip išspręsti užduotį. Todėl agentas turi ieškoti užduoties sprendimo.

Paieškos sudėtingumas ir tas faktas, kad žmonės sugeba išspręsti kai kurias paieškos problemas, efektyviai leidžia daryti prielaidą, kad kompiuteriniai agentai papildomai turi išnaudoti sukauptas žinias, kurios žymiai palengvina paiešką. Tos papildomos žinios yra euristinės žinios [26].

Agentais paremtas minios judėjimo modelis yra studijuojamas įvairiose srityse: miesto planavimas, kelių projektavimas ir evakuacijos procesas [13].

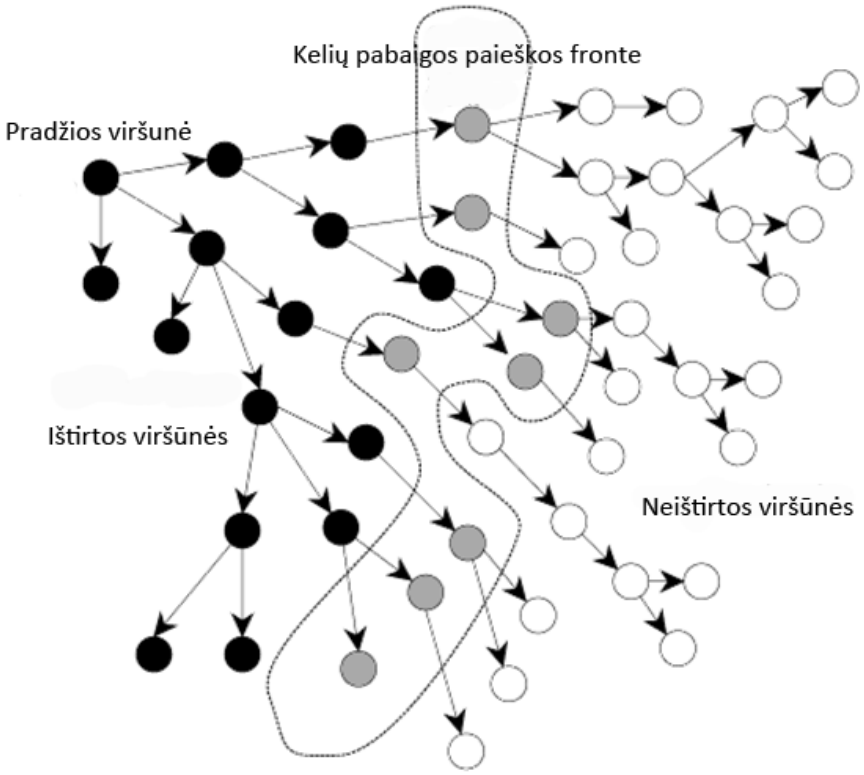
Dauguma atliktų darbų yra paremti agentais ir agentinėmis sistemomis. Šiose sistemose judėjimas, kelio paieškos problema yra sprendžiama kiekvienam agentui atskirai. Agentinės sistemos yra patrauklios dėl kelių priežasčių. Visų pirma, realiose miniose kiekvienas dalyvis priima nepriklausomus sprendimus, todėl tokie minios judėjimo modeliai leidžia pilnai atspindėti unikalią individo situaciją: matomumą, atstumą iki kitų pėsčiųjų ir kitus vietinius faktorius. Taip pat gali būti nustatyti skirtingi modeliavimo parametrai skirtingiems minios dalyviams. Tokiu būdu yra sukuriamas sudėtingas heterogeninis minios judėjimo procesas. Tačiau agentinės sistemos turi ir savo trūkumų. Yra itin sudėtinga sukurti ir aprašyti elgsenos taisykles, kurios modeliuotų realų minios judėjimą. Globalaus kelio paieškos užduotis kiekvienam agentui tampa itin intensyvi skaičiavimų požiūriu (ypatingai realaus laiko agentinėse sistemose). Siekiant apriboti agentinių sistemų trūkumų įtaką, dauguma agentinių sistemų modelių atskiria susidūrimo vengimą ir globalaus kelio paiešką į skirtingus skaičiavimo procesus. Tačiau tokiu atveju yra sukuriama terpė konfliktams, kai konkuruojantys procesai parenka skirtingas tikslų siekimo strategijas. Tuo pačiu vietinio kelio paieška dažnai pateikia trumparegiškus rezultatus, todėl bendras minios judėjimas atrodo mažiau realus. Šios problemos itin išryškėja tose vietose, kur būna didelių minių spūstys ar greitai besikeičianti aplinka.

Natūraliausius rezultatus pateikiantis būdas modeliuoti didelių minių judėjimą, remiantis agentinėmis sistemomis, yra naudoti metodus, kuriuose kiekvienas minios narys planuoja išskirtinai individualiai. Tokie metodai pateikia itin realius rezultatus, nes modeliavimo proceso metu kiekvienam dalyviui yra suteikiamos išskirtinės charakteristikos, kuriomis remiantis išryškėja minios dalyvio unikalumas. Itin sudėtinga tokio tipo dinaminė sistema buvo pademonstruota 1999 metais [11]. Joje buvo sėkmingai modeliuojama ne tik elgsenos dinamika,

bet ir pažintiniai (angl. *cognitive*) agento aspektai: sukauptos žinios ir mokymasis. Vėliau sistema buvo dar labiau patobulinta modeliuojant matomą agento lauką ir atliekant kelio paiešką [32].

Tačiau tokios sistemos kūrimas yra itin sudėtingas ir daug laiko atimantis procesas. Norint pagerinti agentinių sistemų metodų našumą ir išvengti pažintinio (angl. *cognitive*) modelio konstravimo proceso kiekvienam agentui, buvo tyrinėjami įvairūs supaprastinimai minios judėjimo modeliuose: vietinių agentų naudojimas, iš anksto paskaičiuoti statiniai keliai, globalaus kelio paieška naudojant grubų (angl. *coarse*) aplinkos grafą. Vietinius agentus pradėta naudoti dar 1987 metais [30], kai buvo pademonstruota, kad atsirandantis bandos (angl. *flocking*) elgesys miniose gali būti sugeneruotas iš paprastų vietinių taisyklių. Toks agentų panaudojimas leido sukurti vizualiai realų minios judėjimą. Po šios sistemos pristatymo buvo atlikta nemažai tyrinėjimų ir 1999 metais buvo sukurta praplėsta vietinių agentų agentinė sistema. Taip pat buvo tyrinėjami ir kiti elgseną įtakojantys aspektai, kaip sociologiniai klausimai [25], psichologiniai efektai [28], geografijos įtaka [35], socialinė įtaka [14][8], netriviali judėjimo dinamika [3].

3.2.1. Tipinis paieškos algoritmas



2 pav. Problemų sprendimas naudojant paiešką grafe.

Paieškos idėja yra paprasta. Turint grafą, pradinių viršūnių aibę ir galinių viršūnių aibę, nuosekliai patikrinti kelius, kurie veda iš pradinės į galinę viršūnę. Tai yra atliekama įsimenant paieškos frontą ir kelius, atvedusius iki fronto viršūnių. Fronte yra visos viršūnės, kurios gali būti pasiektos per atitinkamą kiekį žingsnių. Pradžioje frontas yra tuščias, tačiau didinant viršūnių kiekį, gaunamas naujas vis platesnis frontas, kol yra apžvelgiamas visas grafas. Priklausomai nuo naujų viršūnių įtraukimo į frontą, turime vis skirtingą paieškos strategiją.

```
1: Procedūra Paieška (G, S, tikslas)
2:      Įvestis
3:          G: grafas su N viršūnių ir A lankų
4:          S: pradinių viršūnių aibė
5:          tikslas: loginė būsenų funkcija
6:      Išvestis
7:          kelias nuo aibės S nario iki viršūnės, kuriai funkcija tikslas yra tiesa
8:          arba ⊥, jei nėra rastų sprendimo kelių
9:      Vietiniai kintamieji
10:         Frontas: aibė kelių
11:     Frontas ←{⟨s⟩: s∈S}
12:     kol (Frontas ≠{})
13:         išsirinkti ir ištrinti ⟨s0,...,sk⟩ iš Frontas
14:         jei ( tikslas(sk)) tada
15:             gražinti ⟨s0,...,sk⟩
16:         Frontas ← Frontas ∪{⟨s0,...,sk,s⟩: ⟨sk,s⟩∈A}
17:     gražinti ⊥
```

3 pav. Standartinis grafo paieškos algoritmas.

Pradžioje frontas yra tuščių kelių aibė. Kiekvienu žingsniu algoritmas plečią frontą iš jo ištrindamas kelią *⟨s0,...,sk⟩*. Jei *(sk)* yra galinė viršūnė, tada sprendimas yra rastas ir yra užfiksuojamas. Kitu atveju, kelias yra ilginamas dar vienu lankstu, surandant viršūnės *(sk)* kaimynus.

3.2.1.1. Algoritmų palyginimo kriterijai

Kelio paieškos algoritmai gali būti palyginami pagal:

- Laiką, kurio prireikė atlikti užduotį.

- Vietą, kurios prireikė saugoti informaciją.
- Kokybę arba rezultatų tikslumą.

Šių algoritmo parametrų įvertinimui skaičiuoti naudojama funkcija, priklausanti nuo pradinių duomenų kiekio. Tai vadinamas asimptotinis algoritmo sudėtingumas. Jis parodo kaip greitai keičiasi šie parametrai keičiant pradinių duomenų dydį [36].

3.2.2. Akla paieška

Kelio paieškos problemos sprendimas naudojant aklą (angl. *blind*) paiešką reiškia, kad šiuo principu veikiantys algoritmai neturi jokios papildomos informacijos apie būsenas, išskyrus aptartas skyriuje 3.1. Kelio paieškos problema.

Akliems kelio paieškos algoritmams visa prieinama informacija yra pradinė būsena, įpėdinio funkcija, tikslo patikrinimo funkcija ir kelio svorio funkcija. Viskas, ką gali atlikti šie kelio paieškos algoritmai, tai generuoti įpėdinius ir tikrinti ar buvo pasiektas tikslas.

Žemiau pateiktoje lentelėje aprašomi pagrindiniai aklos paieškos algoritmai pagal algoritmų palyginimo kriterijus aprašytus skyriuje 3.3.1.1. Algoritmų palyginimo kriterijai.

1 lentelė. Aklos paieškos algoritmų palyginimas.

	Kriterijus			
Algoritmas	Ar pilnas?	Laikas	Vieta	Ar optimalus?
Paieška platyn	Taip[a]	$O(b^{d+1})$	$O(b^{d+1})$	Taip[c]
Vienodų sąnaudų	Taip[a,b]	$O(b^{1+\lvert C^*/\varepsilon\rvert})$	$O(b^{1+\lvert C^*/\varepsilon\rvert})$	Taip
Paieška gilyn	Ne	$O(b^m)$	$O(bm)$	Ne
Gylio apribotas	Ne	$O(b^l)$	$O(bl)$	Ne
Iteracinio gilinimo	Taip[a]	$O(b^d)$	$O(bd)$	Taip[c]
Dvikryptis	Taip[a,d]	$O(b^{d/2})$	$O(b^{d/2})$	Taip[c,d]

Čia: b – šakojimosi faktorius; d – arčiausio (angl. *shallowest*) sprendimo gylis; m – maksimalus paieškos medžio gylis; l – paieškos gylio riba.

[a] – pilnas, jei b baigtinis; [b] – pilnas, jei žingsnio kaina (angl. *step cost*) $\geq \varepsilon$, teigiamiems ε; [c] – optimalus, jei visos žingsnio kainos identiškos; b baigtinis; [d] – jei į abi puses naudojamas paieškos platyn algoritmas.

Aukščiau esančioje lentelėje naudojamų kriterijų paaiškinimas:

- Ar pilnas? – ar algoritmas garantuotai suras sprendimą, kai toks egzistuoja?

- Laikas – kiek laiko reikės norint surasti sprendimą?
- Vieta – kiek atminties reikės norint surasti sprendimą?
- Ar optimalus? – ar algoritmas ras optimalų sprendimą?

Paieškos platyn algoritmas sekančią viršūnę parenka pačią arčiausią paieškos medyje. Algoritmas yra pilnas, optimalus vienetinio žingsnio atveju, jo laiko ir vietos sudėtingumas yra $O(b^{d+1})$. Užimamos vietos kiekis algoritmą padaro praktiškai beveik nepritaikomu.

Vienodų sąnaudų algoritmas yra panašus į paieškos platyn algoritmą, tačiau sekančią viršūnę parenka pagal mažiausią kelio svorio funkciją. Jis yra pilnas ir optimalus, jei žingsnio kaina viršija kokią nors teigiamą ribą ε.

Paieškos gilyn algoritmas sekančią viršūnę parenka giliausią paieškos medyje. Algoritmas nei pilnas, nei optimalus. Jo laiko sudėtingumas yra $O(b^m)$, o vietos – $O(bm)$, kur m – maksimalus paieškos medžio gylis.

Gylio apribotas algoritmas toks pats kaip ir paieškos gilyn algoritmas, tačiau jam yra nustatyta paieškos gylio riba.

Iteracinio gilinimo algoritmas - tai gylio apriboto algoritmo versija su nuolatos didinama paieškos gylio riba, kol yra nerastas tikslas. Algoritmas yra pilnas, optimalus vienetinio žingsnio atveju. Jo laiko sudėtingumas yra $O(b^d)$, o vietos – $O(bd)$.

Dvikryptis algoritmas gali itin smarkiai sumažinti laiko sudėtingumą, tačiau ne visada panaudojamas dėl išaugusių vietos reikalavimų.

3.2.3. Informuota paieška

Informuota paieška skiriasi nuo aklosios tuo, kad informuotos paieškos metu yra naudojamos papildomos, problemą apibūdinančios, sukauptos žinios. Šios sukauptos žinios yra surenkamos už kelio paieškos problemos apibrėžimo ribų ir jų panaudojimas leidžia surasti sprendimus žymiai greičiau ir efektyviau nei aklos paieškos būdu.

3.2.3.1. Inkrementinė paieška

Inkrementiniai paieškos algoritmai, sprendžiant paieškos problemas, kurios yra panašios, dažnai sugeba rasti trumpiausius kelius greičiau nei tai būtų įmanoma sprendžiant atskirai, nes papildomai panaudoja anksčiau atliktų paieškų informaciją [21].

Inkrementinės paieškos metodai nėra plačiai naudojami kelio paieškos užduočiai spręsti. Nors jie nėra plačiai naudojami, tačiau tyrimų ir mokymosi tikslais yra sukurta nemažai inkrementinių kelio paieškos algoritmų. Sukurtų algoritmų apžvalga yra pateikta [10] šaltinyje.

3.2.3.2. Euristinė paieška

Euristinė paieška yra kelio paieškos technika, kuri naudoja euristinę funkciją, kuri suteikia papildomą informaciją apie kelio paieškos algoritmo vykdomą sekantį žingsnį. Euristinė funkcija – tai gairė, kuri veda prie problemos sprendimo. Kadangi dauguma kelio paieškos problemų yra eksponentinio sudėtingumo prigimties, euristinių funkcijų panaudojimas kai kuriais kelio paieškos atvejais gali sumažinti algoritmų sudėtingumą iki polinominio sudėtingumo [31].

Sprendžiant didesnes kelio paieškos problemas specifinės srities žinios turi būti įtrauktos į sprendimo algoritmą. Tai gali būti informacija apie būsenų tipus, perėjimų tarp būsenų įvertinimai, tikslų charakteristikos ir t.t Visa ši papildoma informacija ir yra euristinės žinios, kurios yra transformuojamos į euristinę funkciją.

Kelio paieška, kai viena tarpinė būsena yra „geresnė“ nei kita, yra vadinama euristine kelio paieška. Euristinė funkcija yra labiausiai paplitusi papildomos informacijos perdavimo į paieškos algoritmą forma.

Yra visa geriausias – pirmas (angl. *best – first*) tipo kelio paieškos algoritmų šeima, skirta įvairių charakteristikų paieškai atlikti. Tačiau visos šios šeimos algoritmus jungia vienas bendras raktinis komponentas – euristinė funkcija. Dažniausiai euristinė funkcija bando teisingai nuspėti kelio svorį nuo esamos viršūnės iki galinės viršūnės.

3.2.4. Kelio paieškos algoritmai

Šiame skyriuje išvardinti pagrindiniai ir labiausiai paplitę kelio paieškos algoritmai. Šie algoritmai, išskyrus A* kelio paieškos algoritmą, pasižymi ir inkrementinėmis, ir euristinėmis savybėmis. A* kelio paieškos algoritmas pasižymi tik euristinėmis savybėmis.

3.2.4.1. A* kelio paieškos algoritmas

Kelio paieškos algoritmas A* yra labiausiai paplitęs euristinis paieškos algoritmas, kuris yra plačiai naudojamas programavime, dirbtinio intelekto ir robotikos srityse. Kompiuterių mokslų srityje A* algoritmas yra plačiai naudojamas trumpiausio kelio paieškai grafe. Algoritmas buvo plačiai aprašytas dar 1968 metais [12].

A* kelio paieškos algoritmas yra pirmos mažiausios kainos ir geriausias pirmas kelio paieškos algoritmų mišinys, kuris atsižvelgia ir į euristinės funkcijos reikšmę ir į kelio svorį. A* algoritmas naudoja geriausias pirmas (angl. *best - first*) paieškos taktiką, ieškodamas trumpiausio kelio nuo pradinės iki galinės viršūnių. Grafo viršūnių apėjimo tvarką algoritme nusako euristinė funkcija, kuri yra dviejų papildomų funkcijų suma:

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

Euristinė likusio kelio svorio įvertinimo funkcija privalo nepervertinti kelio svorio iki galinės viršūnes. Pagal šią funkciją apskaičiuota viršūnės vertė turi būti mažesnė nei faktinis kelio svoris 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 nuo viršūnės, esančios prioritetinėje eilėje, algoritmas tolimesnės viršūnės paiešką pradeda nuo viršūnės, įsimintos prioritetinėje eilėje.

3.2.4.2. Pakraščių taupymo A* kelio paieškos algoritmas

Pakraščių taupymo (angl. *fringe saving*) A* kelio paieškos algoritmas yra inkrementinė A* kelio paieškos algoritmo versija [34]. Pakraščių taupymo algoritmas nuo paprasto A* kelio paieškos algoritmo skiriasi tuo, kad trumpiausių kelių nuo pradinės iki galinės viršūnių paieškos metu kelio svorio funkcijos reikšmės gali didėti arba mažėti.

Pirmoji pakraščių taupymo algoritmo paieška yra tokia pati kaip ir algoritmo A*, tačiau kiekviena sekanti paieška yra spartesnė nei A*, nes sekančių paieškų metu pakraščių taupymo algoritmas pakartotinai panaudoja ankstesnės paieškos metu rastą paieškos medį. Panaudojama ta ankstesnio paieškos medžio dalis, kuri yra identiška esamos paieškos medžiui. Ankstesnės paieškos metu rasto paieškos medžio panaudojimas yra atliekamas atstatant neperžiūrėtų viršūnių sąrašą. Neperžiūrėtų viršūnių sąrašo atstatymo operacija yra vykdoma tol, kol esamas kelio paieškos medis sutampa su ankstesnės paieškos medžiu.

3.2.4.3. Apibendrintas adaptyvus A* kelio paieškos algoritmas

Adaptyvus (angl. *adaptive*) A* kelio paieškos algoritmas [19] yra inkrementinis euristinis paieškos algoritmas, kuris greičiau sprendžia panašias paieškos problemas nei paprastas A* kelio paieškos algoritmas, nes paieškos metu adaptyvus A* kelio paieškos algoritmas atnaujina euristinės funkcijos reikšmes, naudodamas ankstesnės paieškos metu surinktą informaciją.

Adaptyviame A* kelio paieškos algoritme euristinė funkcija yra transformuojama iš statinės (A* kelio paieškos algoritmo atveju) į dinaminę. Euristinės funkcijos reikšmės yra nuolatos keičiamos, atsižvelgiant į galinės viršūnės būseną, todėl kelio paieškos procesas tampa informuotas. Tokiu būdu euristinės funkcijos reikšmės tiksliau atspindi galinės viršūnės būseną.

3.2.4.4. Bendro planavimo A* kelio paieškos algoritmas

Bendro planavimo (angl. *lifelong planning*) A* kelio paieškos algoritmas yra inkrementinis euristinės paieškos algoritmas, panašus į A* kelio paieškos algoritmą [20]. Šiame algoritme yra panaudotas inkrementinės paieškos metodas, panašus į dinaminį, griežtai silpnai geresnės funkcijos fiksuoto taško problemos (angl. *Dynamic Strict Weakly Superior Function – Fixed Point Problem,* toliau - *SWSF-FP*) metodą.

Vietoj to, kad naujai atlikti kelio paiešką nuo pradinės iki galinės viršūnės (kelio paieškos A* algoritmo atveju), bendro planavimo A* algoritmas panaudoja ankstesnėse paieškose sukauptus duomenis. Tokiu būdu galima itin paspartinti skaičiavimų laiką. Algoritmas įgavo tokį pavadinimą būtent dėl sugebėjimo panaudoti ankstesnių paieškų duomenis.

Pirmos paieškos metu bendro planavimo A* kelio paieškos algoritmas suranda tokį patį kelią kaip ir algoritmas A*. Šis rastas kelias yra įsimenamas ir kiekvienos kitos paieškos metu ieškomi keliai gali naudotis šia įsiminta informacija. Taip yra sumažinamas reikalingų skaičiavimų kiekis, ieškant naujų kelių, nes kelio dalis jau būna surasta.

Toks anksčiau rastų ir įsimintų kelių panaudojimas taip pat gali sukelti ir problemas. Dažnai problemos kyla, jei paieškos metu yra pakeičiama kelio aplinka arti pradinės viršūnės. Tokiu atveju tenka perskaičiuoti didžiąją dalį rastų kelių, nes senieji pasidaro netikslūs. Todėl kartais bendro planavimo A* kelio paieškos algoritmas gali užtrukti ilgiau nei paprastas A* kelio paieškos algoritmas.

3.3. Fizinės sąveikos sistemos

Kelio paieškos problemą ir tuo pačiu minios judėjimo problemą galima spręsti ir kitu būdu – sukuriant fizinės sąveikos sistemas. Šis būdas atsirado sprendžiant skysčių dinamikos problemas. Viena iš pirmųjų tokių fizinės sąveikos sistemų buvo pademonstruota 2003 metais [16], kurioje minios nariai buvo atvaizduojami kaip vientisas tankio laukas ir kelio paieškos problema minioje buvo sprendžiama dvejomis dalinėmis diferencinėmis lygtimis. Svarbiausia šioje sistemoje buvo tai, kad evoliucionuojančios potencinio lauko funkcijos buvo aprašytos taip, kad tankio laukas optimaliai judėtų link savo tikslo. Detalesnis sistemos aprašymas yra pateiktas [17] šaltinyje. Ši sistema buvo sėkmingai pritaikyta viduramžių mūšių analizei atlikti [6]. Teisingas vienmatės šios sistemos versijos veikimas buvo patikrintas su empiriškai surinktais duomenimis [15] ir gauti rezultatai parodė sistemos tikslumą. Alternatyvūs vienmačiai nepertraukiamo srauto minios modeliai buvo išanalizuoti 2005 metais [7]. Taip pat yra sukurtos fizinės sąveikos sistemos, kurios realiai atvaizduoja minios judėjimo aspektus [5] ir pasižymi tokiais požymiais kaip grūsčių vengimas.

3.3.1. Nepertraukiamo srauto minios

Nepertraukiamo srauto (angl. *continuum crowds*) minios – tai realaus laiko minios modelis, paremtas nepertraukiamo srauto minios dinamika. Šis modelis yra realizuotas dinaminio potencialų lauko pagalba, kurį pasitelkus yra realizuotas globalios navigacijos ir dinaminių judančių objektų (kitų žmonių, mašinų ir t.t.) sąryšis. Tai leidžia efektyviai skaičiuoti didelės minios judėjimą išvengiant jų tarpusavio kolizijų skaičiavimų būtinumo.

Šitas metodas yra išskirtinis tuo, kad yra realizuotas be agentais paremtos dinamikos. Jame srautų judėjimas yra traktuojamas dalelių potencialų minimizavimo uždavinio sprendimu, kuris paklūsta ištisinės aplinkos dėsniams visos sistemos mastu. Rezultate yra gaunama aibė dinaminių potencialų laukų, pagal kuriuos yra modeliuojami kiekvieno individo minioje judėjimo veiksmai.

3.4. DirectCompute technologija

DirectCompute yra Microsoft DirectX 11 programinio bibliotekų paketo dalis, skirta bendro pobūdžio skaičiavimų vykdymui grafinio įrenginio pagalba [2]. Ši technologija yra ypatingai svarbi būsimiems DirectX 11 technologija paremtos programinės įrangos kūrėjams, nes priešingai nei jos konkurentai, suteikia galimybę lengvam resursų apsikeitimui tarp DirectX ir

DirectCompute. Tai reiškia, kad vaizdavimo procedūroje naudotą arba dar tik planuojamą naudoti resursą galima skaityti arba modifikuoti skaičiavimų paprogramėje (angl. *compute shader*). Tai suteikia eilę naujų galimybių įvairių algoritmų kūrimui.

Programos, skirtos vykdymui su DirectCompute, kaip ir bet kokios kitos grafinės paprogramės atveju, turi būti parašytos kaip metodai aukšto lygio grafinėje kalboje (angl. *High Level Shading Language – HLSL*) [23]. 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 vieną paleidimą negali viršyti 65535 grupes. Gijų kiekis grupėse negali viršyti 1024 gijas (leidžiant jas su DirectX 11 aparatūrine įranga) arba 768 gijas (leidžiant jas su DirectX 10 aparatūrine įranga). Priklausomai nuo aparatūrinės įrangos skiriasi ir prieinamas bendros atminties kiekis: DirectX 11 atveju jis yra 32 KB, o DirectX 10 atveju – tik 16 KB.

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

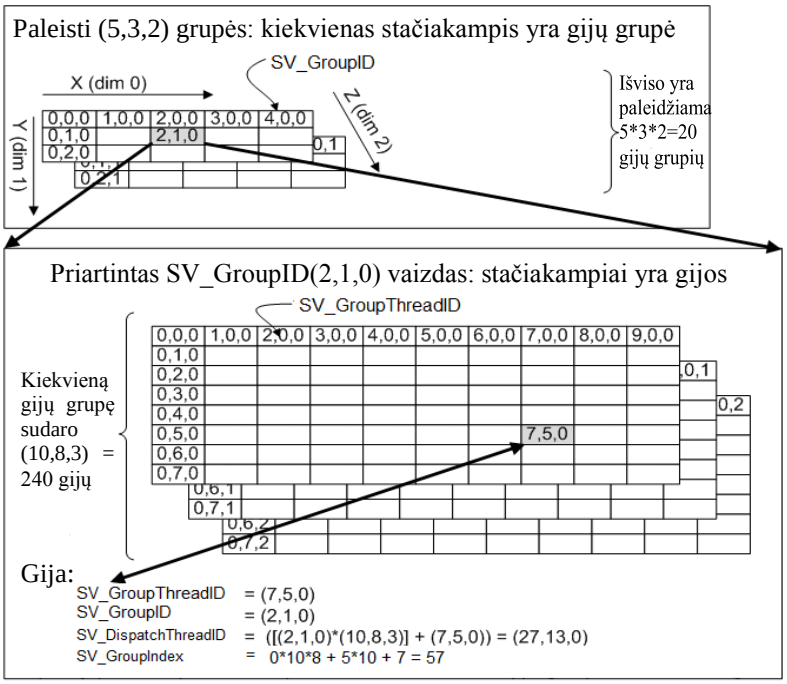
- Tik skaitymą (angl. *read-only*) palaikantys buferiai (paprasti, struktūriniai, vartojimo (angl. *consume*) ir kiti);
- Tik rašymą (angl. *write-only*) palaikantis pridėjimo (angl. *append*) buferis;
- Tiek rašymą, tiek skaitymą (angl *read-write*) palaikantys buferiai (paprasti, struktūriniai ir nerikiuojamos prieigos (angl. *unordered access*)).

Rašymą palaikantys atminties buferiai išlaiko savo informaciją, 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 nei 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 [24]. 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



4 pav. Gijų grupių ir gijų grupėse indeksavimo pavyzdys.

4 paveiksle yra pateiktas 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 [4]:

- Pateikti pakankamai skaičiavimų, 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 konfliktų. 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.

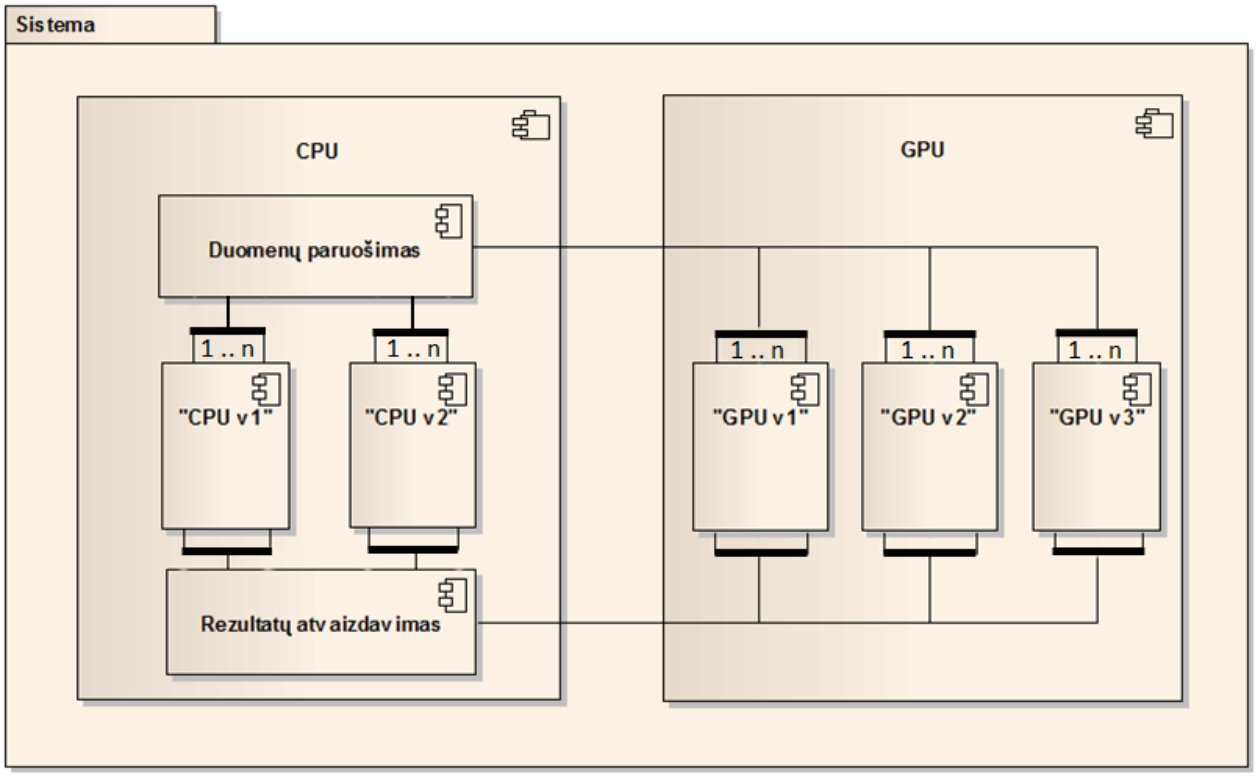
3.5. Pasirinkto metodo pagrindimas

Kelio paieškos tyrimui skirtingų paskirčių procesoriuose buvo pasirinktas informuotos paieškos, euristinis kelio paieškos A* algoritmas. Šis algoritmas buvo pasirinktas nes:

1. Šis algoritmas yra lengvai suprantamas ir jo realizacija nereikalauja didelių pastangų bei sudėtingų skaičiavimų.
2. Šis algoritmas yra plačiai paplitęs ir naudojamas kaip mokymo priemonė, tiriant įvairius kelio paieškos aspektus.
3. Šis algoritmas yra lengvai lygiagretinamas, nes kiekviena kelio paieška gali būti atlikta nepriklausomai nuo kitos.
4. Remiantis 3 punktu galima teigti, kad algoritmas tinkamas vykdyti GPU pusėje.
5. Skaičiavimų perkėlimas į GPU pusę leis atlaisvinti CPU resursus kitiems skaičiavimams, vykdomiems šio algoritmo panaudojimo kontekste.

4. PROJEKTINĖ DALIS

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

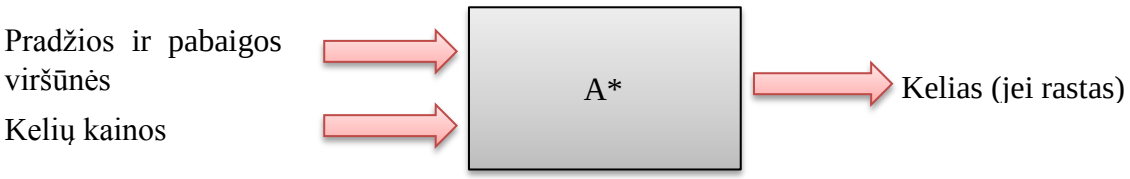


5 pav. Sistemos komponentų veikimo konteksto diagrama.

5 paveiksle matome, kuriuose procesoriuose yra vykdomi realizuoti algoritmai ir kaip yra paskirstyti lygiagrečiai veikiantys komponentai. Lygiagrečiai veikiančių komponentų kiekis gali būti parenkamas nuo 1 iki *n*. CPU pusėje vykdomų algoritmų skirtumai detaliau aptariami skyriuje 4.2.2. Algoritmo optimizacija. Specifinių GPU pusėje vykdomų algoritmų savybių skirtumai aprašyti skyriuje 4.3. Algoritmo atminties struktūros elementai.

4.1. Pradinio algoritmo struktūra

6 paveiksle yra pateiktas bendro kelio paieškos A* algoritmo vaizdas. Kaip matome, teisingam algoritmo veikimui reikalinga iš išorės pateikti pradžios ir pabaigos viršūnes bei grafo kelių kainas. Kitos kelio paieškos problemos dalys (aprašytos skyriuje 3.1. Kelio paieškos problema) dažniausiai būna aprašytos paties algoritmo viduje.



6 pav. Bendras kelio paieškos A* algoritmo vaizdas.

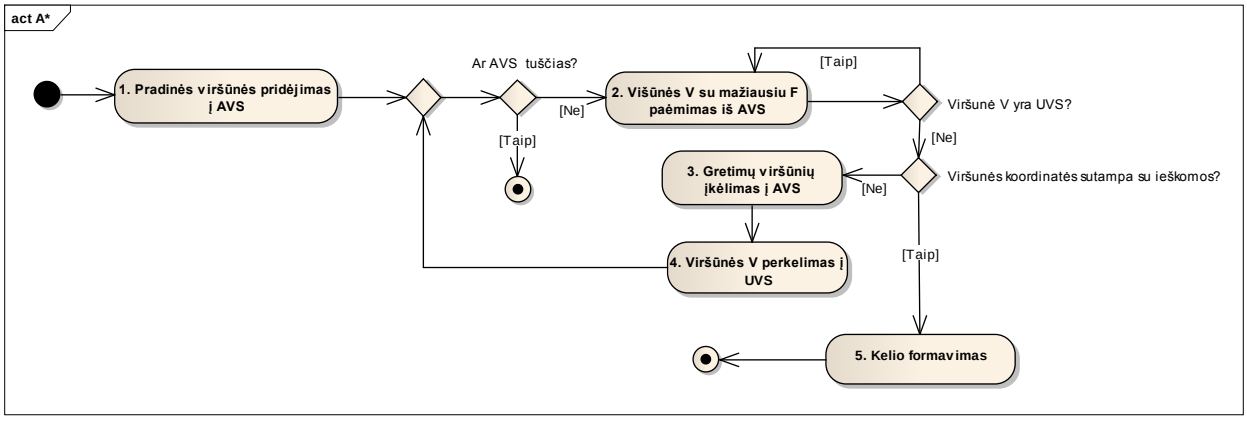
Kelio paieškos A* algoritmas galėtų būti aprašytas taip, kaip pavaizduota 7 paveiksle.

```
1:   Procedūra A*(pradžia, pabaiga)

2:   Uždarytų_viršūnių_aibė = {}
4:   Atvirų_viršūnių_aibė = {⟨pradžia⟩}
5:   Kelio_viršūnės[] = {}
6:   Esamo_kelio_įvertis[pradžia] = 0
7:   Euristinis_įvertis[pradžia] = Euristinė_funkcija(pradžia, pabaiga)
8:   Bendro_kelio_įvertis[pradžia] = Euristinis_įvertis[pradžia]

9:   Kol (Atvirų_viršūnių_aibė ≠{})
10:      S := viršūnė iš Atvirų_viršūnių_aibė su mažiausia Bendro_kelio_įvertis[] reikšme
11:      Jei S yra tikslas Grąžinti Kelio_atstatymas(Kelio_viršūnės, Kelio_viršūnės[tikslas])
12:      Pašalinti S iš Atvirų_viršūnių_aibė
13:      Įtraukti S į Uždarytų_viršūnių_aibė
14:      Kiekvienai viršūnei T gretimai S
15:          Jei T priklauso Uždarytų_viršūnių_aibė tęsti
16:          Tiriamo_kelio_įvertis := Esamo_kelio_įvertis[S] + Atstumas(S, T)
17:          Jei T nepriklauso Atvirų_viršūnių_aibė
18:              Įtraukti T į Atvirų_viršūnių_aibė
19:              Tiriamas_kelias_geresnis := tiesa
20:          Kitu atveju jei Tiriamo_kelio_įvertis < Esamo_kelio_įvertis[T]
21:              Tiriamas_kelias_geresnis := tiesa
22:          Kitu atveju
23:              Tiriamas_kelias_geresnis := melas
24:          Jei Tiriamas_kelias_geresnis
```

```
25:         Kelio_viršūnės[T] := S
26:         Esamo_kelio_svorio_įvertis[T] = Tiriamo_kelio_įvertis
27:         Euristinės_funkcijos_įvertis[T] = Euristinė_funkcija(T, pabaiga)
28:         Bendro_kelio_įvertis[T] = Esamo_kelio_įvertis[T] + Euristinis_įvertis[T]
29:   Grąžinti „Kelio nėra“
```

7 pav. Kelio paieškos A* algoritmo metu vykdomi veiksmai.

Apibendrinus kelio paieškos A* algoritmo metu vykdomas operacijas, jas galime sugrupuoti į atitinkamus žingsnius. 8 paveiksle yra pateiktas žingsninis kelio paieškos A* algoritmo vaizdas, o patys žingsniai detaliau aprašyti skyriuje 4.4. Algoritmo žingsnių realizacijos detalizavimas. Paveiksle naudojamos atvirų viršūnių sąrašo (toliau – AVS) ir uždarytų viršūnių sąrašo (toliau – UVS) santrumpos.



8 pav. Scheminis kelio paieškos A* algoritmo vaizdas.

4.2. Algoritmo modifikacijos

Šiame skyriuje bus aprašomos atliktos kelio paieškos A* algoritmo modifikacijos.

4.2.1. Algoritmo nustatymai

Norint sėkmingai atlikti kelio paieškos A* algoritmo tyrimą skirtingų tipų procesoriuose bei įvertinti atskirus algoritmo vykdymo aspektus buvo įvesti žemiau išvardinti algoritmo nustatymai:

- Įstrižainės. Leidimas arba draudimas algoritmui kelio paiešką atlikti įstriža kryptimi.
- Įstrižainių svoris. Pridedamas papildomas svoris, jei algoritmas sekantį laukelį pasirenka įstriža kryptimi.

- Krypties keitimo svoris. Pridedamas papildomas svoris, jei algoritmas dažnai keičia paieškos kryptį. Nustačius tinkamą krypties keitimo papildomą svorį, surasti keliai būna tiesesni ir natūralesni, tačiau algoritmas veikia lėčiau.
- Uždarytų viršūnių peržiūra. Leidimas arba draudimas algoritmui iš naujo peržiūrėti anksčiau aplankytas viršūnes. Antrinė viršūnių peržiūra kartais leidžia surasti geresnį ir lygesnį kelią
- Euristinė funkcija. Galima keisti euristinės funkcijos išraišką. Šios funkcijos keitimas gali kardinaliai pakeisti kelio paieškos algoritmo trukmę.
- Euristinės funkcijos parametras. Papildomas parametras naudojamas euristinės funkcijos skaičiavimo metu.
- Paieškos dydis. Peržiūrėtų viršūnių kiekis, kai tariama, kad kelio nuo pradinės viršūnės į galinę nėra.

Šių nustatymų įvedimas kelio paieškos A* algoritmui suteikia lankstumo, nes galima reguliuoti ir keisti kelio paieškos parametrus, kol gaunamas tenkinamas rezultatas. Nustatymų pagalba, algoritmo randamas kelias dažniausiai būna tiesesnis, tikslesnis ir natūralesnis.

4.2.2. Algoritmo optimizacija

Realizavus paprastą kelio paieškos A* algoritmo versiją buvo pastebėta, kad kelio paieškos operacija trunka itin ilgai. Tai labiau pasireikšdavo ieškant kelio dideliuose žemėlapiuose ir tais atvejais, kai kelias nebuvo randamas.

Peržvelgus pirminę algoritmo realizaciją pamatėme kelias vietas, kuriose algoritmas galėtų būti paspartintas, todėl atlikome kelias paieškos A* algoritmo optimizacijas.

Visai viršūnės informacijai saugoti buvo panaudotas struktūrinis elementas. Ir visa informacija apie viršūnės būseną (atvira, uždaryta, kelio svorio reikšmė, tėvinė viršūnė ir t.t.) buvo perkelta į šį elementą. Tokiu būdu informacija, kuri įprastai saugoma A* algoritmo viršūnių masyvuose tapo perteklinė ir ją galima buvo šalinti. Sukūrus šį struktūros elementą, tuo pačiu papildomai buvo sukurta viršūnių informacijos matrica, sudaryta iš viršūnių struktūrinių elementų. Jos dėka galima buvo visiškai atsisakyti uždarytų viršūnių sąrašo ir užklausos apie viršūnės būseną buvo aptarnaujamos iš karto, žemėlapio koordinatėmis nusakant viršūnės poziciją viršūnių informacijos matricoje. Tačiau papildomos matricos sukūrimas padidino naudojamos atminties išteklius. Naudojamiems atminties ištekliams sumažinti buvo pakeista kai kurių algoritme naudojamų kintamųjų tipai į mažiau atminties naudojančius tipus. Tipų

pakeitimas leido sutaupyti šiek tiek atminties resursų, tačiau bendras algoritme naudojamos atminties kiekis padidėjo.

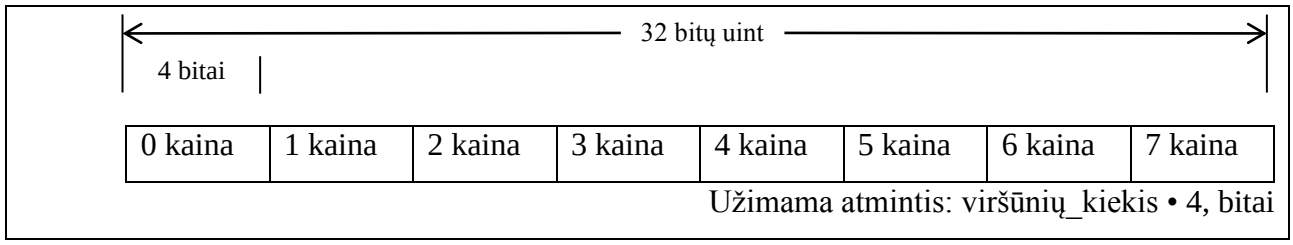
Atvirų viršūnių sąrašas buvo pakeistas į prioritetinę eilę. Taip pat naujų viršūnių įrašymo, ar esamų viršūnių naikinimo operacijos metu naudojamos prioritetinės eilės indeksas, buvo realizuotas panaudojus skaičiavimų matricoje naudojamas žemėlapio koordinates.

Viršūnių informacijos saugojimo matrica iš dvimatės buvo pakeista į vienmatę. Taip pat buvo įvestas apribojimas žemėlapio dydžiui. Žemėlapio kraštinės ilgis turėjo būti 2^n langelių. Tokiu būdu viršūnių vietos radimo operacijos matricoje galėjo būti atliekamos loginių, o ne matematinių operacijų dėka. Tai itin paspartino algoritmo darbą.

Buvo pakeistas po kiekvienos kelio paieškos A* algoritmo užklausos atliekamas viršūnių būklės (atvira, uždaryta) atstatymas. Visų viršūnių reikšmių atstatymas užima papildomai laiko, todėl buvo nuspręsta naudoti vis didėjančius skaičius siekiant nusakyti viršūnės būseną. Pirmos kelio paieškos A* algoritmo užklausos atveju atviros viršūnės bus išskirtos požymiu – „1“, o uždaros požymiu – „2“. Sekančios užklausos metu, viršūnių požymių skaitinės reikšmės bus padidintos dviem ir bus lygios: atviroms viršūnėms – „3“, uždaroms – „4“ ir t.t., visos kitos likusios skaitinės reikšmės nuo ankstesnio skaičiavimo bus ignoruojamos.

4.3. Algoritmo atminties struktūros elementai

Šiame skyriuje yra aprašoma GPU pusėje vykdomų kelio paieškos A* algoritmų naudojama atminties duomenų struktūrų architektūra. Skirtingos architektūros panaudojimo dėka buvo sukurtos kelios algoritmo versijos. Visos iš šių versijų naudoja vienodą kelio svorio elemento struktūrą, kuri yra pateikta 9 paveiksle. Kelių svorio elementai yra saugomi 32 bitų *uint* tik skaitymui (angl. *read-only*) skirtoje struktūroje, kurioje į kiekviena *uint* elementą yra įtalpinti po 8 svorio elementus (kiekvienam kelio elementui yra skiriami 4 bitai). Pavyzdžiui, 0-ojo *uint* elemento 1-jame sub-elemente yra saugoma 1-ojo kelio svoris, o 1-ojo uint elemento 2-jame sub-elemente yra saugoma 11-ojo kelio svoris.

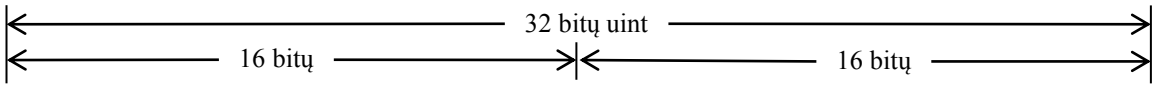


9 pav. Kelių svorio saugojimo elemento struktūra.

Žemiau yra pateikta „GPU v1“ algoritmo naudojama atminties duomenų struktūrų architektūra. Ši algoritmo versija maksimaliai taupo naudojamą GPU atmintį, ir taip pat naudoja ją skirtingoms paskirtims algoritmo vykdymo ir algoritmo rezultatų grąžinimo atveju (pavaizduota 10 ir 11 paveiksluose). Abiem atvejais yra naudojamas 3 *uint* pločio elementas, kurio atmintis yra panaudojama skirtingais tikslais. Kiekvienai GPU gijai yra sukuriama atskira darbo sritis, kurios dydis yra lygus grafo viršūnių kiekiui.

9 paveiksle vienas elementas saugantis *F*, *G*, *ParentX*, *ParentY* ir *Node status* kintamųjų duomenis atitinka vieną grafo viršūnę. Prioritetinė eilė (angl. *priority queue*, toliau – PQ) yra saugoma pradedant 0 tos gijos atminties zonos elementu, ir naudoja sekančių viršūnių elementus PQ indekso didėjimo tvarka. Tai reiškia, kad maksimalus PQ ilgis „GPU v1“ algoritme yra lygus grafo viršūnių kiekiui.

11 paveiksle išėjimo struktūros elementai yra naudojami rastam keliui išvesti, arba nurodyti, kad jis nebuvo rastas. Informacija apie kelią yra talpinama 0-inio gijos atminties zonos elemento 2 *uint* kintamajame, kurio aukščiausiose 16 bituose yra įrašoma ar kelias rastas ar ne, ir jei rastas – tai žemiausiose to kintamojo 16 bituose yra įrašomas rasto kelio ilgis. Kelias yra gaunamas nuskaitant 0 gijos atminties zonos elementus zonos indekso didėjimo tvarka, kur aukščiausi 16 bitų saugo kelio X, o žemiausi 16 bitų – kelio Y viršūnių indeksus.

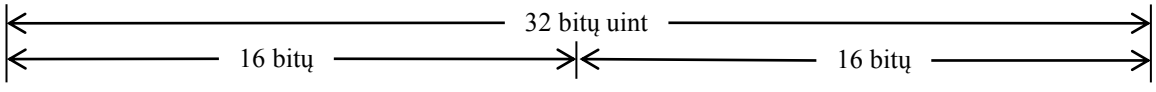


F	G
Parent X	Parent Y
Node status (OPEN, CLOSED)	PQ (stores idexes sorted by F)

Užimama atmintis: 32 • viršūnių_kiekis • 3, bitai

PQ maksimalus ilgis = viršūnių_kiekis

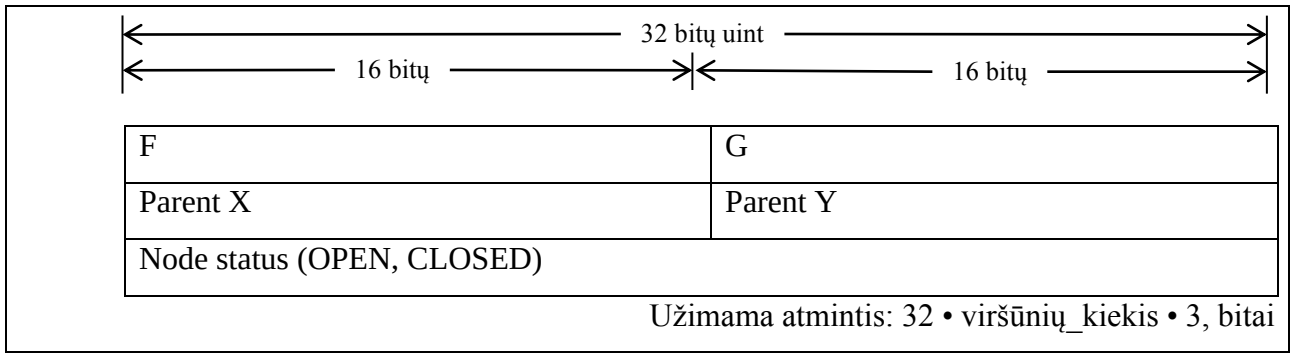
10 pav. „GPU v1“ algoritmo skaičiavimų vykdymui naudojamas struktūros elementas.



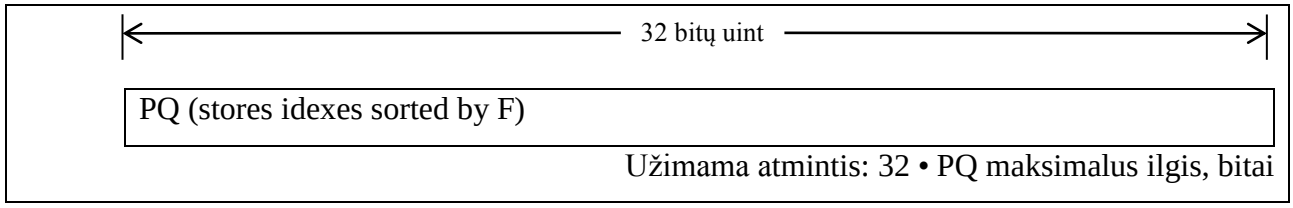
Path X	Path Y
Parent X (not used outside)	Parent Y (not used outside)
[0]: Path status (FOUND, NOT FOUND); [1-*]:not used outside	[0]: Path length (if found); [1-*]:not used outside

11 pav. „GPU v1“ algoritmo rezultatų gražinimui naudojamas atminties struktūros elementas.

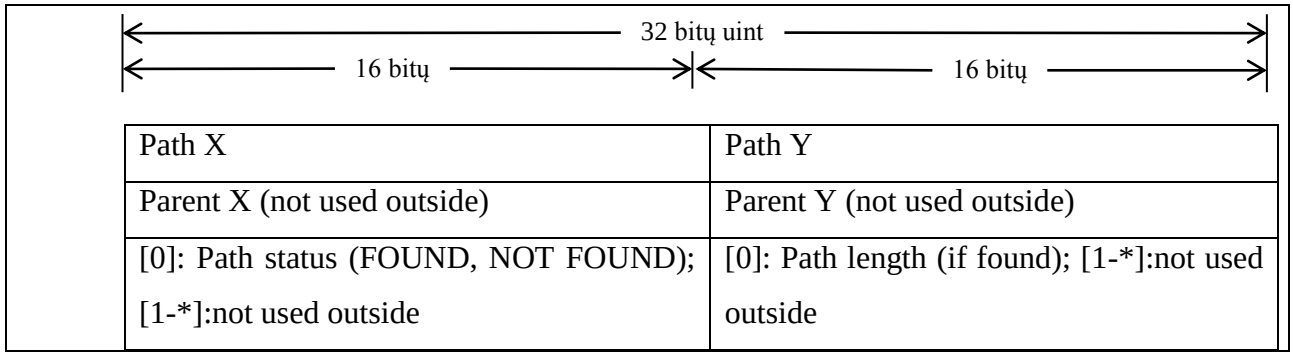
Toliau yra pateikta „GPU v2“ algoritmo naudojama atminties duomenų struktūrų architektūra, kuri yra labai panaši į „GPU v1“ algoritmo naudojamą architektūrą. Esminis skirtumas yra tas, kad PQ elementas yra saugomas atskiroje duomenų struktūroje (pavaizduota 13 paveiksle). Todėl 12 paveiksle matome, kad elemente 2 *uint* kintamasis išnaudoja visus 32 bitus, kaip ir PQ saugomi elementai. Tai reiškia, kad operacijų kiekiui imlioje vietoje yra nenaudojamas kintamųjų atminties suspaudimas, o tai padidina atminties sąnaudas, tačiau suteikia didesnį algoritmo našumą (tyrimas nr.1 ir tyrimas nr. 2). PQ duomenų struktūros informacija nėra grąžinama į CPU pusę.



12 pav. „GPU v2“ algoritmo skaičiavimų vykdymui naudojamas struktūros elementas.



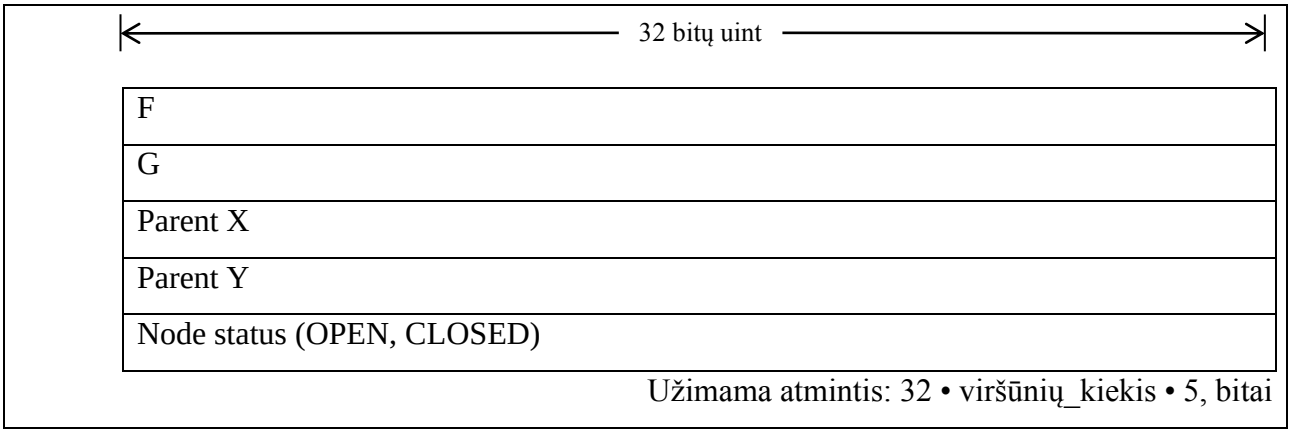
13 pav. „GPU v2“ algoritme naudojamas laikinos struktūros elementas.



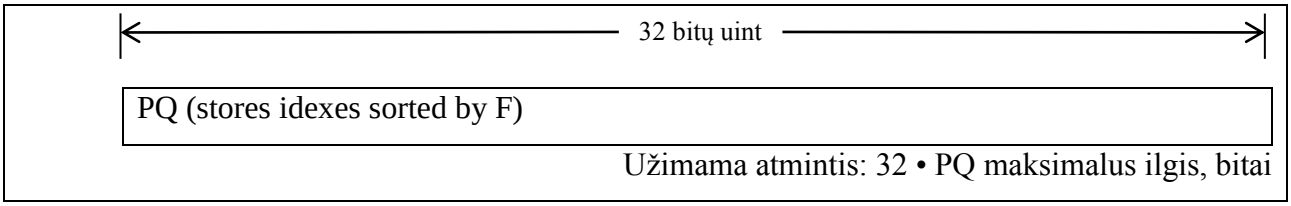
14 pav. „GPU v2“ algoritmo rezultatų gražinimui naudojamas atminties struktūros elementas.

Žemiau yra pateikta „GPU v3“ algoritmo naudojama atminties duomenų struktūrų architektūra, kurioje nėra naudojamas beveik joks išėjimo arba laikinų duomenų kintamųjų suspaudimas. Todėl viršūnės elemento plotis išauga nuo 3 iki 5 *uint* kintamųjų, o tai žymiai padidina atminties sąnaudas, tačiau dar labiau pakelia algoritmo našumą (tyrimas nr.1 ir tyrimas

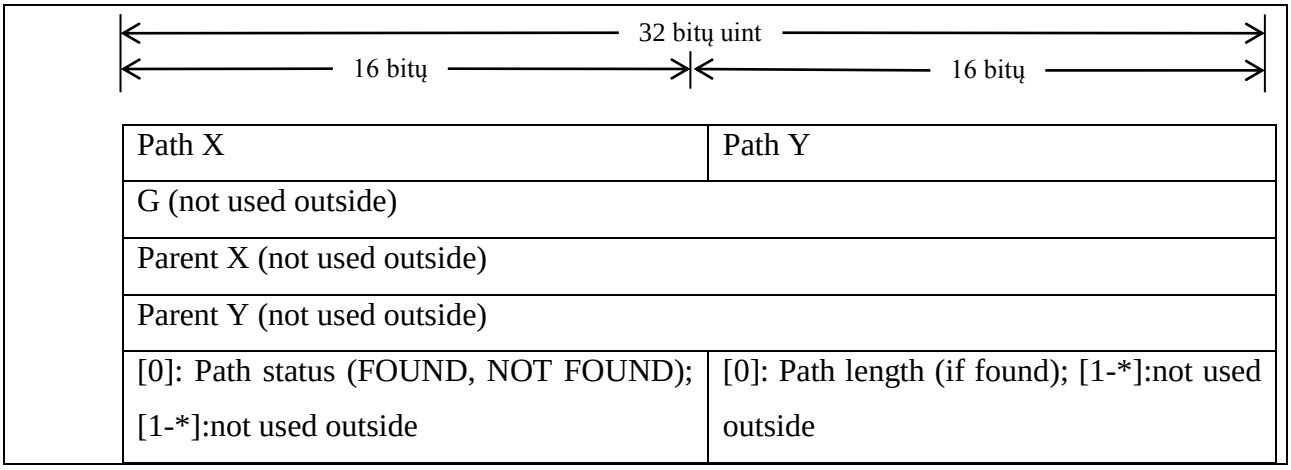
nr. 2). Lyginant su „GPU v1“ ir „GPU v2“ algoritmais, informacija apie rasto kelio statusą bei jo ilgį yra saugoma ne 2, o 4 gijos zonos 0-io indekso elemente.



15 pav. „GPU v3“ algoritmo skaičiavimų vykdymui naudojamas struktūros elementas.



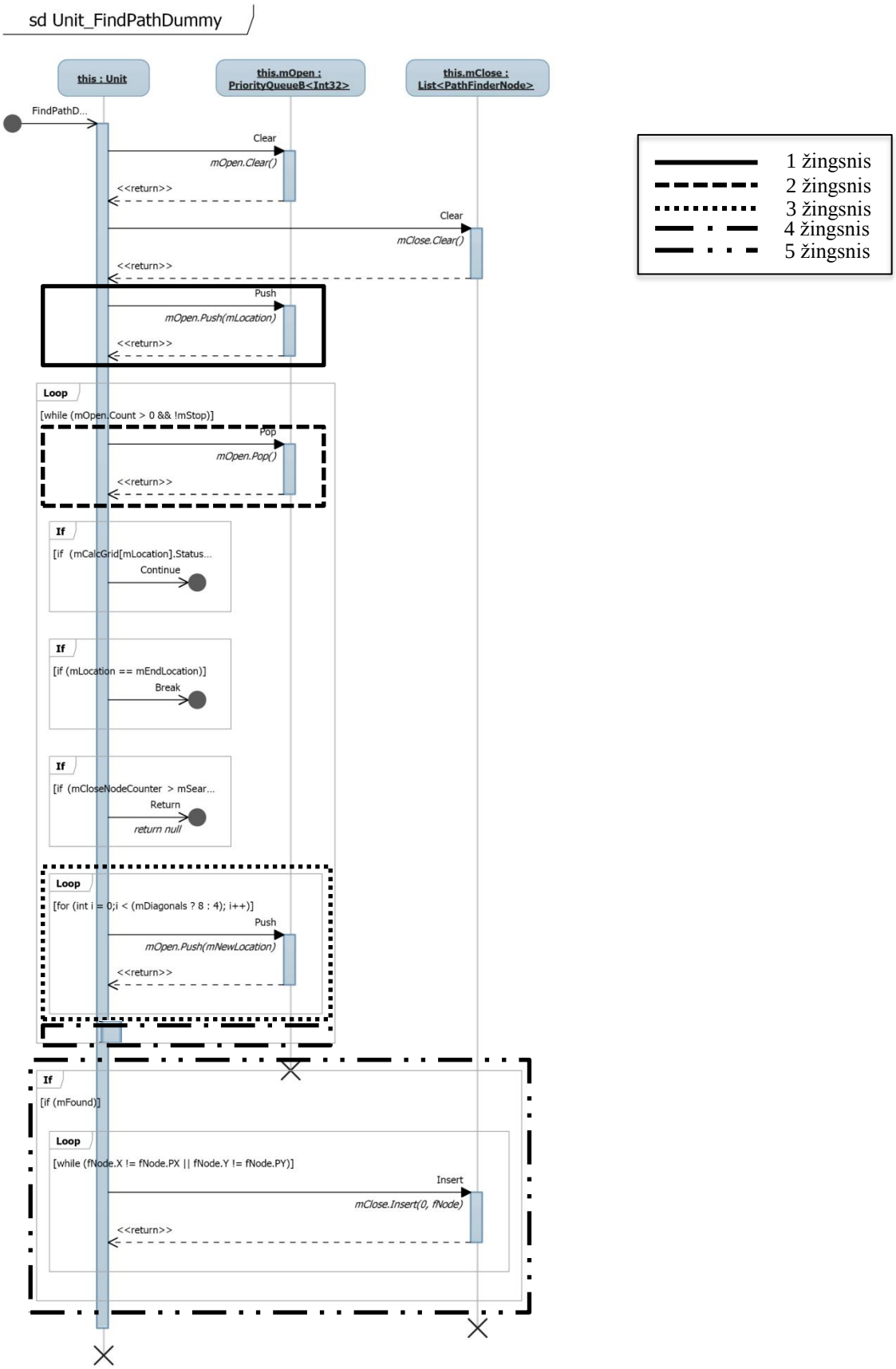
16 pav. „GPU v3“ algoritme naudojamas laikinos struktūros elementas.



17 pav. „GPU v3“ algoritmo rezultatų gražinimui naudojamas atminties struktūros elementas.

4.4. Algoritmo žingsnių realizacijos detalizavimas

Kelio paieškos A* algoritmo realizacijos apibendrinta sekų diagrama pavaizduota 18 paveiksle. Sekančiuose skyreliuose bus smulkiau aprašomi visi algoritmo žingsniai.



18 pav. Kelio paieškos A* algoritmo sekų diagrama.

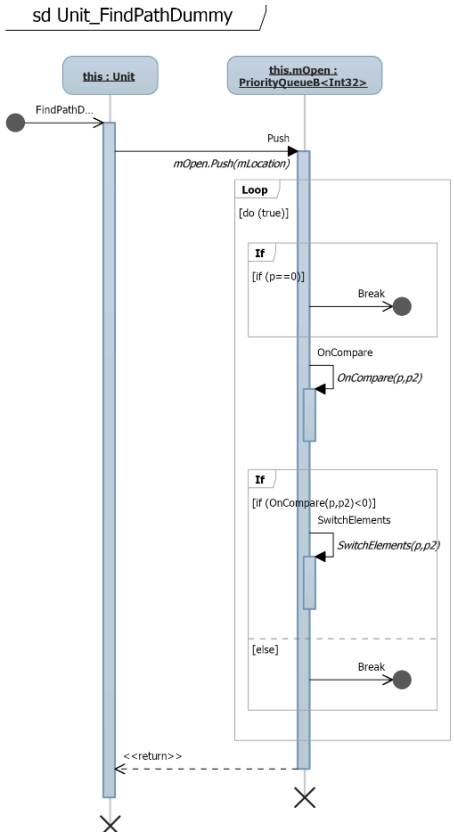
Sekų diagramos viršuje esantys „*mOpen.Clear()*“ ir „*mClose.Clear()*“ kreipiniai atitinkamai išvalo saugomus AVS ir UVS.

„While“ ciklo viduje po viršūnės su mažiausia bendro kelio svorio įverčio reikšme esantys sąlyginiai „*if*“ sakiniai atlieka tokius veiksmus (aprašomi eilės tvarka):

- Tikrina ar paimta viršūnė nėra UVS sąraše, kas reikštų, jog ji jau buvo išnagrinėta. Tokiu atveju yra praleidžiama esama *„while“* ciklo iteracija ir yra pereinama prie sekančios AVS viršūnės.
- Tikrina ar paimta viršūnė nėra lygi ieškomo kelio pabaigos taškui, jeigu taip – *„while“* ciklas yra nutraukiamas.
- Tikrinama ar išnagrinėtų viršūnių kiekis (vykdomos paieškos gylis) neviršija nustatytąjį maksimalų dydį. Jeigu viršija – *„while“* ciklas yra nutraukiamas taip ir neradus ieškomo kelio.

4.4.1. Žingsnis 1. „Pradinės viršūnės pridėjimas į AVS“

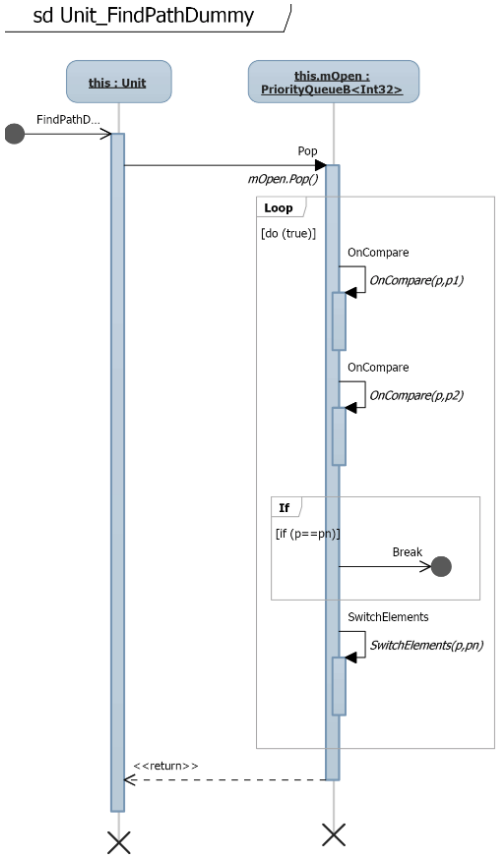
Pradine viršūne yra laikoma ieškomo kelio pradžios viršūnė. Ji yra įdedama į AVS ir nuo jos prasidės visas A* algoritmo kelio paieškos procesas.



19 pav. Kelio paieškos A* algoritmo detali 1 žingsnio sekų diagrama.

4.4.2. Žingsnis 2. „Viršūnės su mažiausiu bendru kelio įverčiu paėmimas iš AVS“

Visos viršūnės, kurios yra saugomos AVS, yra išrikiuotos bendro kelio įverčio reikšmės mažėjimo tvarka. Šiame žingsnyje yra paimama viršūnė su mažiausia bendro kelio įverčio reikšme. Ši viršūnė yra išimama iš AVS, o visos nuo 1 iki *n* indeksus turinčios AVS viršūnės yra paslenkamos per vieną vietą kairėn. Šios viršūnės statusas pasikeičia į atvirą.



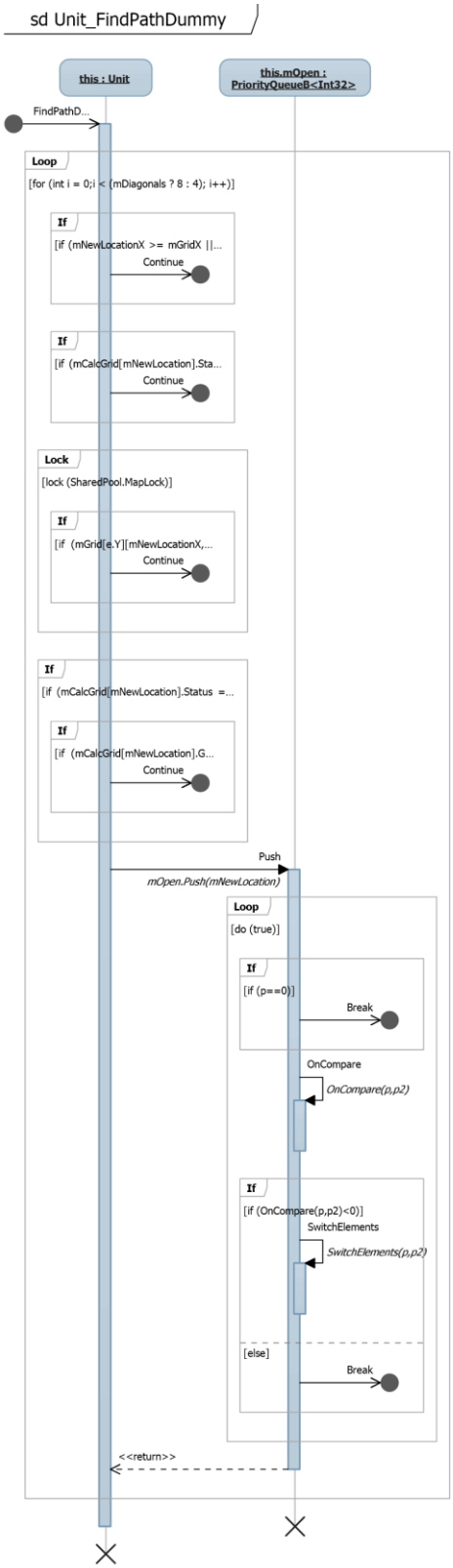
20 pav. Kelio paieškos A* algoritmo detali 2 žingsnio sekų diagrama.

4.4.3. Žingsnis 3. „Gretimų viršūnių ikėlimas į AVS“

Šiame žingsnyje yra analizuojamos visos tiriamosios viršūnės gretimos viršūnės. Jeigu jos nėra UVS, jas galima pasiekti ir jų esamo kelio įverčio reikšmė (kelio iki tų viršūnių kaina) yra didesnė už tiriamos viršūnės kelio įverčio reikšmę – tos viršūnės yra įkeliamos į AVS pagal joms paskaičiuotą bendro kelio įverčio reikšmę (kelio iki viršūnės kaina su pridėtu euristiniu įverčiu). Visų naujai įkeliamų viršūnių statusas tampa atviru, taip pat yra atžymima, kad į šią viršūnę buvo patekta iš tiriamosios viršūnės (Įsimenamos tėvinės viršūnės pozicijos reikšmės „*ParentX*“, „*ParentY*“).

Įkėlimas į AVS yra vykdomas atsižvelgus į viršūnės bendro kelio įverčio reikšmę, kuria remiantis yra randama viršūnės vieta AVS sąraše. Iš esmės AVS struktūra atitinka surikiuotą

sąrašą. Viršūnės įterpimas perkelia visas viršūnes su indeksu, didesniu nei įterpiamos viršūnės, per vieną elementą į dešinę.



21 pav. Kelio paieškos A* algoritmo detali 3 žingsnio sekų diagrama.

4.4.4. Žingsnis 4. „Viršūnės perkėlimas į UVS“

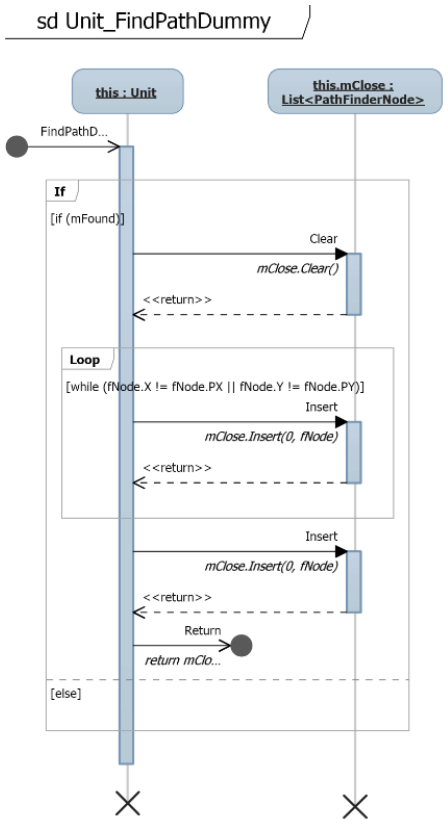
Kai visos gretimos tiriamajai viršūnės yra išnagrinėtos – tiriamoji viršūnė yra laikoma ištirta ir ji yra nukeliama į UVS. Kadangi UVS yra realizuotas ne kaip atskiras sąrašas, o kaip viršūnių informacijos matricos dalis, tai čia pakanka paprasčiausia pakeisti šios viršūnės statusą į uždarytą.

```
mCalcGrid[mNewLocation].Status = mOpenNodeValue;
```

22 pav. Kelio paieškos A* algoritmo 4 žingsnio realizacija.

4.4.5. Žingsnis 5. „Kelio formavimas“

Šiame žingsnyje yra suformuojamas kelias pagal viršūnių informacijos matricoje saugomą informaciją. Kelias yra formuojamas pradedant ieškomo kelio pabaigos viršūne ir yra sudaromas pereinant į viršūnę, iš kurios buvo ateita į šią viršūnę (naudojami kintamieji *ParentX*, *ParentY*). Atitinkamai joje yra pereinama į viršūnę, iš kurios buvo ateita į tą viršūnę ir t.t. Procesas yra kartojamas kol nėra pasiekiama ieškomo kelio pradžios viršūnė. Jeigu kelias nebuvo rastas – šitas žingsnis nėra vykdomas.



23 pav. Kelio paieškos A* algoritmo detali 5 žingsnio sekų diagrama.

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 vykdymo laiką. Visi šiame darbe pateikti laiko matavimai yra išskaičiuojami kaip penkių to paties matavimo paleidimų vidurkis. 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.

Šiame darbe yra nutarta vykdyti tokius tyrimus:

1. Kelio paieškos A* algoritmų vykdymo laiko priklausomybė nuo užklausų kiekio paprastame žemėlapyje (žemėlapis aprašytas skyriuje 5.3. Tyrimui naudojamos įrangos bei parametrų aprašas).
2. Kelio paieškos A* algoritmų vykdymo laiko priklausomybė nuo užklausų kiekio sudėtingame žemėlapyje. (žemėlapis aprašytas skyriuje 5.3. Tyrimui naudojamos įrangos bei parametrų aprašas).
3. Duomenų perkėlimo iš CPU į GPU ir iš CPU į GPU atminties buferius laiko priklausomybė nuo užklausų kiekio.
4. Gijų grupės dydžio įtaka kelio paieškos A* algoritmo našumui.
5. Euristinės funkcijos įtaka kelio paieškos vykdomo laikui.
6. DirectX technologijos galimybių lygio įtaka kelio paieškos algoritmų našumui.

5.2. Pasirinktų tyrimų paskirtis

2 lentelė. Pasirinktų tyrimo metodų aprašas ir paskirtis

Tyrimo numeris	**Paskirtis**
1	Nustatyti kelio paieškos algoritmo A* ir jo modifikacijų vykdymo laiko priklausomybę nuo užklausų kiekio paprastame žemėlapyje. Bendru būdu ištirti kelio paieškos algoritmo veikimą skirtingų paskirčių procesoriuose. Parodyti, kad grafinis procesorius gali būti sėkmingai naudojamas kelio paieškos algoritmo A* spartinimui. Kiekybiškai palyginti vykdymo laikus skirtingų paskirčių procesoriuose.
2	Nustatyti kelio paieškos algoritmo A* ir jo modifikacijų vykdymo laiko

Tyrimo numeris	Paskirtis
	priklausomybę nuo užklausų kiekio sudėtingame žemėlapyje. Įsitikinti, kad žemėlapio sudėtingumas daro stiprią įtaką paieškos algoritmo vykdymo laikui.
3	Nustatyti duomenų perkėlimo iš CPU į GPU atminties buferius laiko priklausomybę nuo užklausų kiekio. Įvertinti perkeliamų duomenų kiekio priklausomybę nuo algoritmo vykdymo laiko.
4	Nustatyti duomenų perkėlimo iš CPU į GPU atminties buferius laiko priklausomybę nuo užklausų kiekio. Įvertinti perkeliamų duomenų kiekio priklausomybę nuo algoritmo vykdymo laiko.
5	Nustatyti gijų grupės dydžio įtaką GPU tipo kelio paieškos algoritmuose. Kiekybiškai įvertinti teisingai pasirinkto gijų grupės dydžio suteikiamą naudą. Nustatyti bendras tendencijas tarp gijų grupės dydžio ir užklausų kiekio.
6	Nustatyti euristinės funkcijos įtaką kelio paieškos vykdymo laikui. Palyginti įvairių euristinių funkcijų įtaką tiek CPU, tiek GPU tipo kelio paieškos algoritmams. Rasti efektyviausią euristinę funkciją. Kiekybiškai įvertinti teisingai pasirinktos euristinės funkcijos suteikiamą naudą.
7	Ištirti DirectX technologijos galimybių lygio įtaką kelio paieškos algoritmo našumui.

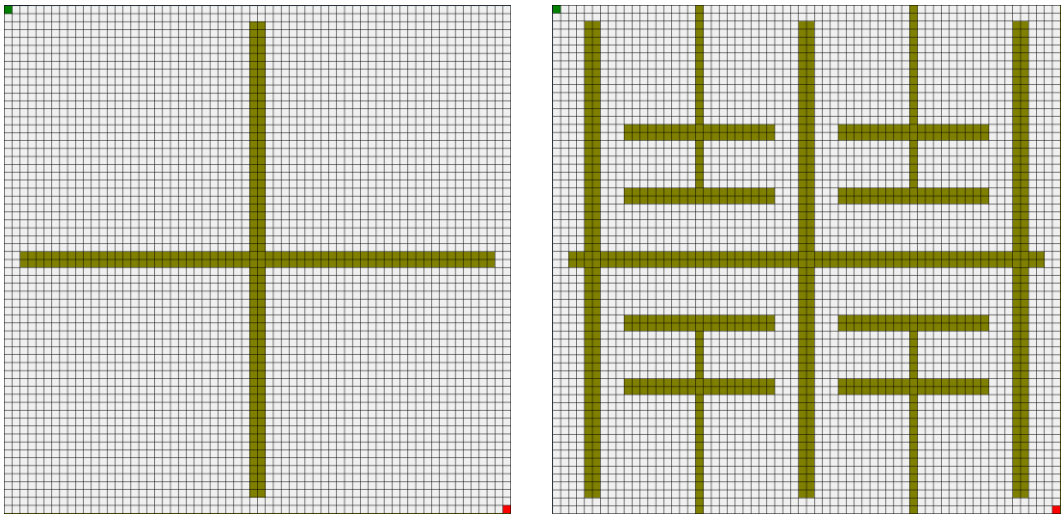
5.3. Tyrimui naudojamos įrangos bei parametrų aprašas

Tyrimui yra naudojama tokia aparatūrinė bei programinė įranga:

- Centrinis procesorius: Intel Core2 Duo E8500 @3.5 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.

Norint atlikti tyrimus, aprašytus skyriuje 5.2. Pasirinktų tyrimų paskirtis, buvo realizuotos dvi CPU kelio paieškos algoritmo ir trys GPU kelio paieškos algoritmo versijos. „CPU v1“ yra paprasta kelio paieškos algoritmo A* versija. „CPU v2“ yra šiek tiek optimizuota to paties paieškos algoritmo versija. „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.

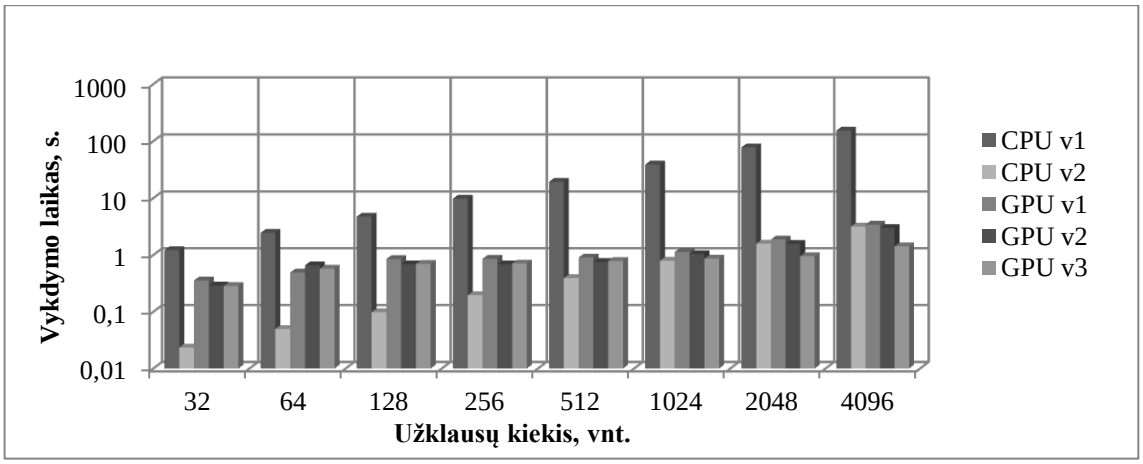


24 pav. Kelio paieškos A* algoritmo tyrime naudojami paprastas (kairėje) ir sudėtingas (dešinėje) žemėlapiai.

24 paveiksle pavaizduoti du žemėlapiai. Atliekant kelio paiešką šie žemėlapiai yra interpretuojami kaip 64x64 viršūnių dydžio grafai. Paveiksle laisvi langeliai balti, neperžengiami langeliai pažymėti tamsia spalva. Trumpiausio kelio paieška yra vykdoma iš (0,0) viršūnės į (63,63) viršūnę. Paveiksluose 25 – 32 yra pateikti tyrimų rezultatai atliekant skirtingus kelio paieškos užklausų kiekius šiuose žemėlapiuose.

5.4. Tyrimo rezultatai

5.4.1. Tyrimas nr. 1



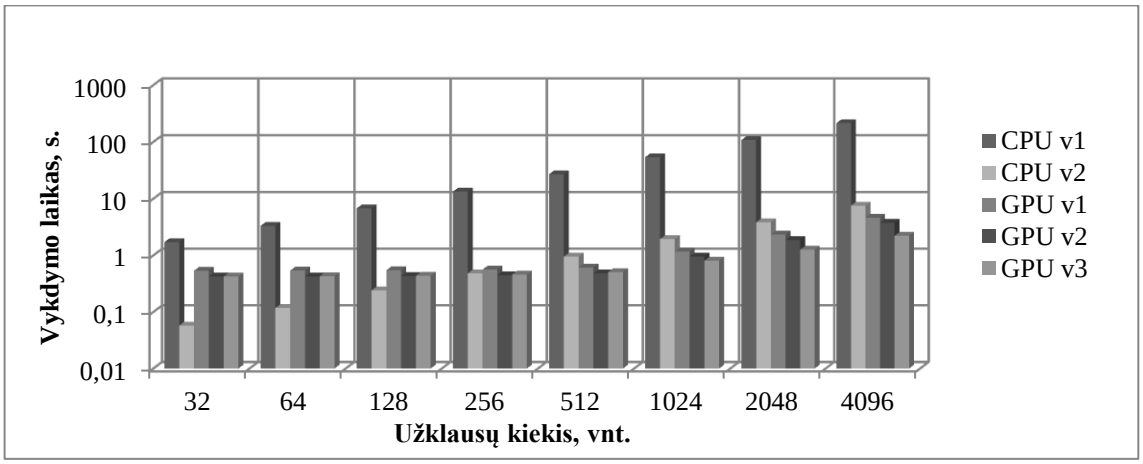
25 pav. A* kelio paieškos algoritmo versijų vykdymo laikų palyginimas paprastame žemėlapyje.

25 paveiksle yra parodytas realizuotų dviejų CPU bei trijų GPU kelio paieškos A* algoritmų versijų skaičiavimų vykdymo laikų palyginimas paprastame žemėlapyje.

Neoptimizuota A* algoritmo versija „CPU v1“ akivaizdžiai dirba lėčiau nei visos kitos algoritmo versijos. Optimizuota „CPU v2“ algoritmo versija dirba greičiau nei GPU algoritmo versijos, kai užklausų kiekis yra tarp 32 ir 1024. Palyginus „CPU v1“ ir „CPU v2“ algoritmų skaičiavimų atlikimo laikus gauname, kad „CPU v2“ dirba ~486 kartus greičiau.

„GPU v3“ algoritmo atveju vienos grafo viršūnės informacijai saugoti reikia 20 baitų atminties. Kintamųjų informacijos suspaudimo dėka šias sąnaudas sumažiname iki 12 baitų vienai viršūnei „GPU v1“ algoritmo atveju. Tačiau tai neigiamai įtakoja našumą. Lyginant visų optimizuotų algoritmų skaičiavimų atlikimo laikus, gauname ~3,52 kartų didesnį našumą naudojant GPU vietoje CPU („CPU v2“) 4096 paieškos užklausų atveju.

5.4.2. Tyrimas nr. 2



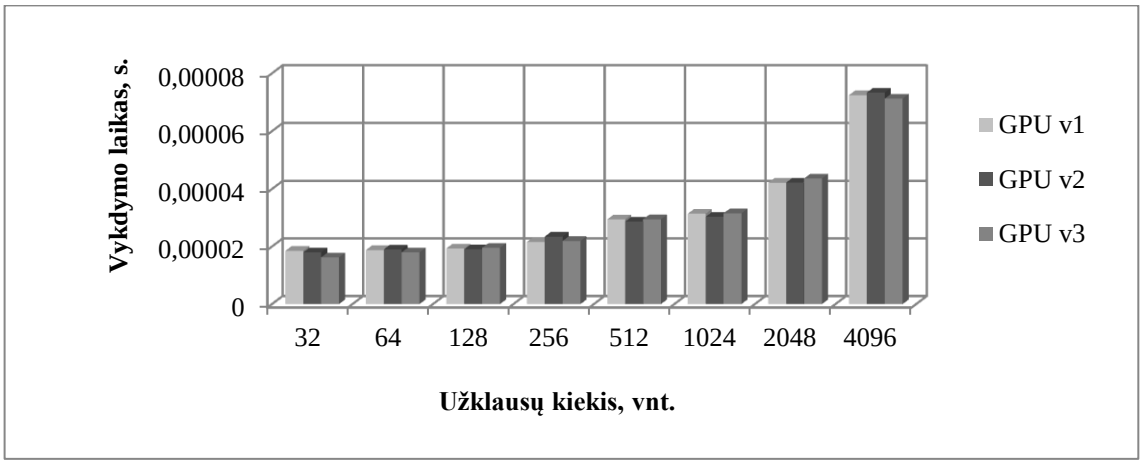
26 pav. A* kelio paieškos algoritmo versijų vykdymo laikų palyginimas sudėtingame žemėlapyje.

26 paveiksle yra parodytas realizuotų dviejų CPU bei trijų GPU kelio paieškos A* algoritmų versijų skaičiavimų vykdymo laikų palyginimas sudėtingame žemėlapyje.

Neoptimizuota A* algoritmo versija „CPU v1“ akivaizdžiai dirba lėčiau nei visos kitos algoritmo versijos. Optimizuota „CPU v2“ algoritmo versija dirba greičiau nei GPU algoritmo versijos kai užklausų kiekis yra tarp 32 ir 128. Palyginus „CPU v1“ ir „CPU v2“ algoritmų skaičiavimų atlikimo laikus gauname, kad „CPU v2“ dirba tik ~28 kartus greičiau.

Palyginus visų optimizuotų algoritmų skaičiavimų atlikimo laikus gauname ~3,40 kartų didesnį našumą naudojant GPU vietoje CPU („CPU v2“) 4096 paieškos užklausų atveju. Palyginus šiuos rezultatus su 1 tyrimo metu gautais rezultatais matome, kad esant sudėtingai kelio paieškos problemai GPU tipo algoritmai veikia sparčiau prie mažesnių užklausų kiekių nei paprasto žemėlapio atveju.

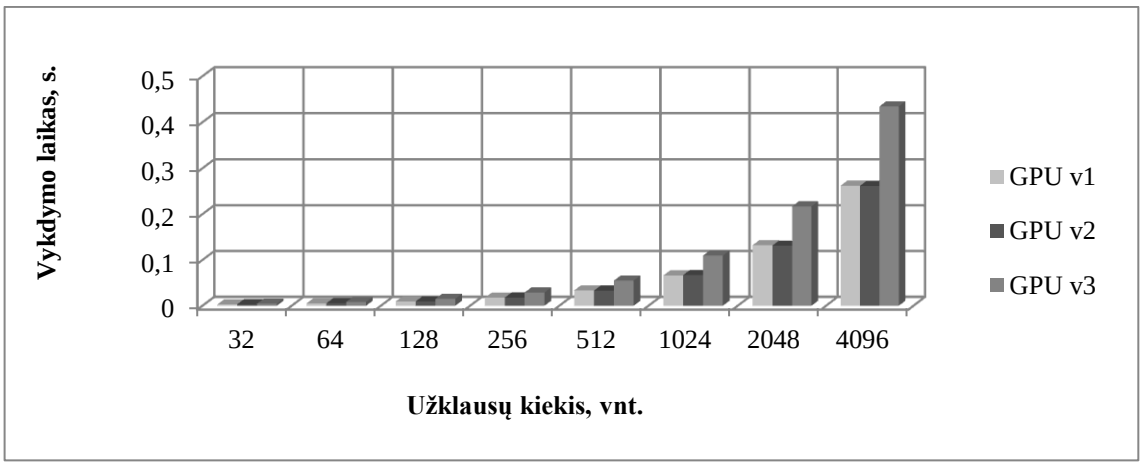
5.4.3. Tyrimas nr. 3



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

27 paveiksle yra atvaizduotas duomenų perkėlimo iš CPU į GPU atminties buferius laikas. Kaip matome, bendra laiko augimo, augant perkeliamų duomenų kiekiui, tendencija išlieka pastovi. Visais GPU pusėje vykdomų algoritmų atvejais užtrunkamas panašus laikas.

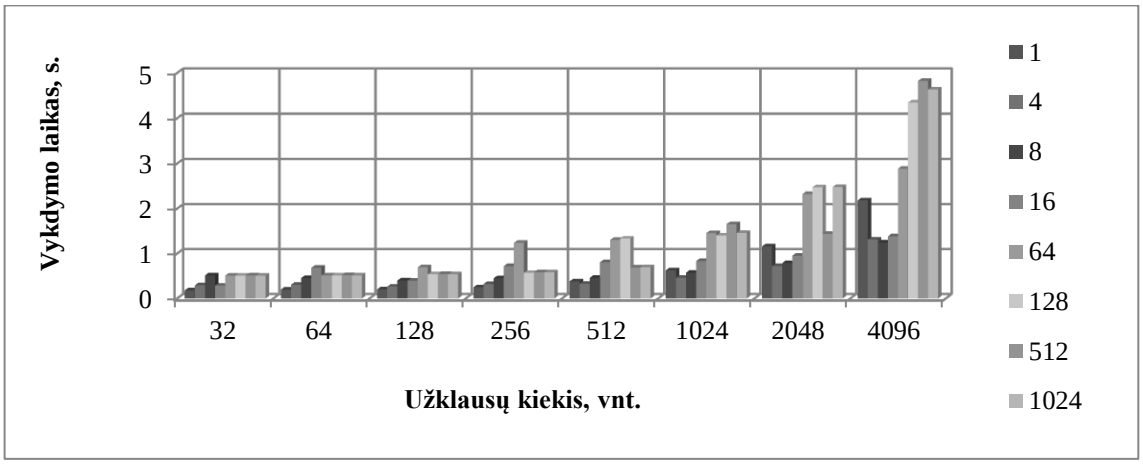
5.4.4. Tyrimas nr. 4



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

28 paveiksle yra pateiktas duomenų perkėlimo iš GPU į CPU atminties buferių laikas. Duomenų perkėlimo operacijos laikas yra pastebimai priklausomas nuo perkeliamos atminties dydžio. Esant tam pačiam užklausų kiekiui, „GPU v3“ algoritmo atveju perkeliamas duomenų kiekis yra ~1,67 kartų didesnis lyginant su kitais GPU algoritmais, tuo pačiu laiko sąnaudos išauga ~1,66 kartus. Todėl galima daryti prielaidą, kad perkeliamos atminties dydžio ir išaugusių laiko sąnaudų priklausomybė yra tiesinė.

5.4.5. Tyrimas nr. 5



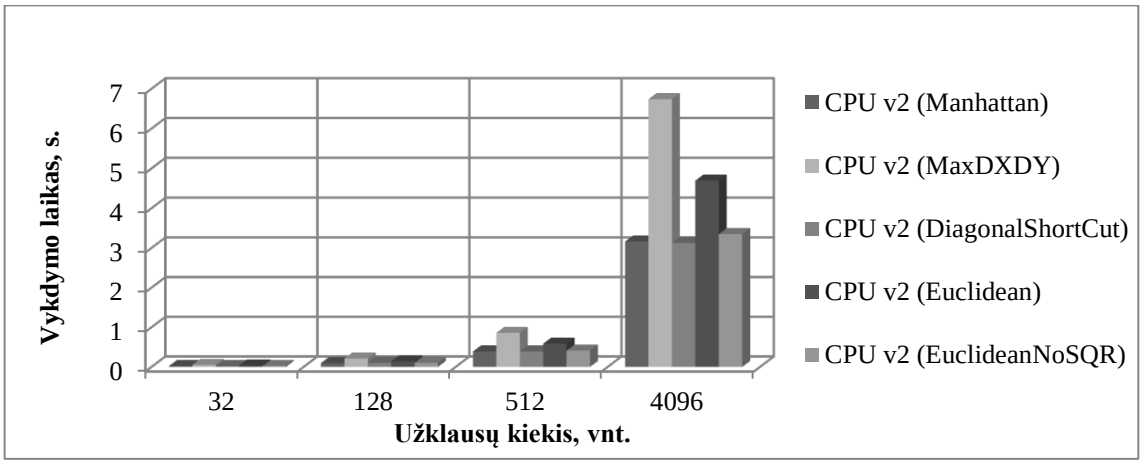
29 pav. Gijų grupės dydžio įtaka A* kelio paieškos algoritmo našumui.

29 paveiksle yra parodytas algoritmo „GPU v3“ vykdymas keičiant gijų grupės dydį. 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. Svarbu pabrėžti, kad 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.

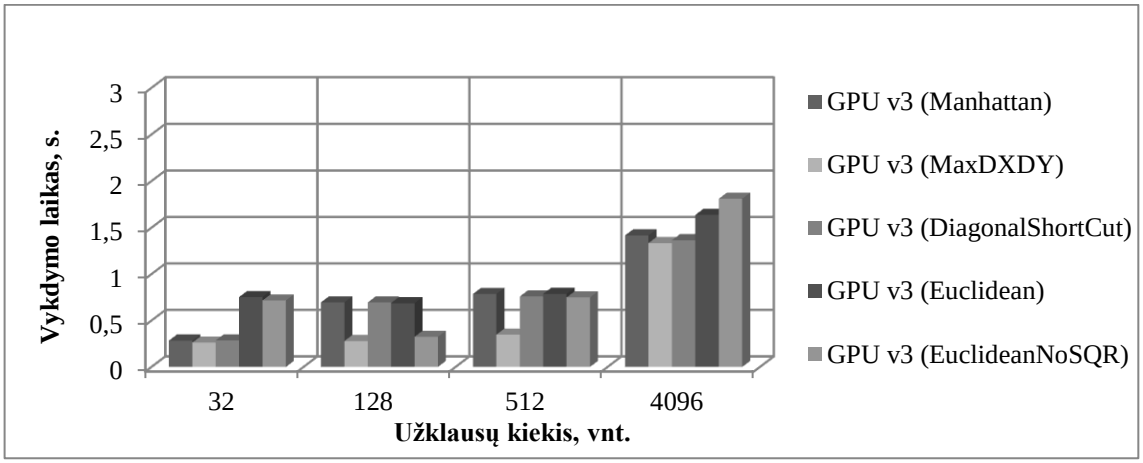
Matome, kad teisingas gijų grupės dydžio pasirinkimas sprendžiant kelio paieškos problemą paprastame žemėlapyje, leido paspartinti algoritmo vykdymo laiką ~3,72 kartus 4096 užklausų atveju ir net ~4,99 kartus 256 užklausų atveju.

Iš gautų rezultatų taip pat matome, kad surinkti duomenys nerodo stiprios tiesioginės priklausomybės tarp vykdomų užklausų kiekio ir gijų grupės dydžio. Tačiau galime įžvelgti tendenciją, kad didėjant vykdomų užklausų kiekiui geriau yra rinktis didesnę gijų grupę. Todėl geriausias gijų grupės dydis kelio paieškos algoritmo vykdymo metu turėtų būti parenkamas dinamiškai, priklausomai nuo vykdomų užklausų kiekio.

5.4.6. Tyrimas nr. 6



30 pav. Euristinės funkcijos įtaka „CPU v2“ kelio paieškos A* algoritmo vykdymo laikui.



31 pav. Euristinės funkcijos įtaka „GPU v3“ kelio paieškos A* algoritmo vykdymo laikui.

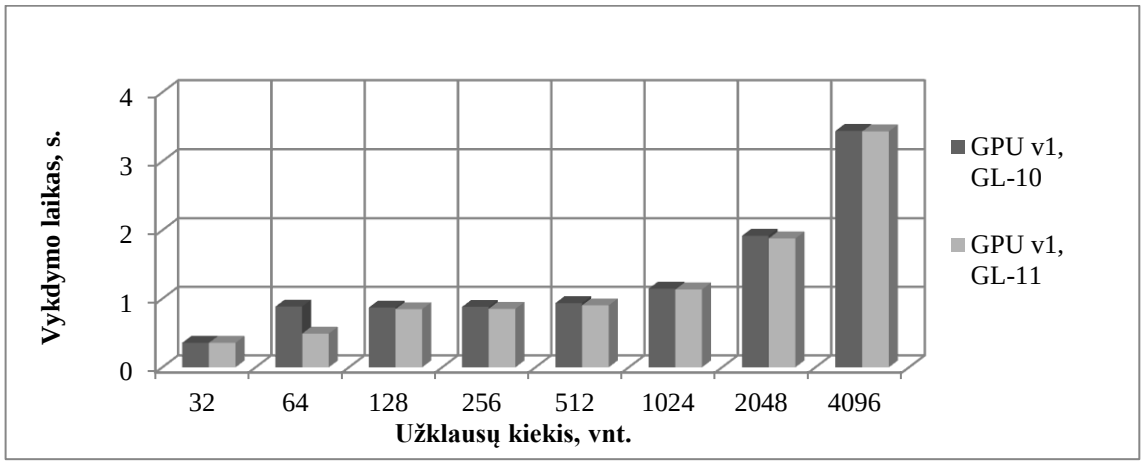
Iš 30 ir 31 paveikslų matome, kad net ir paprastame žemėlapyje euristinės funkcijos parinkimas turi didelę įtaką. Kelio paieškos A* algoritmo „CPU v2“ atveju geriausius rezultatus parodė *Manhattan* euristinė funkcija, o prasčiausius *MaxDXDY*. Taip pat matome, kad CPU tipo algoritmo atveju vykdymų užklausų kiekis neturi įtakos bendrai euristinių funkcijų įtakai, nes visais užklausų kiekio atvejais matoma tokia pati vykdymo laiko didėjimo tendencija. Palyginus rezultatus šiuo atveju gauname, kad teisingos euristinės funkcijos parinkimas paspartina algoritmą ~2,13 kartus.

Greičiausiu GPU tipo algoritmo „GPU v3: vykdymo laiko rezultatai buvo kiek kitokie. Geriausius rezultatus parodė *Manhattan* euristinė funkcija, tačiau prasčiausius *EuclideanNoSQR* esant 4096 užklausų kiekiui. Iš paveikslo akivaizdžiai matome, kad GPU algoritmų atveju

užklausų kiekis taip pat turi nemažą įtaką algoritmo vykdymo laikui, todėl paskaičiuoti kiekybinio našumo pagerėjimo GPU algoritmo atveju nepavyks.

Iš aptartų rezultatų galime teigti, kad net pakankamai paprasto kelio paieškos metu yra itin svarbu tinkamai parinkti euristinę funkciją. Taip pat euristinės funkcijos parinkimas turėtų būti vykdomas dinamiškai, prisitaikant prie konkretaus žemėlapio.

5.4.7. Tyrimas nr. 7



32 pav. DirectX technologijos galimybių lygio (GL) įtaka kelio paieškos A* algoritmo našumui.

32 paveiksle pateikta DirectX technologijos funkcijų lygio įtaka GPU tipo algoritmui „GPU v1“. Kaip matome iš paveikslo DirectX funkcijų lygio įtaka yra nepriklausoma nuo užklausų kiekio ir beveik visais atvejais matoma šiokia tokia nauda. Reiktų atkreipti dėmesį, kad esant 64 užklausų kiekiui toks gautas rezultatas gali būti dėl matavimo metu kartu suveikusių išorinių sistemos procesų, visiškai nesusijusių su matuojama sistema.

Vidutiniškai „GPU v1“ algoritmas paspartėja ~1,005 kartus. Tai yra itin mažas prieaugis prie bendro algoritmo našumo, tačiau reiktų atkreipti dėmesį, kad šis našumo prieaugis atsiranda iš naudojamos technologijos patobulinimo, o ne iš A* algoritmo realizacijos subtilybių.

6. IŠVADOS

1. Išanalizavus kelio paieškos problemą paaiškėjo, kad ją galima apibrėžti keturiais komponentais: pradine būsena, galimų veiksmų sąrašu, tikslo patikrinimo funkcija ir kelio svorio funkcija. Šie komponentai apjungti į vientisą duomenų struktūrą yra panaudojami kaip kelio paieškos problemos sprendimo algoritmo įėjimo duomenys.
2. Nustatyta, kad kelio paieškos problema gali būti sprendžiama dviem būdais: agentinėmis sistemomis ir fizinės sąveikos sistemomis. Agentinės sistemos pasižymi kelio paieškos radimo unikalumu (gali būti atsižvelgiama į matomumą, minios judėjimo taisykles, kiekvieno dalyvio psichologinę būseną ir t.t.), tačiau kelio radimas naudojant agentines sistemas yra sudėtingas ir daug skaičiavimų reikalaujantis procesas. Fizinės sąveikos sistemos pateikia realybę labiau atitinkančius rezultatus (minios grūsčių vengimas, natūralaus kelio pasirinkimas, sklandus judėjimas ir t.t.), tačiau skaičiavimų metu dažnai netenkama unikalumo.
3. Kelio paieškos algoritmo A* ir jo modifikacijų vykdymo laiko tyrimas (tyrimas nr. 1) parodė, kad optimizuotas kelio paieškos A* algoritmas pačiu geriausiu atveju vykdomas ~486 kartus greičiau nei neoptimizuota algoritmo versija.
4. Detalesnė tyrimo nr. 1 rezultatų analizė parodė, kad sprendžiant paprastą kelio paieškos problemą optimizuotas kelio paieškos A* algoritmas vykdomas CPU pusėje dirba greičiau nei sukurti GPU pusėje veikiantys algoritmai, kai užklausų kiekis yra mažesnis nei 2048. Užklausų kiekiui esant didesniam nei 2048, greičiau vykdomas GPU pusėje dirbantis algoritmas. Įvertinus bendrus paieškos laikus, gauname ~3,52 kartų didesnį algoritmo našumą naudojant GPU vietoje CPU 4096 paieškos užklausų atveju.
5. Paieškos žemėlapio sudėtingumo įtakos tyrimas (tyrimas nr. 2) parodė, kad GPU pusėje veikiantys algoritmai veikia sąlyginai sparčiau sprendžiant sudėtingą kelio paieškos problemą nei paprastą. Algoritmų pranašumai pastebimi, kai užklausų kiekis ima viršyti 128 užklausas.
6. Duomenų perkėlimo iš CPU į GPU tyrimas (tyrimas nr. 3) parodė, kad duomenų perkėlimo iš CPU į GPU atminties buferius operacijos trukmės didėjimo tendencija išlieka pastovi didėjant užklausų kiekiui. Duomenų perkėlimo iš GPU į CPU tyrimas (tyrimas nr. 4) parodė, kad duomenų perkėlimo iš GPU į CPU atminties buferius

operacijos trukmė yra tiesiškai priklausoma nuo perkeliamos atminties dydžio. Esant tam pačiam užklausų kiekiui, atminties netaupantis GPU pusėje veikiantis algoritmas dirba ~1,66 kartus lėčiau, tačiau jo vykdymo metu yra perkeliamas ~1,67 kartų didesnis duomenų kiekis.

7. Gijų grupės dydžio įtakos algoritmų vykdymo laikui tyrimas (tyrimas nr. 5) parodė, kad gijų grupių dydis turi didelę įtaką GPU pusėje vykdomų algoritmų našumui. Optimalus gijų grupės dydis pritaikytas prie paleidžiamų gijų grupių kiekio leidžia geriau išnaudoti bei pilniau užpildyti turimus aparatūrinius grafinio įrenginio resursus. Teisingas gijų grupės dydžio pasirinkimas sprendžiant kelio paieškos problemą leido paspartinti algoritmo vykdymo laiką ~3,72 kartus 4096 užklausų atveju ir net ~4,99 kartus 256 užklausų atveju.
8. Euristinės funkcijos įtakos algoritmų vykdymo laikui tyrimas (tyrimas nr. 6) parodė, kad kelio paieškos geriausius rezultatus gavome naudodami euristinę funkciją *Manhattan*, prasčiausius – *MaxDXDY*. Net ir pakankamai paprasto kelio paieškos metu yra itin svarbu tinkamai parinkti euristinę funkciją. Euristinės funkcijos parinkimas turėtų būti vykdomas dinamiškai pritaikant ją prie konkrečios paieškos problemos. Teisingos euristinės funkcijos parinkimas leido paspartinti CPU pusėje vykdomą algoritmą ~2,13 kartus.
9. DirectX technologijos galimybių lygio įtakos tyrimas (tyrimas nr. 7) parodė, kad DirectX galimybių lygis turi labai mažą įtaką GPU pusėje vykdomų kelio paieškos A* algoritmų vykdymo laikui. Vidutiniškai algoritmas paspartėja tik ~1,005 kartus.

7. PADĖKOS

Norime išreikšti padėkas:

- Vadovui Tomui Blažauskui už konsultacijas ruošiant magistrinį darbą.
- Kauno Technologijos Universiteto, Programų inžinerijos katedros profesoriui V. Štuikiui už pateiktas išsamias žinias ir metodikas, kurios padėjo paruoštį šį magistrinį darbą.
- Pietų Kalifornijos Universiteto, Informatikos katedros profesoriui S. Koenig už savo svetainėje, adresu http://idm-lab.org, talpinamų straipsnių medžiagą ir pagalbą gilinantis į agentinių sistemų teoriją.

8. LITERATŪROS SARAŠAS

[1] **AMD**. AMD App Acceleration. *Advanced Micro Devices, Inc.* 2011 [žiūrėta 2011.05.10]. 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 2011.05.11]. Prieiga per internetą: http://ecn.channel9.msdn.com/o9/pdc09/ppt/CL03.pptx

[3] **Brogran D. C., Hodgins J. K.** Group behaviors for systems with significant dynamics. *In Autonomous Robots*. 1997, p. 137–153.

[4] **Cebenoyan C., Thibieroz N.** DirectCompute Performance on DX11 Hardware. *GDC 2010, NVIDIA, AMD.* 2010 [žiūrėta 2011.05.05]. Prieiga per internetą: http://developer.amd.com/gpu_assets/DirectCompute%20Performance.zip

[5] **Chenney, S.** Flow tiles. *In 2004 ACM SIGGRAPH / Eurographics Symposiumon Computer Animation*. 2004, p. 233–242.

[6] **Clements R. R., Hughes R. L.** Mathematical modelling of a mediaeval battle: the battle of Agincourt, 1415. *Math. Comput. Simul. 64, 2.* 2004, p. 259–269.

[7] **Colombo R. M., Rosini M. D.** Pedestrian flows and nonclassicalshocks. *Mathematical Methods in the Applied Sciences 28, 13*. 2005.

[8] **Cordeiro O. C., Braun A., Silveira C. B., Musse S. R., Cavalheiro G. G. H.** Concurrency on social forces simulation model. *First International Workshop on Crowd Simulation*. 2005.

[9] **Franklin S. Graesser A.** Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages.* Springer-Verlag, 1996.

[10] **Frigioni D., Marchetti-Spaccamela A., Nanni U.** Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms.* 2000, p. 251– 281.

[11] **Funge J., Tu X., Terzopoulos D.** Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *In Proceedings of SIGGRAPH 99, Computer Graphics Proceedings, Annual Conference Series*. 1999, p. 29–38.

[12] **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.* 1986, p. 100–107.

[13] **Helbing D., Bunza L., Werner T.** Self-organized pedestrian crowd dynamics and design solutions. *Traffic Forum 12*. 2003.

[14] **Helbing D., Molnar P., Schweitzer F.** Computer simulations of pedestrian dynamics and trail formation. *In Evolution of Natural Structures*. 1994, p. 229–234.

[15] **Hongwan L., Wai F. K., Chor C. H**. A study of pedestrian flow using fluid dynamics. *Tech. Rep*. 2003.

[16] **Hughes R. L.** The flow of human crowds. *Annu. Rev. Fluid Mech*. 2003, p. 169–182

[17] **Hughes R. L.** A continuum theory for the flow of pedestrians. *Transportation Research Part B 36 , 6*. 2002.

[18] **KHRONOS**. OpenCL Overview. *KHRONOS Group.* 2011 [žiūrėta 2011.05.03]. Prieiga per internetą: http://www.khronos.org/opencl/

[19] **Koenig S., Likhachev M.** A new principle for incremental heuristic search: Theoretical results. *In Proceedings of the International Conference on Automated Planning and Scheduling*. 2006, 402–405, 2006.

[20] **Koenig S., Likhachev M., Furcy D.** Lifelong Planning A*. *Artificial Intelligence Journal, 155*. 2004, p. 93-146.

[21] **Koenig S., Likhachev M., Liu Y., Furcy D.** Incremental heuristic search in artificial intelligence. *Artificial Intelligence Magazine*. 2004, p. 99–112.

[22] **Luger F., Stubblefield W.** *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. The Benjamin/Cummings Publishing Company, Inc. 2004.

[23] **MSDN**. HLSL. *Microsoft Corporation.* 2011 [žiūrėta 2011.05.06]. Prieiga per internetą: http://msdn.microsoft.com/en-us/library/bb509561%28v=vs.85%29.aspx

[24] **MSDN.** ID3D11DEVICECONTEXT::DISPATCH METHOD. *Microsoft Corporation*. 2011 [žiūrėta 2011.05.06]. Prieiga per internetą: http://msdn.microsoft.com/en-us/library/ff476405%28v=vs.85%29.aspx

[25] **Musse S. R., Thalmann D.** A model of human crowd behavior: Group inter-relationship and collision detection analysis. *In Computer Animation and Simulation '97*. 1997, p. 39–51.

[26] **Nilsson N.** *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers. 1998.

[27] **NVIDIA**. What is CUDA?. *NVIDIA Corporation.* 2011 [žiūrėta 2011.05.08]. Prieiga per internetą: http://www.nvidia.com/object/what_is_cuda_new.html

[28**] Pelechano N., Obrien, K., Silverman B., Badler, N.** Crowd simulation incorporating agent psychological models, roles and communication.*First International Workshop on Crowd Simulation*. 2005.

[29] **Peng C., Ruihua X., Xiaolei Z.** A Modified Heuristic Search Algorithm for Pedestrian Simulation. *Stanford University*. 2010 [žiūrėta 2011.05.15] Prieiga per internetą:

http://eil.stanford.edu/pengao/Papers/A%20Modified%20Heuristic%20Search%20Algorithm%20for%20Pedestrian%20Simulation.pdf

[30] **Reynolds C. W.** Flocks, herds, and schools: A distributed behavioral model. *In Computer Graphics (Proceedings of SIGGRAPH 87)*. 1987, vol. 21, p. 25–34.

[31] **Robin**. What is heuristic search?. *Aftificial intelligence.* 2011 [žiūrėta 2011.05.01]. Prieiga per internetą: http://intelligence.worldofcomputing.net/ai-search/heuristic-search.html

[32] **Shao W., Terzopoulos D.** Autonomous pedestrians. *In SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. New York, 2005, p. 19–28.

[33] **Skadron K**. *Thrends in Multicore Architecture*. ASPLOS '08. 2008 [žiūrėta 2011.05.08]. Prieiga per internetą: http://gpgpu.org/static/asplos2008/ASPLOS08-2-manycore.pdf

[34] **Sun X., Yeoh W., Koenig S.** Dynamic Fringe-Saving A*. *In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS).*2009, p. 891-898

[35] **Sung M., Gleicher M., Chenney, S.** Scalable behaviors for crowd simulation. *Computer Graphics Forum 23, 3*. 2004, p. 519–528.

[36] **Thomas H. C., Charles E. L., Rivest. R. L., Stein C.** *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001. Chapter 1: Foundations, p. 3–122.

9. PRIEDAI

9.1. Publikuotas straipsnis

Šis straipsnis buvo publikuotas leidinyje *Informacinės technologijos. XVI tarpuniversitetinė magistrantų ir doktorantų konferencija*, Technologija, 2011. p. 69 – 72.

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:

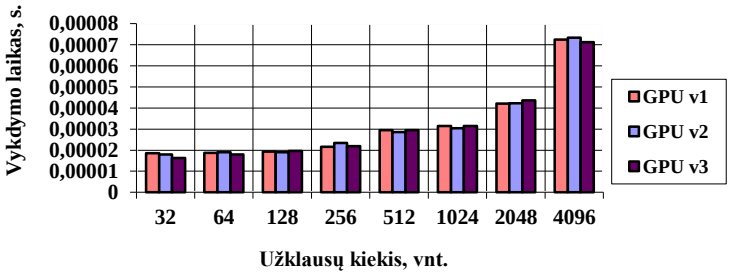
- Centrinis procesorius: Intel Core2 Duo E8400 @3.2 GHz.
- Operatyvioji atmintis: Kingston HyperX DDR2 4 GB @854 MHz.
- Grafinis procesorius: Gigabyte 450 GTS OC2.
- Operacinė sistema: Microsoft Windows 7 x64.

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.

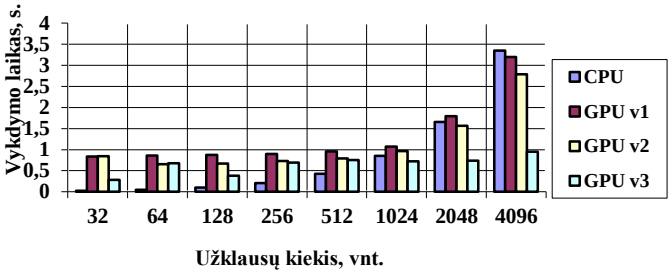
1 – 4 paveiksluose yra pateikti rezultatai atliekant skirtingus paieškos užklausų kiekius 64x64 viršūnių dydžio grafe. Grafo viduryje yra suformuota kliūtis, paliekanti tik siaurus praėjimus grafo kraštuose. Trumpiausio kelio paieška yra vykdoma iš (0,0) viršūnės į (63,63) viršūnę. Gijų grupės dydis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



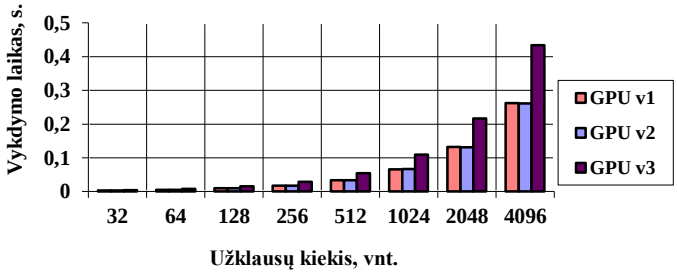
1 pav. Duomenų perkėlimo iš CPU į GPU atminties buferių laikas A* kelio paieškos algoritmo vykdymo metu.

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.



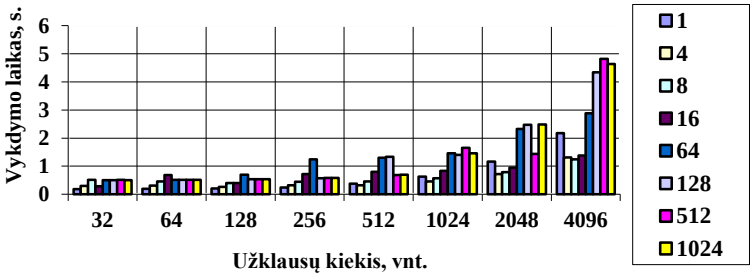
2 pav. A* kelio paieškos algoritmo versijų vykdymo laikų palyginimas.

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].

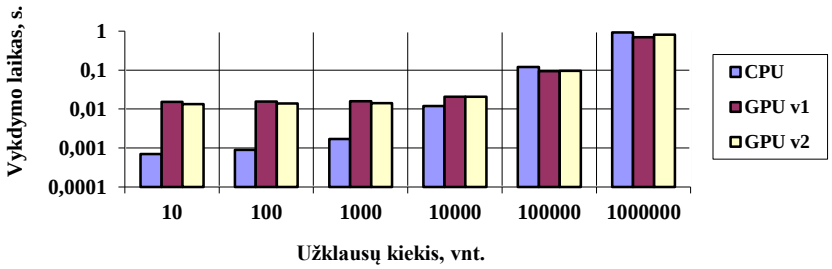


4 pav. Gijų grupės dydžio įtaka A* kelio paieškos algoritmo našumui.

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.