VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Eismo dalyvių kelyje atpažinimas naudojant dirbtinius neuroninius tinklus ir grafikos procesorių

On-road vehicle recognition using neural networks and graphics processing unit

Magistro baigiamasis darbas

Atliko:	Povilas Kinderis	(parašas)
Darbo vadovas:	Mindaugas Eglinskas	(parašas)
Recenzentas:	prof. dr. Romas Baronas	(parašas)

Vilnius – 2011

Santrauka lietuvių kalba

Kasmet daugybė žmonių būna sužalojami autoįvykiuose, iš kurių dalis sužalojimų būna rimti arba pasibaigia mirtimi. Dedama vis daugiau pastangų kuriant įvairias sistemas, kurios padėtų mažinti nelaimių skaičių kelyje. Tokios sistemos gebėtų perspėti vairuotojus apie galimus pavojus, atpažindamos eismo dalyvius ir sekdamos jų padėtį kelyje. Eismo dalyvių kelyje atpažinimas iš vaizdo yra pakankamai sudėtinga, daug skaičiavimų reikalaujanti problema. Šiame darbe šiai problemai spręsti pasitelkti stereo vaizdai, nesugretinamumo žemėlapis bei konvoliuciniai neuroniniai tinklai. Konvoliuciniai neuroniniai tinklai reikalauja daug skaičiavimų, todėl jie optimizuoti pasitelkus grafikos procesorių ir OpenCL. Gautas iki 33,4% spartos pagerėjimas lyginant su centriniu procesoriumi. Stereo vaizdai ir nesugretinamumo žemėlapis leidžia atmesti didelius kadro regionus, kurių nereikia klasifikuoti su konvoliuciniu neuroniniu tinklu. Priklausomai nuo scenos vaizde, reikalingų klasifikavimo operacijų skaičius sumažėja vidutiniškai apie 70-95% ir tai leidžia kadrą apdoroti atitinkamai greičiau.

Raktiniai žodžiai: eismo dalyvių atpažinimas, konvoliucinis neuroninis tinklas, grafinis procesorius, GP, OpenCL, OpenCV, stereo vaizdas, nesugretinamumo žemėlapis.

Santrauka anglų kalba (Summary)

Many people are injured during auto accidents each year, some injures are serious or end in death. Many efforts are being put in developing various systems, which could help to reduce accidents on the road. Such systems could warn drivers of a potential danger, while recognizing on-road vehicles and tracking their position on the road. On-road vehicle recognition on image is a complex and computationally very intensive problem. In this paper, to solve this problem, stereo images, disparity map and convolutional neural networks are used. Convolutional neural networks are very computational intensive, so to optimize it GPU and OpenCL are used. 33.4% speed improvement was achieved compared to the central processor. Stereo images and disparity map allows to discard large areas of the image, which are not needed to be classified using convolutional neural networks. Depending on the scene of the image, the number of the required classification operations decreases on average by 70-95% and this allows to process the image accordingly faster.

Keywords: on-road vehicle recognition, convolutional neural network, graphics processing unit, GPU, OpenCL, OpenCV, stereo image, disparity map.

TURINYS

ĮVADAS

Vidutiniškai kas minutę žmogus žūna autoavarijoje. Per metus mažiausiai 10 milijonų žmonių būna sužalojami autoįvykiuose, iš kurių 2-3 milijonai žmonių sužalojimų būna rimti. Spėjama, kad dėl šių priežasčių patiriamos išlaidos, įskaitant ligoninių sąskaitas, sugadintą nuosavybę ir kitas išlaidas, gali sudaryti 1-3 procentus pasaulio bendrojo vidaus produkto [SBM06]. Dedama vis daugiau pastangų kuriant įvairias sistemas, kurios padėtų mažinti nelaimių skaičių kelyje. Todėl intelektualios navigacinės ir automatinės kliūčių atpažinimo sistemos įvairiose transporto priemonėse susilaukia vis daugiau dėmesio. Tokios sistemos gebėtų perspėti vairuotojus apie galimus pavojus, atpažindamos eismo dalyvius (pėsčiuosius, motociklininkus, mašinas, sunkvežimius) ir sekdamos jų padėtį kelyje.

Eismo dalyvių kelyje atpažinimas analizuojant vaizdą yra pakankamai sudėtinga problema. Todėl per pastaruosius du dešimtmečius ši problema susilaukė nemažai dėmesio ir yra aktyvus tyrimų objektas [SMB06a], nes dėl autoįvykių yra prarandamas stulbinantis skaičius gyvybių bei prarandama daug finansų ir yra sukaupta nemažai technologijų kompiuterinės regos tyrimuose. Taip pat viena iš priežasčių yra didelis procesorių galingumo augimas (tiek centrinio procesoriaus, tiek grafikos procesoriaus), įgalinantis daug skaičiavimų reikalaujančias video apdorojimo užduotis atlikti kur kas greičiau nei bet kada anksčiau.

Eismo dalyvių kelyje atpažinimo problemai įveikti galime pasinaudoti dviejų kamerų video srautu bei konvoliuciniais neuroniniais tinklais. Dviejų kamerų video srauto reikia norint nustatyti vaizdo gylį, t.y. kuris objektas yra toliau, kuris arčiau. Tą pasiekti galime keliais būdais. Vienas iš dažniausiai sutinkamų būdų yra dvi lygiagrečiai filmuoti sukalibruotos kameros, tam tikru atstumu viena nuo kitos (analogija – žmogaus akys). Šios kameros galėtų būti įmontuotos transporto priemonės priekinėje dalyje, patogioje filmuoti vietoje. Tačiau yra nemažai kliūčių, tokių kaip: videokamerų vibracija važiuojant keliu, netobulai sukalibruotos videokameros ar įvairios sudėtingos scenos. Daugelio kalibravimo problemų galime išvengti panaudodami specialius lęšius, kurie gebėtų fokusuoti stereo vaizdą ant vienos kameros matricos. Tuomet vietoje dviejų videokamerų užtektų vienos ir nereikėtų rūpintis ar kameros yra lygiagrečios, ar filmuoja tuo pačiu kampu, ar matricų parametrai yra vienodi. Taip pat viena iš problemų yra video srauto apdorojimo greitis šiuolaikiniuose kompiuteriuose. Nors šiuolaikiniai centriniai procesoriai (CP) yra pakankamai pajėgūs atlikti įvairius uždavinius, tačiau jie nėra pakankamai greiti apdoroti video srautą realiu laiku (bent 15 kadrų per sekundę dažniu), atpažįstant jame eismo dalyvius. Taip pat norint naudoti didesnes videokamerų rezoliucijas geresniems rezultatams gauti, atitinkamai reikia ir daugiau skaičiuojamosios galios. Dėl šių priežasčių gali atsirasti eismo dalyvių kelyje atpažinimo vėlavimas, kas nėra priimtina, norint šią informaciją

gauti realiu laiku važiuojant keliu. Viena iš alternatyvų vaizdo apdorojimo paspartinimui yra grafiniai procesoriai (GP). Jei dalis skaičiavimų būtų perkelta į GP, tuomet CP nebūtų apkrautas ir galėtų atlikti kitas užduotis ar pasiruošti sekančio kadro apdorojimui.

Pastaraisiais metais grafiniai procesoriai (GP) susilaukia nemažai dėmesio ne vien iš kompiuterinių žaidimų entuziastų. Šį procesorių taip pat galima pritaikyti įvairiems bendro pobūdžio uždaviniams spręsti, jei sprendžiamas uždavinys yra tinkamas GP architektūrai ir uždaviniui tinka GP veikimo pobūdis. Lyginant su centriniu procesoriumi (CP), galime pasiekti kelis, keliasdešimt ar net kelis šimtus kartų didesnį efektyvumą, nes GP gali lygiagrečiai atlikti daug daugiau slankaus kablelio operacijų. Tačiau ne visų uždavinių sprendimai gali būti paspartinti pasitelkus šį procesorių [LH07], taip pat yra įvairių ribojimų bei sunkumų, tokių kaip: pradinių duomenų perkėlimo į GP operatyviąją atmintį greitis, GP skirtų programavimo priemonių nepakankamas išsivystymas [OHL+08], klaidų sekimo įrankių trūkumas. GP panaudojimas bendro pobūdžio uždaviniams spręsti yra pakankamai naujas būdas spartinti skaičiavimus ir vis dar vystosi bei tobulėja – gimsta naujos technologijos, įvairūs standartai ir būdai, kaip galime programuoti pasitelkus GP, kaip aptikti ir taisyti klaidas.

Yra siūloma daug įvairių metodų, kaip būtų galima atpažinti, klasifikuoti paveikslėlius. Tačiau vienas iš moderniausių, labiausiai pripažintų ir taikomų metodų yra konvoliucinių neuroninių tinklų panaudojimas. Šie neuroniniai tinklai yra taikomi rašmenų, veidų ar tam tikrų objektų atpažinimui. Šiuos neuroninius tinklus galime pritaikyti ir eismo dalyvių atpažinimui [CH07]. Tačiau neuroninis tinklas gali būti labai didelis – su daug neuronų ir jungčių, todėl suskaičiuoti neuroninio tinklo rezultatus gali prireikti daug resursų. Konvoliucinių neuroninių tinklų veikimo spartinimui galima pasitelkti GP.

Daugelyje nagrinėtų šaltinių pabrėžiamos techninės įrangos problemos ir nepakankamai greitai veikiantys metodai. Akivaizdu, kad šios problemos dar labiau išryškėja bandant apdoroti didesnės rezoliucijos video srautą ar vaizdus. Nebuvo pastebėti mėginimai spartinti eismo dalyvių atpažinimą su GP. Tačiau buvo mėginama atpažinti eismo dalyvius naudojantis konvoliuciniais neuroniniais tinklais [CH07], kurių skaičiavimas buvo spartinamas su GP [SKP10]. Pastarajame šaltinyje buvo pabrėžta, kad yra dedama per mažai pastangų, bandant paspartinti konvoliucinius neuroninius tinklus su GP, todėl buvo pristatytas karkasas, skirtas spartinti konvoliucinius neuroninius tinklus su GP. Tačiau šis karkasas buvo įgyvendintas su NVIDIA CUDA technologija, todėl yra prisirišama tik prie NVIDIA techninės įrangos. Atsiradus tokiems standartams kaip OpenCL, atsirado galimybė pritaikyti konvoliucinius neuroninius tinklus (ir paspartinti eismo dalyvių atpažinimą) ne vien NVIDIA GP, bet ir kitokiems procesoriams, tokiems kaip AMD GP ar Cell Broadband Engine procesoriams.

Pagrindinė problema, kuri sprendžiama magistriniame darbe, yra eismo dalyvių atpažinimo spartinimas su GP, naudojantis konvoliuciniais neuroniniais tinklais bei spartinant atpažinimo procesą, naudojantis OpenCL standartu, kuris suteikia galimybę būti nepriklausomam nuo techninės įrangos, su kuria yra vykdoma OpenCL programa.

Šio magistrinio darbo tikslas yra pasiūlyti metodą bei konvoliucinio neuroninio tinklo modelį eismo dalyvių kelyje atpažinimui iš vaizdo, procesą optimizuojant pasitelkus nesugretinamumo žemėlapį bei GP.

Šio magistrinio darbo uždaviniai:

- Sukurti konvoliucinio neuroninio tinklo modelį eismo dalyviams atpažinti.
- Sukurti eismo dalyvių atpažinimo metodą pritaikius konvoliucinius neuroninius tinklus, nesugretinamumo žemėlapį bei GP.
- Atlikti greitaveikos analizę lyginant CP ir GP.

Sistemos prototipui sukurti buvo naudojamos C++, C#, OpenCL C programavimo kalbos ir Visual Studio 2010 IDE. Taip pat buvo naudojami .NET 4, Qt 4.7.1 karkasai bei OpenCV 2.2, EmguCV 2.2, OpenCL 1.0 bibliotekos. C#, .NET bei EmguCV buvo naudojami sukurti pagrindinę prototipo dalį, kuri buvo naudojama eksperimentavimo, neuroninio tinklo apmokymo, pradinio kadro apdorojimo tikslais. C++, Qt ir OpenCL buvo naudojami sistemos greitaveikos analizei atlikti, lyginant CP su GP. OpenCV biblioteka buvo naudojama pradiniam nesugretinamumo žemėlapio skaičiavimui.

Stereo vaizdams filmuoti buvo naudojamas Nikon D7000 fotoaparatas ir Loreo 3D Lens in a Cap 9005 objektyvas. Filmavimas buvo atliekamas automagistralėje Vilnius-Kaunas-Klaipėda. Fotoaparatas buvo pritvirtintas prie trikojo, kuris buvo pritvirtintas prieš priekinio keleivio sėdynę ir nukreiptas į priekinį mašinos langą.

Pirmajame skyriuje yra pateikta literatūros apžvalga. Pateiktas GP ir CP palyginimas, pristatytas OpenCL, pristatyti pagrindiniai eismo dalyvių atpažinimo būdai ir metodai (kurie buvo naudojami įvairiuose tyrimuose), pristatyta kaip vaizdo atpažinimo problemai spręsti buvo naudojami neuroniniai tinklai ir kaip buvo bandyta pritaikyti neuroninius tinklus grafiniam procesoriui.

Sekančiuose šio darbo skyriuose yra pateikiama šio darbo analitinė, projektavimo ir realizavimo dalis. Antrame skyriuje yra aprašomas neuroninio tinklo modelis bei struktūra. Yra aprašoma kaip neuroninis tinklas yra sukonstruojamas bei apmokomas. Aptariami neuroninio tinklo parametrai. Trečiame skyriuje yra aprašoma kaip yra gaunamas stereo vaizdas, kaip naudojamas nesugretinamumo žemėlapis išskirti potencialiems vaizdo regionams, kaip apdorojamas pradinis paveikslėlis prieš jį įvedant į neuroninį tinklą, kaip neuroninio tinklo rezultatų skaičiavimas yra perkeliamas į GP. Taip pat aprašoma, kaip yra apdorojami neuroninio

tinklo rezultatai bei kaip iš šių rezultatų yra nustatoma eismo dalyvio pozicija kadre. Pateikiamas greičių palyginimas tarp centrinio procesoriaus ir grafinio procesoriaus. Sekančiame skyriuje yra pateikiamos išvados bei rezultatai.

1. ŠALTINIŲ APŽVALGA

1.1. Grafinis procesorius

Grafiniai procesoriai (angl. *GPU, graphic processing unit*) pastaraisiais metais yra intensyviai tobulinami bei tampa kur kas galingesnės ir lankstesnės žaidimų, vizualizacijų ir efektų kūrime, pasiekdamos vis našesnių ir tikroviškesnių rezultatų kompiuteriniuose žaidimuose. Šiuolaikiniai GP leidžia grafikos programuotojams modifikuoti standartinę vaizdo apdorojimo eigą, įgalindamas juos perprogramuoti viršūnių (angl. *vertex shader*) bei fragmentų – pikselių (angl. *fragment shader*) apdorojimą. Senesni GP turi 2 tipų programuojamus modulius: viršūnių (angl. *vertex processing unit*) bei fragmentų (angl. *fragment processing unit*). Naujesniuose GP šie moduliai buvo apjungti į bendrus tiek viršūnes, tiek geometriją, tiek fragmentus apdorojančius modulius [LH07, ND10]. Pirmasis personaliniams kompiuteriams skirtas unifikuotas grafinis procesorius NVIDIA GeForce 8800 buvo pristatytas 2006 metais [ND10]. Modulių apjungimas lemia dar didesnį lankstumą bei suteikia daugiau laisvės grafikos programuotojams. Naujausi grafikos procesoriai, palaikantys DirectX 10/11 technologiją, taip pat leidžia perprogramuoti geometrijos apdorojimą (angl. *geometry shader*).

Šie unifikuoti moduliai gali vykdyti vartotojo nurodytus instrukcijų rinkinius: viršūnių apdorojimo programas (angl. *vertex programs*) kiekvienai viršūnei apdoroti, geometrijos apdorojimo programas (angl. *geometry programs*) objektų geometrijai apdoroti, fragmentų apdorojimo programas (angl. *fragment programs*) kiekvienam pikseliui apdoroti bei skaičiavimo programas (angl. *compute programs*) [LH07]. Viršūnių apdorojimo programos gali pasiekti geometrijos ir atributinius duomenis (spalvą, tekstūros koordinates ir pan.), kurie būna patalpinti GP atmintyje, per vaizdavimo sąrašus (angl. *display list*) arba naujesniuose GP – per viršūnių buferio objektus (angl. *VBO, vertex buffer objects*). Fragmentų programos gali pasiekti 32 bitų tikslumo slankaus kablelio duomenis, kurie gali būti perduoti per tekstūrą, o naujesniuose GP galima apdoroti ir 64 bitų tikslumo slankaus kablelio duomenis. Paprastai fragmentų programų išvestis keliauja į kadro buferį (angl. *frame buffer*), tačiau naujesniuose GP šią išvestį galima nukreipti į slankaus kablelio tekstūrą (pasitelkus piešimo į tekstūrą techniką), pasinaudojus kadro buferio objektais (angl. *FBO, frame buffer objects*). Šia GP savybe galima pasinaudoti suskaičiuojant bet kokius bendresnio pobūdžio uždavinius: galima naudoti vieną tekstūrą kaip duomenis ir rezultatus patalpinti į kitą, kuri gali būti vėliau panaudojama sekančiuose skaičiavimo etapuose. Tačiau šis būdas reikalauja specifinių OpenGL ar DirectX žinių, taip pat reikia išmanyti bent vieną šešėliavimo (angl. *shading*) kalbą. Tai sukelia programuotojams papildomų problemų, kurie anksčiau su šiomis technologijomis nebuvo susidūrę.

Esant tiek slankaus kablelio skaičiavimų potencialo GP procesoriuose, natūralu, kad norėtųsi tuo pasinaudoti ir paspartinti savo programas, jei tai įmanoma. Todėl atsiranda įvairios technologijos, specifikacijos, kurios ganėtinai supaprastina GP programavimą, ir nebelieka poreikio išmanyti OpenGL ar DirectX bei šešėliavimo kalbos. Vienos populiariausių šiuo metu yra Khronos Group išleista OpenCL specifikacija [Khr10b] ir NVIDIA CUDA architektūra. Taip pat atsirado ir Microsoft DirectCompute API kartu su DirectX 11, kuri veikia ir su DirectX 10 palaikančiomis vaizdo plokštėmis. Šios GP programavimo priemonės nereikalauja daug papildomų žinių, norint pradėti programuoti GP, lyginant su klasikiniu būdu, naudojant šešėliavimo programas. Taip pat naujausi GP procesoriai palaiko tikslesnius – 64 bitų slankaus kablelio skaičius. Į šių dienų GP galima žiūrėti kaip į duomenų lygiagrečiuosius procesorius, kurių skaičiavimo galia gerokai lenkia CP bei yra specializuoti tam tikros srities skaičiavimuose [LH07].

1.1.1. CP prieš GP. Kuris greitesnis atliekant daug skaičiavimų lygiagrečiai?

1 lentelėje pateikti šiuo metu rinkoje esančių pačių galingiausių GP charakteristikos. Kaip matome, šie procesoriai pasižymi iš ties labai didele skaičiavimo galia. Palyginimui, Intel i7 860 centrinis procesorius gali pasiekti 89.6 GFLOPS viengubo tikslumo skaičiavimo galią [SKP10]. Tai yra net apie 30 kartų mažiau nei ATI HD 5870 grafinis procesorius.

Kur slypi GP galios paslaptis? Nors šiuolaikiniai CP taktinis dažnis yra išties nemažas, tačiau branduolių skaičius nėra toks didelis lyginant su GP. Grafinių procesorių ALU (angl. *arithmetic logic unit*) modulių taktinis dažnis nors ir yra mažesnis, tačiau GP turi šių modulių kur kas daugiau nei CP. Tačiau šie įspūdingi skaičiai atspindi tik teorinę maksimalią galią ir norint ją pasiekti, reikėtų, kad kiekvienas GP srauto multiprocesorius vykdytų visas MUL ir MAD operacijas vienu metu [SKP10].

1 lentelė. Šiuo metu rinkoje esančių greičiausių GP parametrai [SVS10, Amd10]

Grafikos procesorius	GTX 480	HD 5870
ALU taktinis dažnis (MHZ)	1401	850
ALU skaičius	480	1600
Skaičiavimo galia (viengubo tikslumo, GFLOPS)	1344.96	2720
Skaičiavimo galia (dvigubo tikslumo, GFLOPS)	672.48	544

Įdomią diskusiją CP ir GP greičių palyginimo tema galima rasti [Dom07]. Autorius atliko mažą tyrimą: sugeneravo 100 Bezier kreivių. Iš pradžių ši scena yra piešiama naudojantis CPU (Intel Pentium M 750, 1.86GHz, 533MHz FSB) cairo grafinėje aplinkoje, skaičiuojamas laikas. Po to – pasinaudojus GPU (ATI X300, 64MB RAM) viršūnių apdorojimo programa (Vertex

Shader). Rezultatai pateikti 2 lentelėje. Matome, kad GP susidorojo su užduotimi pakankamai neblogai.

2 lentelė. CPU prieš GPU skaičiavimo greičiai [Dom07]

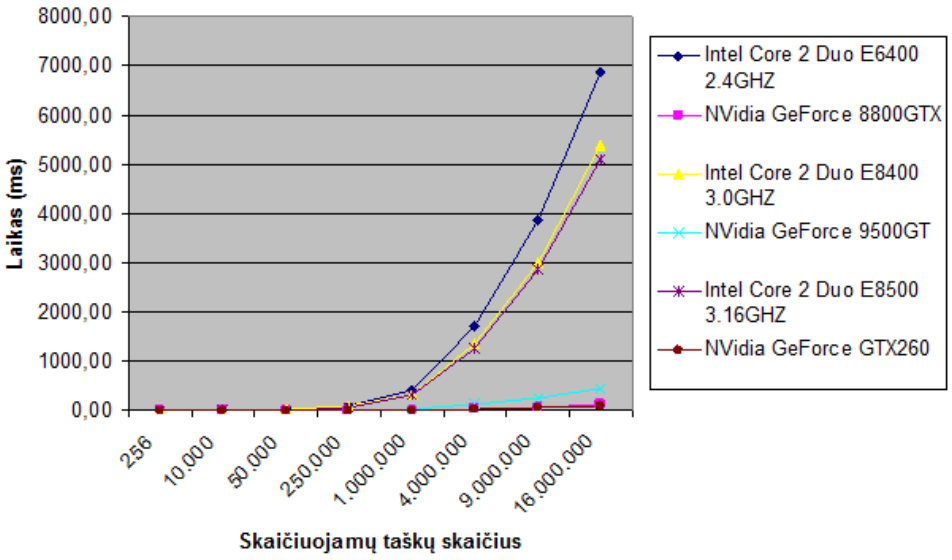
Skaičiavimo aplinka	Skaičiavimo ir atvaizdavimo laikas
CPU, GTK+ cairo x surface	0.2202 s
CPU, GTK+ cairo img. surface	0.1546 s
GPU, OpenGL CG vertex shader	0.0074 s

Šis bandymas su Bezier kreivėmis buvo pakartotas ir su kita sistema [KK09]. Buvo naudojamas 2400MHZ greičiu veikiantis Intel Core 2 Duo E6600 dviejų branduolių centrinis procesorius bei NVIDIA 8800GTX grafikos apdorojimo posistemis. Bandymas buvo atliktas su 100, 1000 bei 10000 kreivėmis. Rezultatai pateikti 3 lentelėje. NVIDIA 8800GTX procesorius skaičiavimus atliko vidutiniškai 750 kartų greičiau nei Intel Core 2 Duo E6600 procesoriaus vienas branduolys.

3 lentelė. CPU prieš GPU testo rezultatai [KK09]

Skaičiavimo aplinka ir kreivių skaičius	Skaičiavimo ir atvaizdavimo laikas
CPU (GTK+, cairo), 100 Bezier kreivių	0,12956 s
GPU (OpenGL, GLSL), 100 Bezier kreivių	0,00017 s
CPU (GTK+, cairo), 1000 Bezier kreivių	1,25284 s
GPU (OpenGL, GLSL), 1000 Bezier kreivių	0,00161 s
CPU (GTK+, cairo), 10000 Bezier kreivių	12,73047 s
GPU (OpenGL, GLSL), 10000 Bezier kreivių	0,01785 s

Viena iš galimų užduočių, kurios sprendimą GP gali gerokai paspartinti yra NURBS paviršių vaizdavimas [KK09]. Autorius naudojo „klasikinį“ GP programavimo metodą – OpenGL su GLSL fragmentų šešėliavimo programa. Buvo testuojama pasitelkus 3 testavimo sistemas ir pasiekti rezultatai: pirmoje testavimo sistemoje GP sugebėjo viską suskaičiuoti 60 kartų greičiau, antroje – 12 kartų, trečioje – 47 kartus. Šie rezultatai pavaizduoti 1 pav.



1 pav. NURBS paviršiaus taškų skaičiavimo GP ir CP greičių palyginimas [KK09]

Akivaizdu, kad atliekant pakankamai daug aritmetinių operacijų lygiagrečiai, šiuolaikiniai GP gerokai lenkia CP. Taip pat galima rasti daug CUDA pritaikymo pavydžių [Nvi10], kur pavyko paspartinti konkrečios problemos sprendimą pasitelkus NVIDIA grafikos procesorius ir CUDA technologiją.

1.1.2. GP architektūra ir programavimo modelis

GP visada buvo procesorius, siūlantis daug skaičiavimo resursų. Viena iš svarbiausių pastarųjų metų GP vystymo tendencijų yra suteikti programuotojams lengvesnį priėjimą prie šių resursų. Per pastaruosius metus, GP evoliucionavo iš fiksuoto funkcionalumo specialios paskirties procesoriaus į visiškai išvystytą lygiagretųjį programuojamą procesorių su papildomu fiksuotu specialios paskirties funkcionalumu [OHL+08].

GP programuojami elementai yra realizuoti remiantis vienos programos kelių duomenų vienetų programavimo modeliu (angl. *single-program multiple-data, SPMD*) [OHL+08]. Tam, kad GP dirbtų efektyviai, GP apdoroja daug elementų (viršūnių ar fragmentų) lygiagrečiai naudodamas tą pačią programą. Kiekvienas elementas nepriklauso vienas nuo kito ir negali pasiekti kito elemento duomenų. Taigi visos GP programos privalo išlaikyti tokią struktūrą: daug lygiagrečių elementų, kurie visi yra apdorojami lygiagrečiai vykdant tą pačią programą.

Kiekvienas elementas gali operuoti 32 bitų tikslumo sveikųjų ar slankaus kablelio skaičių duomenimis (naujesniuose GP galima naudoti ir 64 bitų tikslumo duomenis), naudodamas bendrojo pobūdžio instrukcijų rinkinį. Elementai gali nuskaityti duomenis iš globalios atminties, o naujesni GP leidžia ir rašyti duomenis į šią atmintį.

Šis programavimo modelis yra puikiai tinkamas vykdyti programas be išsišakojimų, kur daug elementų gali būti apdorojami, naudojant identišką instrukcijų seką. Toks programavimo modelis yra vadinamas vienos instrukcijos kelių duomenų vienetų (angl. *single-instruction multiple-data, SIMD*) [OHL+08]. Tačiau GP programoms tampant sudėtingesnėms, atsirado poreikis leisti skirtingiems elementas eiti per skirtingas programos atšakas, kuris nulėmė bendresnio SPMD modelio naudojimą. Kaip tai palaiko grafikos procesorius? Elementai tapo grupuojami į blokus, ir kiekvienas blokas yra apdorojamas lygiagrečiai. GP skaičiavimo programos yra sudaromos tokiu būdu [OHL+08]:

1. Programuotojas apibrėžia skaičiavimų sritį kaip struktūrizuotą gijų tinklelį.
2. SPMD bendrosios paskirties programa suskaičiuoja kiekvienos gijos reikšmę.
3. Kiekvienos gijos reikšmė yra suskaičiuojama kombinuojant matematines operacijas ir atminties operacijas (nuskaitant iš globalios atminties bei rašant į globalią atmintį).
4. Rezultato buferis globalioje atmintyje gal būti panaudojamas kaip įeiga kitiems skaičiavimams.

NVIDIA trečios kartos Fermi skaičiavimo architektūra yra organizuojama į 16 srauto multiprocesorius (angl. *streaming multiprocessors, SMs*), kurių kiekvienas turi 32 branduolius. GigaThread darbo planuoklis paskirsto gijų blokus srauto multiprocesoriams, dinamiškai balansuodamas skaičiavimų apkrovą visame grafiniame procesoriuje. Daugelio gijų srauto multiprocesoriai suplanuoja ir įvykdo gijų blokus bei individualias gijas. Kiekvienas srauto multiprocesorius gali vykdyti iki 1536 lygiagrečių gijų. Kai gijų blokas baigia vykdyti savo programą ir atlaisvina srauto multiprocesoriaus resursus, darbo planuoklis priskiria šiam srauto multiprocesoriui naują gijų bloką. PCIe jungtis jungia GP ir jo operatyviąją atmintį su CP ir sistemos operatyviąja atmintimi. Fermi GP architektūra balansuoja lygiagretaus skaičiavimo galią su lygiagrečiais DRAM atminties valdikliais. Ši architektūra turi 6 GDDR5 DRAM interfeisus, kurių kiekvieno plotis yra 64 bitai.

1.2. OpenCL

Šių dienų kompiuteriai be centrinio procesoriaus, dažnai dar turi ir kitokio tipo procesorius, tokius kaip grafikos procesorius. Labai svarbu įgalinti programinę įrangą kuriančius žmones išnaudoti šias heterogenines skaičiavimo platformas [Khr10b].

Kurti aplikacijas heterogeninėms skaičiavimo platformoms yra pakankamai sunku. Daugelio branduolių programavimas CP ir GP yra skirtingas. CP grįstas lygiagretusis programavimo modelis yra paremtas standartais, naudoja bendrą adresų erdvę ir nevykdo vektorinių operacijų. Bendrojo pobūdžio GP programavimo modeliai turi sudėtingas atminties hierarchijas ir vektorines operacijas, tačiau šie modeliai gali skirtis, priklausomai nuo platformos

ar konkretaus procesoriaus kūrėjo. Visa tai labai apsunkina programinę įrangą kuriančiam žmogui išnaudoti visus heterogeninės platformos procesorius (centrinius procesorius, grafinius procesorius ar kitus procesorius), rašant vieną multi-platforminį kodą [Khr10b]. Labiau nei bet kada anksčiau, atsirado poreikis efektyviai išnaudoti heterogenines skaičiavimo platformas – nuo didelio galingumo serverių, personalinių kompiuterių iki nešiojamų įrenginių, kurie gali turėti įvairų kiekį CP, GP ir kitokius procesorius tokius kaip DSP ar Cell Broadband Engine, kuris yra naudojamas Sony Playstation 3 žaidimų konsolėje [BF10].

OpenCL – tai atviras ir nemokamas standartas, skirtas bendrojo pobūdžio lygiagrečiajam programavimui, naudojantis CP, GP ar kitokiais procesoriais bei suteikiantis portatyvų ir efektyvų priėjimą prie heterogeninių skaičiavimo platformų galios.

OpenCL yra sudarytas iš API, skirtos koordinuoti lygiagrečius skaičiavimus tarp heterogeninių procesorių, ir daugiaplatforminės programavimo kalbos. OpenCL standartas [Khr10b]:

1. Palaiko duomenimis grįstus ir užduotimis grįstus programavimo modelius.
2. Panaudoja dalį ISO C99 standarto, pridėdamas tam tikrus plėtinius lygiagretumui.
3. Apibrėžia nuoseklius skaitmenų reikalavimus, remiantis IEEE 754 standartu.
4. Apibrėžia konfigūruojamą profilį rankiniams ir įterptiesiems įrenginiams.
5. Efektyviai suderinamas su OpenGL, OpenGL ES ir kitais grafikos API.

1.2.1. OpenCL architektūra

OpenCL suteikia žemo lygio aparatūrinės įrangos abstrakciją ir programavimą palaikantį karkasą. OpenCL naudoja 4 modelių hierarchiją [Khr10b]:

- Platformos modelis.
- Atminties modelis.
- Vykdymo modelis.
- Programavimo modelis.

1.2.1.1. Platformos modelis

Modelis yra sudarytas iš vieno pagrindinio šeimininko (pagrindinio kompiuterio), kuris yra prijungtas prie vieno ar daugiau OpenCL įrenginių [Khr10b]. OpenCL įrenginys yra dalinamas į vieną ar daugiau skaičiavimo vienetų (angl. *compute units, CUs*), kurie toliau yra dalinami į vieną ar daugiau apdorojimo elementų (angl. *processing elements, PEs*). Skaičiavimai yra atliekami apdorojimo elementuose.

OpenCL programa veikia pagrindiniame kompiuteryje (angl. *host*) sutinkamai su pagrindinio kompiuterio vietine platforma. Ši programa siunčia komandas iš pagrindinio

kompiuterio vykdyti skaičiavimus įrenginio apdorojimo elementuose. Šie skaičiavimo vieneto apdorojimo elementai vykdo vieną instrukcijų srautą, naudodami SIMD modelį (vykdo vieną ir tą patį instrukcijų srautą) arba naudodami SPMD modelį (vykdo programą su išsišakojimais – kiekvienas PE išlaiko savo programos skaitiklį).

1.2.1.2. Vykdymo modelis

OpenCL programa susideda iš dviejų dalių: branduolių, kurie yra vykdomi viename ar keliuose OpenCL įrenginiuose, ir pagrindinio kompiuterio programos, kuri apibrėžia kontekstą branduoliams ir valdo jų vykdymą [Khr10b].

Kai pagrindinė programa nusiunčia branduolį vykdymui, indeksavimo erdvė yra apibrėžiama. Branduolio egzempliorius yra vykdomas kiekviename indeksavimo erdvės taške. Branduolio egzempliorius yra vadinamas darbo elementu, kuris yra identifikuojamas savo koordinatėmis indeksavimo erdvėje, kurios suteikia globalų identifikatorių darbo elementui. Kiekvienas darbo elementas vykdo tą patį kodą, tačiau specifinis kodo vykdymo kelias ir duomenys, su kuriais yra operuojama, gali skirtis kiekviename darbo elemente.

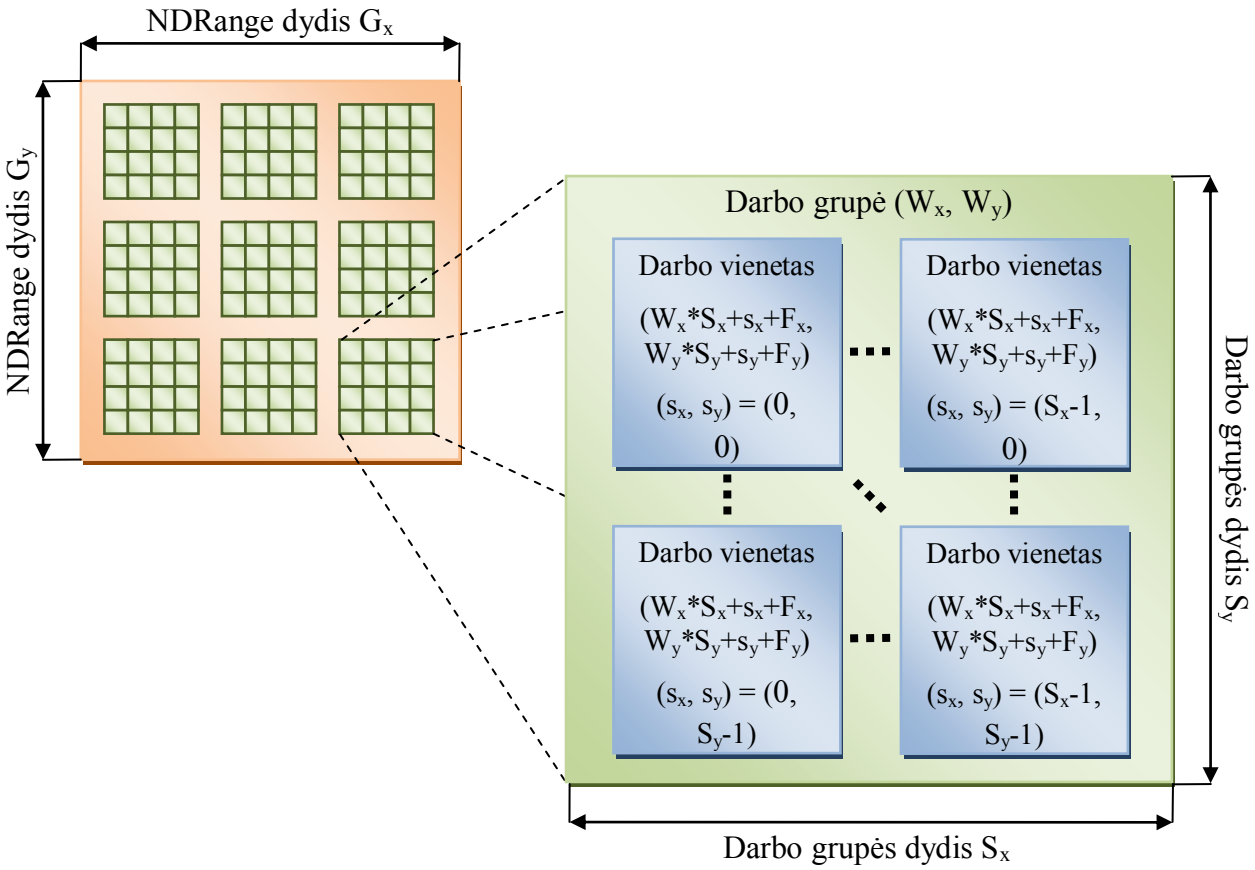
Darbo elementai yra organizuojami į darbo grupes, kurios suteikia grubesnį indeksavimo erdvės suskaidymą. Kiekvienai darbo grupei yra priskiriamas unikalus darbo grupės identifikatorius, kurio dimensinė struktūra yra tokia pati, kokia yra naudojama apibrėžiant darbo elementų indeksavimo erdvę. Darbo elementams darbo grupės viduje yra priskiriami unikalūs lokalūs identifikatoriai, taigi vienas darbo elementas gali būti unikaliai identifikuotas jo globaliu identifikatoriumi arba jo lokalaus identifikatoriaus ir darbo grupės identifikatoriaus kombinacija. Darbo elementai konkrečioje darbo grupėje yra vykdomi tuo pačiu metu pasitelkus vieno skaičiavimo įrenginio apdorojimo elementus.

Indeksavimo erdvė, kurią palaiko OpenCL, yra vadinama NDRange (2 pav.). NDRange – tai N-dimensinė indeksavimo erdvė, kur N gali būti 1, 2 arba 3. NDRange yra apibrėžiamas N ilgio sveikųjų skaičių masyvu, kuris nusako indeksavimo erdvės dydį kiekvienoje dimensijoje, pradedant indeksu F (0 pagal nutylėjimą). Kiekvieno darbo elemento globalus ir lokalus identifikatoriai yra N-dimensinis elementų sąrašas. Globalaus identifikatoriaus komponentai – tai reikšmės intervale nuo F iki F plius elementų skaičius šioje dimensijoje minus vienas.

Darbo grupėms identifikatoriai yra priskiriami naudojantis panašiu būdu kaip ir priskiriant globalius identifikatorius darbo elementams. N ilgio masyvas apibrėžia darbo grupių skaičių kiekvienoje dimensijoje. Darbo elementai yra priskiriami į darbo grupę ir jiems yra suteikiami lokalūs identifikatoriai, kurių komponentų reikšmės yra intervale nuo nulio iki darbo grupės dydis konkrečioje dimensijoje minus vienas. Taigi, darbo grupės identifikatoriaus ir konkrečioje darbo grupėje lokalaus identifikatoriaus kombinacija unikaliai apibrėžia darbo elementą.

Kiekvienas darbo elementas yra identifikuojamas dviem būdais: globaliu indeksu ir darbo grupės indeksu plius lokaliu indeksu darbo grupės viduje.

Daug programavimo modelių gali būti atvaizduojami į šį vykdymo modelį, tačiau OpenCL palaiko šiuos: duomenų lygiagretųjį ir užduočių lygiagretųjį programavimo modelius.



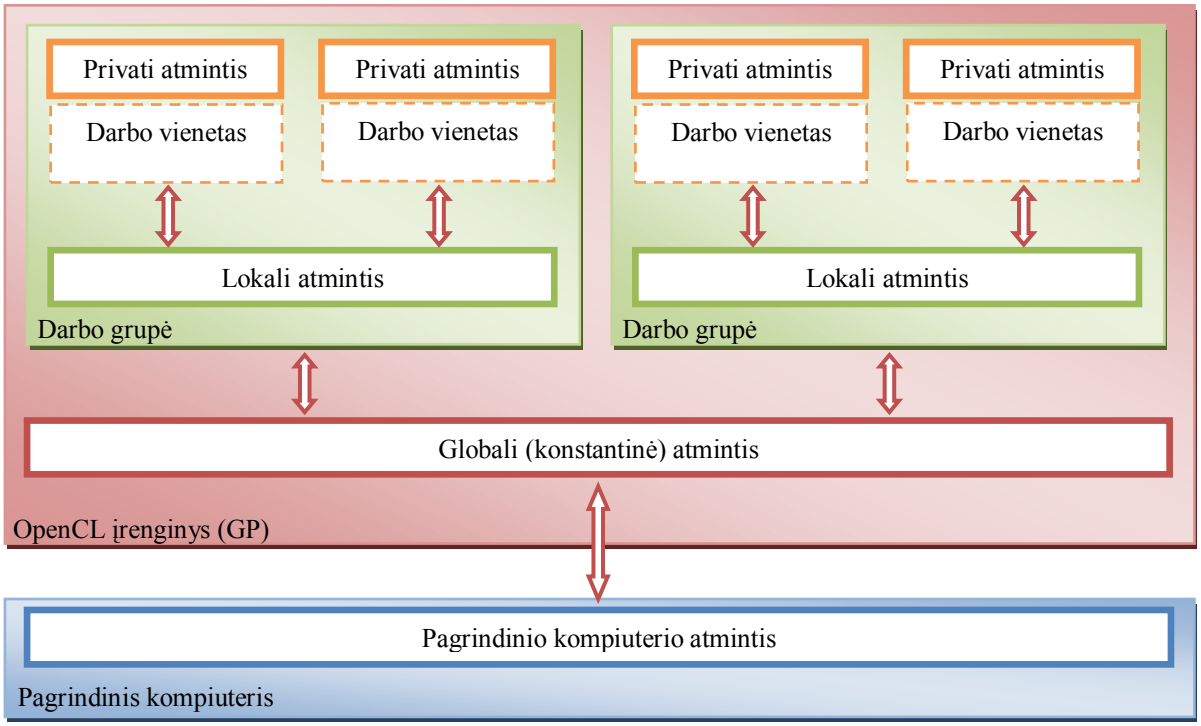
2 pav. NDRange indeksavimo erdvės pavyzdys, kuris rodo darbo elementus, jų globalius identifikatorius ir darbo elementų atvaizdavimą į darbo grupės ir lokalaus identifikatoriaus porą (parengta pagal [Khr10b])

1.2.1.3. Atminties modelis

Darbo elementai, vykdydami branduolį, gali pasiekti 4 atskirus atminties regionus [Khr10b] (3 pav.):

- **Globalią atmintį.** Visiems darbo elementams iš visų darbo grupių yra leidžiama tiek rašyti į šią atmintį, tiek skaityti iš jos. Darbo elementai gali skaityti iš arba rašyti į bet kurį atminties objekto elementą.
- **Konstantinę atmintį.** Globalios atminties regionas, kuris išlieka nepakitęs branduolio vykdymo metu. Pagrindinė programa (veikianti pagrindiniame kompiuteryje) išskiria ir inicijuoja atminties objektus, kurie yra padedami į konstantinę atmintį.

- **Lokalią atmintį.** Tai – atminties regionas, kuris yra lokalus darbo grupei. Šis atminties regionas gali būti naudojamas išskiriant kintamuosius, kuriais gali dalintis visi darbo elementai konkrečioje darbo grupėje.
- **Privačią atmintį.** Tai – privatus darbo elemento atminties regionas. Kintamieji apibrėžti darbo elemento privačioje atmintyje nėra matomi kitam darbo elementui.



3 pav. OpenCL atminties modelis (parengta pagal [Khr10a])

4 lentelė aprašo branduolio ir pagrindinės kompiuterio programos gebėjimus išskiriant vietą iš konkrečios atminties regiono, išskyrimo tipą (statinis – kompiliavimo metu, dinaminis – veikimo metu) ir priėjimo prie atminties tipą.

4 lentelė. Atminties regionai – išskyrimo ir atminties priėjimo gebėjimas [Khr10b]

	Globali	**Konstantinė**	**Lokali**	**Privati**
Pagrindinio kompiuterio programa	Dinaminis išskyrimas Skaitymo/rašymo priėjimas	Dinaminis išskyrimas Skaitymo/rašymo priėjimas	Dinaminis išskyrimas Priėjimas neleistinas	Išskyrimas neleistinas Priėjimas neleistinas
Branduolys	Išskyrimas neleistinas Skaitymo/rašymo priėjimas	Statinis išskyrimas Skaitymo/rašymo priėjimas	Statinis išskyrimas Skaitymo/rašymo priėjimas	Statinis išskyrimas Skaitymo/rašymo priėjimas

Pagrindinio kompiuterio ir OpenCL įrenginio atminties modeliai didžiąja dalimi yra nepriklausomi vienas nuo kito. Taip yra dėl būtinybės, kad pagrindinis kompiuteris yra apibrėžtas už OpenCL ribų. Tačiau šie modeliai kartais turi sąveikauti vienas su kitu. Ši sąveika gali įvykti dviem būdais: išreikštinai kopijuojant duomenis arba atvaizduojant ir pašalinant atvaizdavimą į tam tikrus atminties objekto regionus.

Tam, kad nukopijuoti duomenis išreikštinai, pagrindinis kompiuteris įdeda į eilę komandas, skirtas duomenims perkelti tarp atminties objekto ir pagrindinio kompiuterio atminties.

Atvaizdavimo ir atvaizdavimo pašalinimo metodai, skirti sąveikai tarp pagrindinio kompiuterio ir OpenCL atminties objektų, leidžia pagrindiniam kompiuteriui atvaizduoti tam tikrą atminties objekto regioną į savo adresinę erdvę.

1.2.1.4. Programavimo modelis

OpenCL vykdymo modelis palaiko duomenų lygiagretųjį ir užduočių lygiagretųjį programavimo modelius, taip pat šių dviejų modelių hibridus [Khr10b]. Tačiau pirminis modelis, į kurį yra atsižvelgiama projektuojant OpenCL yra duomenų lygiagretusis modelis.

1.2.1.4.1. Duomenų lygiagretusis programavimo modelis

Duomenų lygiagretusis programavimo modelis apibrėžia skaičiavimą kaip instrukcijų seką, kuri yra pritaikoma keletui atminties objekto elementų vienu metu. Indeksavimo erdvė, susieta su OpenCL vykdymo modeliu, apibrėžia darbo elementus ir kaip duomenys yra atvaizduojami į šiuos darbo elementus. Griežtai duomenų lygiagrečiajame modelyje yra apibrėžiamas vienas į vieną sugretinimas tarp darbo elementų ir elementų atminties objekte, su kuriais branduolys gali būti vykdomas lygiagrečiai. OpenCL realizuoja laisvesnį duomenų lygiagretųjį programavimo modelį, kur griežtas vienas į vieną sugretinimas nėra reikalaujamas.

OpenCL suteikia hierarchinį duomenų lygiagretųjį programavimo modelį. Yra du būdai apibrėžti hierarchinį padalinimą. Išreikštiniame modelyje programuotojas apibrėžia darbo elementų skaičių, kurie turi būti vykdomi lygiagrečiai, taip pat apibrėžia kaip darbo elementai yra padalinami tarp darbo grupių. Neišreikštiniame modelyje programuotojas apibrėžia tik darbo elementų skaičių ir šių darbo elementų padalinimas tarp darbo grupių yra valdomas OpenCL realizacijos.

1.2.1.4.2. Užduočių lygiagretusis programavimo modelis

OpenCL užduočių lygiagretusis programavimo modelis apibrėžia modelį, kuriame vienas branduolio egzempliorius yra vykdomas nepriklausomai nuo jokios indeksavimo erdvės. Tai yra

logiškai ekvivalentu kas vykdyti branduolį naudojantis skaičiavimo vienetu su darbo grupe, kuri turi tik vieną darbo elementą. Naudojantis šiuo modeliu, vartotojai gali išreikšti lygiagretumą:

- Naudodami vektorinius duomenų tipus, kuriuos realizuoja įrenginys.
- Įdedant į eilę keletą užduočių.
- Įdedant į eilę savuosius branduolius, sukurtus naudojantis ortogonaliu OpenCL programavimo modeliu.

1.2.1.4.3. Sinchronizacija

Yra dvi OpenCL sinchronizavimo sritys:

- Darbo elementai vienoje darbo grupėje.
- Komandos, kurios yra įdėtos į komandų eilę viename kontekste.

Sinchronizacija tarp darbo elementų vienoje darbo grupėje yra atliekama naudojantis darbo grupės barjeru. Visi darbo grupės darbo elementai privalo pasiekti barjerą, kol bent vienam darbo elementui yra leidžiama vykdyti kodą už barjero ribų. Sinchronizacija galima tik darbo grupės viduje – tarp darbo grupių sinchronizacija negalima.

Komandų eilėse sinchronizacijos taškai tarp komandų yra:

- **Komandų eilės barjeras.** Komandų eilės barjeras užtikrina, kad visų prieš tai į komandų eilę įdėtų komandų vykdymas buvo baigtas ir bet kokie atminties objekto atnaujinimai yra matomi sekančioms į eilę padėtoms komandoms. Šis barjeras gali būti naudojamas tik sinchronizuojant tarp komandų vienoje komandų eilėje.
- **Įvykio laukimas.** Visos OpenCL API funkcijos, kurios įdeda į komandų eilę komandas, gražina įvykį, kuris identifikuoja komandą ir atnaujinamus atminties objektus. Sekančios komandos, kurios laukia šio įvykio, yra užtikrinamos, kad pakeitimai atminties objektuose yra matomi prieš pradedant šių komandų vykdymą.

1.2.1.5. Atminties objektai

Atminties objektai yra kategorizuojami į du tipus: buferio objektus ir paveikslo objektus. Buferio objektas saugo vienos dimensijos elementų rinkinį, o paveikslo objektas yra naudojamas saugoti dviejų ar trijų dimensijų tekstūrą, kadro buferį ar paveikslą.

Buferio objekto elementai gali būti skaliarinio (tokie kaip sveikieji ar slankaus kablelio skaičiai), vektorinio ar vartotojo apibrėžto tipo. Paveikslo objektas yra naudojamas reprezentuoti buferį, kuris gali būti naudojamas kaip tekstūra ar kadro buferis. Paveikslo objekto elementai yra parenkami iš iš anksto apibrėžto paveikslo formatų sąrašo. Minimalus atminties objekto elementų skaičius yra vienas.

Fundamentalūs skirtumai tarp buferio ir paveikslo objektų [Khr10b]:

- Elementai buferyje yra dedami nuosekliai ir branduolys, kuris yra vykdomas įrenginyje, gali pasiekti šiuos elementus naudodamas rodykles. Paveikslo elementai yra dedami naudojant vartotojui nematomą formatą, ir šie elementai negali būti pasiekiami naudojantis rodyklėmis. Įtaisytąsias funkcijas, kurios leidžia branduoliui skaityti iš paveikslo ar rašyti į jį, suteikia OpenCL C programavimo kalba.
- Buferio objektuose duomenys yra laikomi tokiame pačiame formate kaip į juos kreipiasi branduolys, tačiau paveikslo objekte duomenų formatas, naudojamas patalpinti paveikslo elementus, gali ir nebūti toks pat koks yra naudojamas branduolyje. Paveikslo elementai branduolyje visada yra 4 komponentų vektoriai (kiekvienas komponentas gali būti slankaus kablelio arba sveikasis su ženklu ar be jo skaičius). Įtaisytosios funkcijos, kurios yra skirtos skaityti iš paveikslo, konvertuoja paveikslo elementus iš formato, kuriuo naudojantis šie elementai buvo patalpinti, į 4 komponentų vektorius. Ir panašiai, įtaisytosios funkcijos, skirtos rašyti į paveikslą, konvertuoja paveikslo elementus iš 4 komponentų vektoriaus į atitinkamą paveikslo formatą (pavyzdžiui, keturis 8 bitų elementus).

Branduoliai paima atminties objektus kaip įeitį, ir išveda rezultatus į vieną ar kelis atminties objektus.

1.2.1.6. OpenCL karkasas

OpenCL karkasas leidžia taikomajai programai naudoti pagrindinį kompiuterį ir vieną ar kelis OpenCL įrenginius kaip vieną heterogeninę lygiagrečiąją kompiuterinę sistemą. Šis karkasas susideda iš šių komponentų [Khr10b]:

- **OpenCL platformos sluoksnis.** Platformos sluoksnis įgalina pagrindinio kompiuterio programą aptikti įrenginius bei jų pajėgumą ir sukurti kontekstus.
- **OpenCL vykdymo aplinka.** Vykdymo aplinka leidžia pagrindinio kompiuterio programai manipuliuoti kontekstais kai tik jie yra sukuriami.
- **OpenCL kompiliatorius.** OpenCL kompiliatorius sukuria programos vykdomuosius failus, kuriuose yra OpenCL branduoliai. Kompiliatoriaus realizuota OpenCL C programavimo kalba palaiko ISO C99 kalbos poaibį bei turi praplėtimus, kurie yra skirti lygiagretumui išreikšti.

1.2.2. OpenCL ir CUDA

2009 metais NVIDIA pristatė trečios kartos Fermi GPU skaičiavimo architektūrą. Ši architektūra realizavo IEEE 754-2008 standartą ir gerokai padidino dvigubo tikslumo skaičiavimo greitį. Taip pat buvo papildomai pridėta atminties nuo iškraipymų apsauga ECC

(angl. *error correcting code*), 64 bitų unifikuotas adresavimas, instrukcijos skirtos C, C++, Fortran, OpenCL ir DirectCompute programavimo kalboms [ND10]. Tai leido NVIDIA GP vykdyti programas parašytas CUDA C, CUDA C++, CUDA Fortran, DirectCompute ir OpenCL programavimo kalbomis. Nors ir CUDA siūlo aukšto lygio GP programavimo interfeisą, tačiau didžiausias CUDA trūkumas, lyginant su OpenCL, yra tai, kad CUDA programos veikia tik su NVIDIA technine įranga [SKP10].

1.3. Eismo dalyvių kelyje atpažinimas

Eismo įvykių statistika rodo, kad pagrindinis pavojus kelyje, su kuriuo susiduria vairuotojai, yra kiti eismo dalyviai [SBM06]. Dėl šios priežasties įvairių pagalbinių sistemų, skirtų perspėti vairuotoją apie vairavimo aplinką ir galimus susidūrimus su kitais eismo dalyviais, kūrimas susilaukia daug dėmesio. Kuriant tokias sistemas, pirmasis žingsnis yra sukurti patikimą ir veiksmingą eismo dalyvių kelyje aptikimą. Eismo dalyvių aptikimas ir sekimas turi daug pritaikymų: transporto priemonių grupavimas (angl. *platooning*, tai transporto priemonių keliavimas dideliu greičiu ir mažu atstumu viena nuo kitos greitkeliuose), sustojimas ir važiavimas (angl. *stop and go*, tai transporto priemonių keliavimas mažu greičiu ir mažu atstumu viena nuo kitos miestuose) ir autonominis važiavimas.

Eismo dalyvių aptikimas naudojant optinius jutiklius yra labai sunki užduotis dėl labai didelio nepastovumo, kaip eismo dalyviai gali atrodyti kelyje [SBM06]. Jie gali varijuoti savo forma, dydžiu ir spalva. Taip pat išvaizda priklauso nuo eismo dalyvio pozos ir yra veikiama aplinkinių objektų. Eismo dalyvių aptikimas reikalauja greitesnio apdorojimo, kadangi eismo dalyvio greitis kelyje yra susijęs su apdorojimo dažniu.

Eismo dalyvių aptikimo sistemos reikalauja daug skaičiavimo pajėgumų, nes šios sistemos privalo apdoroti gaunamus vaizdus realiu laiku ar bent jau arti to, tam kad užtektų laiko vairuotojui sureaguoti į įvykius kelyje [SBM06]. Daugelis esančių metodų atlieka du pagrindinius žingsnius:

1. Hipotezės generavimas – iškeliamos hipotezės, spėjama kur paveikslėlyje yra eismo dalyviai.
2. Hipotezės verifikavimas – atliekamas testavimas, tam kad įsitikinti, ar eismo dalyviai iš tiesų yra ten.

1.3.1. Aktyvūs ir pasyvūs jutikliai

Vienas iš labiausiai paplitusių metodų eismo dalyviams kelyje aptikti yra naudoti aktyvus jutiklius, kurių veikimas yra pagrįstas radaru (milimetrinės bangos), lazeriu (pvz., LIDAR) ar akustika [SBM06]. Radaro atveju radijo bangos yra paskleidžiamos į aplinką, iš kurios dalis

energijos atsispindi atgal į radaro imtuvą. LIDAR (angl. *light detection and ranging*) taip pat spinduliuoja ir priima elektromagnetinę spinduliuotę, tačiau didesniu dažniu – operuoja ultravioletinėje, matomajame ir infraraudoname elektromagnetinio spektro regionuose.

Šie jutikliai yra vadinami aktyviais, nes jie aptinka atstumą iki objekto matuodami išspinduliuoto signalo keliavimo laiką, kuris atsispindi nuo šių objektų. Vienas iš pagrindinių šių jutiklių privalumų yra tai, kad jie gali matuoti tam tikrus dalykus (pvz., atstumą) tiesiogiai nereikalaujant jokių didelių skaičiavimo resursų [SBM06]. Radarų sistemos gali aptikti objektus bent už 150 metrų per rūką ar lietų, kur vidutinis vairuotojas gali matyti tik 10 metrų ar mažiau. LIDAR jutikliai yra pigesni nei radarai, tačiau šie jutikliai veikia prasčiau už radarus, kai lyja ar sninga. Lazeriu pagrįstos sistemos yra tikslesnės nei radarai, tačiau jų pritaikymas yra ribotas dėl didesnių kaštų. Prototipinės transporto priemonės, kurios naudoja aktyvius jutiklius, demonstravo daug žadančius rezultatus. Tačiau, kai daug tokių pačių transporto priemonių juda ta pačia kryptimi tuo pat metu, to paties tipo jutikliai trukdo vieni kitiems.

Optiniai jutikliai, tokie kaip kameros, yra vadinami pasyviais jutikliais, nes jie surenka duomenis pasyviai – nieko nespinduliuodami [SBM06]. Vienas iš pasyvių jutiklių privalumų yra jų kaina. Nebrangių kamerų dėka, galima įrengti kameras tiek transporto priemonės priekyje, tiek užpakalyje, kas leistų turėti beveik 360 laipsnių apžvalgos kampą. Optiniai jutikliai gali būti naudojami efektyvesniam transporto priemonių sekimui, kurios įsuka į kelią ar juda iš vienos kelio pusės į kitą. Taip pat, vizuali informacija gali būti labai svarbi daugelyje susijusių pritaikymų, tokių kaip eismo juostos aptikimo, kelio ženklų atpažinimui ar tam tikrų objektų identifikavimui. Ir tai nereikalauja jokių modifikacijų kelio infrastruktūrai.

1.3.2. Hipotezės generavimo metodai

Hipotezės generavimo metodus galime suskirstyti į 3 pagrindines kategorijas [SBM06]:

1. Žiniomis grįsti metodai.
2. Stereo vaizdu grįsti metodai.
3. Judėjimu grįsti metodai.

Šių metodų pagrindinė užduotis yra sparčiai surasti galimas eismo dalyvių pozicijas paveikslėlyje, kurias vėliau būtų galima detaliau analizuoti.

1.3.2.1. Žiniomis grįsti metodai

Žiniomis grįsti metodai naudoja iš anksto žinomas žinias, tam kad būtų galima iškelti hipotezes apie eismo dalyvių poziciją paveikslėlyje. Tai galėtų būti informacija apie simetriją, spalvą, šešėlius, geometrines savybes (pvz., kampai, horizontalios bei vertikalios briaunos), tekstūras ir eismo dalyvių šviesas [SBM06].

1.3.2.1.1. Simetrija

Viena iš pagrindinių žmogaus sukurtų objektų savybių yra simetrija, kuri dažnai yra naudojama objektų aptikimui ir atpažinimui [SBM06]. Eismo dalyvių vaizdas, gautas iš priekio ar galo, bendru atveju yra simetrinis horizontalia ir vertikalia kryptimi. Tačiau skaičiuojant simetriškumą iškyla svarbi problema, kuomet paveikslėlyje atsiranda homogeninių sričių. Šiose srityse simetrijos įvertinimas yra jautrus triukšmui. Informacija apie briaunas gali būti įtraukta į simetrijos įvertinimą, tam kad būtų galima išfiltruoti homogenines sritis.

1.3.2.1.2. Spalvos

Nors ir keletas jau egzistuojančių sistemų naudoja spalvinę informaciją hipotezių generavimui, tačiau ši informacija taip pat yra labai naudinga ir kliūčių aptikimui, eismo juostos ar kelio sekimui [SBM06].

1.3.2.1.3. Šešėliai

Transporto priemonių aptikimui galime naudoti ir šešėlių informaciją [SBM06]. Atliekant paveikslėlių intensyvumo tyrimus buvo aptikta, kad regionai po transporto priemone yra tamsesni nei kiti asfaltuoto kelio regionai. Tačiau yra labai sunku nustatyti tinkamas ribines reikšmes, nes šešėlių intensyvumas priklauso nuo paveikslėlio ryškumo, o paveikslėlio ryškumą lemia oro sąlygos. Dėl šių priežasčių ribinės reikšmės negali būti fiksuotos. Akivaizdu, kad labai sunku nustatyti šešėlio srities žemesniąją ribinę reikšmę, tuo tarpu aukštesnioji ribinė reikšmė gali būti apytiksliai apskaičiuota analizuojant kelio sritis, kuriomis nevažiuoja joks eismo dalyvis (pvz., kelio sritis prieš transporto priemonę).

1.3.2.1.4. Kampai

Transporto priemonė bendruoju atveju turi stačiakampę formą su keturiais kampais (viršuje kairėje, viršuje dešinėje, apačioje kairėje, apačioje dešinėje). Šiuo faktu galima pasinaudoti bandant aptikti transporto priemones paveikslėlyje [SBM06]. Turėdami 4 šablonus kiekvienam kampui aptikti, naudojantis įvairiais paieškos paveikslėlyje metodais, galime mėginti surasti kiekvieną iš 4 kampų ir taip mėginti nuspėti transporto priemonės padėtį paveikslėlyje.

1.3.2.1.5. Vertikalios ir horizontalios briaunos

Iš skirtingų kampų daryti transporto priemonės paveikslėliai, ypač iš priekio ar iš galo, turi daug horizontalių ir vertikalių struktūrų, tokių kaip priekinis langas, bamperiai ir pan. Naudojant daug vertikalių ir horizontalių briaunų, galime mėginti nuspėti transporto priemonės poziciją paveikslėlyje [SBM06].

1.3.2.1.6. Tekstūros

Transporto priemonės buvimas paveikslėlyje sukelia lokalius intensyvumo pasikeitimus. Dėl tam tikrų bendrų transporto priemonių panašumų šie intensyvumo pasikeitimai seka tam tikrą tekstūros šabloną. Ši tekstūros informacija gali būti naudojama kaip užuomina, kur galėtų būti transporto priemonė paveikslėlyje, tam kad būtų galima sumažinti paieškos plotą [SBM06].

1.3.2.1.7. Transporto priemonių šviesos

Daugelis aukščiau aptartų hipotezių generavimo metodų nėra labai naudingi nakties metu. Nakties sąlygomis gautuose paveiksliukuose būtų labai sunku ar iš vis neįmanoma aptikti šešėlius, vertikalias ar horizontalias briaunas ar kampus. Ryškiausia vizuali transporto priemonės savybė naktį yra lempos. Galima mėginti aptikti transporto priemonių lempų poras bei atsižvelgiant į formą, dydį ir minimalų atstumą tarp transporto priemonių mėginti nuspėti transporto priemonės poziciją paveikslėlyje [SBM06].

1.3.2.2. Stereo vaizdu grįsti metodai

Yra du pagrindiniai metodų tipai, kurie naudoja stereo informaciją transporto priemonėms vaizde aptikti [SBM06]. Vieni naudoja skirtumų žemėlapius (angl. *disparity map*), kiti – antiperspektyvinę transformaciją – atvirkštinės perspektyvos žemėlapio braižymas (angl. *inverse perspective mapping*).

Tam, kad gauti stereo vaizdą, reikia turėti 2 kameras, kurios turi būti sukalibruotos būti lygiagrečiai viena kitai. Tačiau galima ir vienos kameros stereo vaizdo sistema [DAL+07]. Ši sistema naudoja vieną kamerą, du veidrodžius bei prizmę, kuri nukreipia veidrodžių vaizdą tiesiai į kamerą. Vienos pusės vaizdas yra nukreipiamas į vieną kameros jutiklio pusę, kitos – į kitą kameros jutiklio pusę.

1.3.2.2.1. Nesugretinamumo žemėlapis

Skirtumas tarp kairiojo ir dešiniojo vaizdų atitinkamų pikselių yra vadinamas nesugretinamumas (angl. *disparity*) [SBM06]. Visų vaizdo taškų nesugretinamumas suformuoja nesugretinamumo žemėlapį. Jei stereo įrangos parametrai yra žinomi, tuomet nesugretinamumo žemėlapis gali būti konvertuojamas į nufotografuotos scenos 3D žemėlapį. Nesugretinamumo žemėlapio skaičiavimas yra daug skaičiavimo resursų reikalaujanti užduotis, nes reikia išspręsti atitikimo problemą kiekvienam pikseliui, tačiau tai įmanoma padaryti realiu laiku naudojant Pentium klasės procesorių. Kai nesugretinamumo žemėlapis yra paruoštas, visi pikseliai, kurie patenka į dominantį gylį (nustatomą pagal nesugretinamumo intervalą), yra nustatomi ir akumuliuojami į nesugretinamumo histogramą. Jei kokia nors kliūtis yra prieš transporto

priemonę dominančiame gylyje, tuomet bus pasiekiamas maksimumas atitinkamoje histogramoje.

1.3.2.2.2. Atvirkštinės perspektyvos žemėlapio braižymas

Terminas „atvirkštinės perspektyvos žemėlapio braižymas“ neatitinka tikrosios perspektyvos žemėlapio braižymo inversijos, nes tai matematiškai yra neįmanoma [SBM06]. Šis terminas labiau reiškia inversiją su papildomu apribojimu, kad atvirkščiai sužymėti taškai žemėlapyje guli ant horizontalios plokštumos. Jei įsivaizduotume tašką p 3D erdvėje, perspektyvos žemėlapio braižymas reiškia liniją, praleistą per šį tašką ir N projekcijos centrą. Tam, kad rastume taško atvaizdį, perkertame liniją su vaizdo plokštuma. Atvirkštinės perspektyvos žemėlapio braižymas gali būti apibrėžimas kaip procedūra: paveiksliuko taškui $p`_I$ trasuojame asocijuotą spindulį per N į horizontaliąją plokštumą. Spindulio susikirtimas su horizontaliąja plokštuma yra atvirkštinės perspektyvos žemėlapio braižymo rezultatas, pritaikytas vaizdo taškui $p`_I$. Jei sukomponuotume perspektyvą ir atvirkštinę perspektyvą, horizontalioji plokštuma būtų atvaizduota į pačią save ir iškilios scenos dalys atrodytų iškraipytos.

1.3.2.3. Judėjimu grįsti metodai

Visi aukščiau pristatyti metodai naudoja erdvės ypatybes tam, kad atskirtų eismo dalyvius ir foną. Dar vienas būdas, kurį galime panaudoti, yra santykinis judesys, išgautas apdorojant optinį srautą [SBM06]. Pažymėkime paveiksliuko intensyvumą taške (x, y) duotuoju laiku t taip: E(x, y, t). Paveiksliukuose pikseliai atrodo tarsi judėtų dėl santykinio judesio tarp jutiklio ir scenos. Šio judesio vektorinis laukas o(x, y) yra vadinamas optiniu srautu. Optinis srautas gali suteikti stiprios informacijos hipotezių generavimui. Priešinga kryptimi artėjančios transporto priemonės sukelia nukrypstantį srautą, kuris gali būti kiekybiškai atskirtas nuo srauto, kurį sukelia pačios mašinos savasis judėjimas. Transporto priemonės, kurios lenkia ar atsiskiria, sukelia mažai nukrypstantį srautą. Tam, kad būtų galima pasinaudoti šiuo pastebėjimu kliūtims aptikti, paveiksliukas pirmiausia turi būti suskaldytas į mažus gabaliukus ir tuomet vidutinis greitis turi būti suskaičiuojamas kiekviename tokiame gabaliuke. Paveiksliuko gabaliukai, kurių greitis labai skiriasi nuo globalaus greičio įvertinimo, yra pažymimi kaip galimos kliūtys.

1.3.3. Hipotezės verifikavimo metodai

Hipotezių verifikavimo įeitis yra aibė galimų eismo dalyvių pozicijų vaizde, kurie buvo sugeneruoti hipotezių generavimo žingsnyje. Atliekant hipotezių verifikavimą, įvairūs testai yra

atliekami, tam kad įvertinti hipotezių teisingumą. Hipotezių verifikavimo metodai gali būti suskirstyti į dvi pagrindines kategorijas [SBM06]:

1. Šablonu grįsti metodai.
2. Išvaizda grįsti metodai.

1.3.3.1. Šablonu grįsti metodai

Šablonu grįsti metodai naudoja iš anksto apibrėžtus transporto priemonių klasių šablonus ir atlieka koreliaciją tarp vaizdo ir šablono [SBM06]. Vienas tokių metodų pavyzdys galėtų būti verifikavimo schema, kuri remtųsi transporto numerių ir galinio lango buvimo faktu.

1.3.3.2. Išvaizda grįsti metodai

Hipotezių verifikavimas, naudojant išvaizdos modelius, yra laikomas dviejų klasių šablonų klasifikavimo problema: transporto priemonės ir ne transporto priemonės [SBM06]. Šablonų klasifikavimo sistemos kūrimas apima optimalios apsisprendimo ribos tarp kategorizuojamų klasių paiešką.Tai nėra lengva užduotis, nes vienos klasės transporto priemonės gali būti labai įvairios. Vienas galimas užduoties sprendimas yra išmokti apie apsisprendimo ribą, remiantis klasifikatoriaus treniravimu, treniravimui naudojant iš treniravimo aibės išgautas savybes.

Išvaizda grįsti metodai mokosi apie transporto priemonių išvaizdos savybes naudodami aibę mokymosi vaizdų, kuriuose užfiksuotas transporto priemonės klasės išvaizdos įvairumas. Tam, kad pagerinti rezultatus, ne transporto priemonių klasės įvairumas taip pat yra modeliuojamas. Iš pradžių yra surenkamas didelis kiekis treniravimosi vaizdų ir kiekvienas mokymosi vaizdas yra pateikiamas kaip aibė lokalių ir globalių savybių. Tuomet apsisprendimo riba, ar tai transporto priemonė ar ne, yra išmokstama apmokant klasifikatorių (pvz., neuroninis tinklas) ar modeliuojant kiekvienos klasės savybių tikimybinį pasiskirstymą.

1.3.4. Aptikimo ir sekimo integravimas

Transporto priemonės aptikimas gali būti žymiai pagerintas tikslumo ir laiko atžvilgiu, pasinaudojant laikinu duomenų tolydumu. Tai gali būti pasiekta pasinaudojus sekimo mechanizmu, kurio pagalba būtų galima iškelti hipotezes apie transporto priemonės poziciją ateinančiuose kadruose. Sekimas remiasi faktu, kad labai mažai tikėtina, jog transporto priemonė pasirodys tik viename kadre. Transporto priemonės pozicija gali būti nuspėta remiantis istoriniais duomenimis bei prognozavimo mechanizmu [SBM06].

1.3.5. Jutiklių suliejimas

Kuriant vairuotojams skirtas pagalbos sistemas, iškyla papildomų sunkumų, jog tos sistemos turi būti tinkamos ir miesto aplinkoje, kur kelio ženklai, sankryžos, eismo kamščiai ir

kiti eismo dalyviai (motociklai, dviračiai, pėstieji ar pan.) gali egzistuoti [SBM06]. Išskirtinai vaizdu paremtos sistemos ir algoritmai dar nėra pakankamai galingi, kad susidorotų su sudėtingomis eismo situacijomis. Tam, kad praplėsti vairuotojams skirtų pagalbos sistemų pritaikymą, šios sistemos turi pasinaudoti informacija iš keleto jutiklių, tiek aktyvių, tiek pasyvių.

Jutiklių savybės nulemia, kad kiekvienas jutiklis gali suprasti tiek tam tikras aplinkos savybes. Taigi vienas jutiklis negali visapusiškai atspindėti vairavimo aplinkos [SBM06]. Keleto jutiklių sistemos turi potencialo suteikti didesnį patikimumo ir saugumo lygį.

1.3.6. Techninės įrangos problemos

Eismo dalyvių aptikimo sistemos reikalauja daug skaičiavimo pajėgumų, nes šios sistemos privalo apdoroti gaunamus vaizdus realiu laiku ar bent jau arti to, tam kad užtektų laiko vairuotojui sureaguoti į įvykius kelyje [SBM06]. Esant netrivialiam transporto priemonės greičiui, apdorojimo gaišties laikas turėtų būti mažas (ne ilgiau kaip 100 milisekundžių) ir apdorojimo dažnumas turėtų būti didelis (daugiau nei 15 kadrų per sekundę). Dėl gaišties laiko apribojimų ir iškylančių patikimumo problemų persiunčiant ir gaunant video duomenis, didžioji dalis vaizdo apdorojimo turi būti atliekama pačioje transporto priemonėje. Kompiuterinės regos algoritmai bendruoju atveju reikalauja labai daug resursų tam, kad būtų galima duomenis apdoroti realiu laiku ir laikytis apdorojimo greičio bei dažnumo apribojimų.

1.4. Vaizdų atpažinimas naudojant neuroninius tinklus

Neuroniniai tinklai atlieka šablono atpažinimo užduotis labai gerai, apmokius juos dideliu kiekiu treniravimo duomenų [SKP10]. Konvoliuciniai neuronai tinklai (KNT) šiuo metu yra pats moderniausias būdas atlikti paveiksliukų klasifikaciją (pvz., optinis raidžių atpažinimas). KNT – tai iš daugiasluoksnio suvokimo neuroninio tinklo išvestas neuroninis tinklas, optimizuotas atpažinti dviejų dimensijų šablonus. Šis neuroninis tinklas yra naudojamas ranka rašytoms raidėms atpažinti [CPS06, LBB+98], veidams [LGT+97], akims ir transporto priemonių numeriams aptikti. Tai tik keletas galimų pritaikymų.

1.4.1. Neuroniniai tinklai

Dirbtinių neuroninių tinklų tyrimų motyvacija nuo pat šių tyrimų pradžios buvo pastebėjimas, kad žmogaus smegenys skaičiuoja visiškai kitokiu būdu nei įprasti kompiuteriai. Smegenys yra labai sudėtingas, nelinijinis bei lygiagretusis kompiuteris (informacijos apdorojimo sistema) [Hay99]. Smegenys turi gebėjimą organizuoti jų sudedamąsias dalis – neuronus – taip, kad jos galėtų atlikti tam tikrus skaičiavimus (pvz., atpažinti įvairius šablonus,

suvokti įvairią informaciją ar atlikti motorinį valdymą) daug kartų greičiau nei greičiausi šių dienų kompiuteriai. Vienas iš pavyzdžių galėtų būti žmogaus rega, ką galime laikyti informacijos apdorojimo užduotimi. Regos sistemos funkcija yra suteikti aplinkos reprezentaciją ir visą reikalingą informaciją, kurios mums reikia sąveikaujant su aplinka. Smegenys reguliariai įvykdo suvokimo atpažinimo užduotis (pvz., pažystamo veido atpažinimas, kuris yra nepažystamoje aplinkoje) apytiksliai per 100-200 milisekundžių, kur tuo tarpu daug mažesnio sudėtingumo užduotys gali būti vykdomos labai ilgai įprastiniame kompiuteryje.

Kaip žmogaus smegenys sugeba tą padaryti? Gimimo metu, smegenys turi didžiulę struktūrą ir sugebėjimą sukurti savas taisykles, ką mes vadiname patirtimi. Patirtis yra sukuriama laikui bėgant – per pirmuosius 2 metus po gimimo smegenys vystosi intensyviausiai, tačiau vystymasis tęsiasi daug ilgiau po to.

Pačia bendriausia forma neuroninis tinklas yra mašina, kuri yra suprojektuota modeliuoti būdą, kuriuo smegenys atlieka tam tikrą užduotį ar tam tikrą dominančią funkciją. Tinklas dažniausiai yra realizuojamas naudojantis elektroniniais komponentais arba yra simuliuojamas kompiuteryje, naudojantis programine įranga. Tam, kad pasiekti gerus rezultatus, neuroninis tinklas naudoja didžiulį kiekį tarpusavyje susijungusių paprastų skaičiavimo ląstelių, vadinamų neuronais ar apdorojimo vienetais. Neuroninis tinklas – kaip prisitaikanti mašina – gali būti apibrėžta taip [Hay99]: neuroninis tinklas yra stipriai lygiagretusis išskirstytas procesorius, pagamintas iš paprastų apdorojimo vienetų, kurie turi natūralų polinkį kaupti patirtines žinias ir leisti šias žinias panaudoti. Neuroninis tinklas yra panašus į smegenis dviem atžvilgiais [Hay99]:

1. Žinios yra įgyjamos iš aplinkos mokymosi procese.
2. Tarp neuroninių jungčių stiprumas – sinapsių svoris – yra naudojamas išsaugoti įgyjamoms žinioms.

Mokymo procesui atlikti naudojama procedūra yra vadinama mokymosi algoritmu, kurio funkcija yra metodiškai modifikuoti tinklo sinapsių svorius tam, kad pasiekti norimą modelio tikslą.

1.4.2. Konvoliuciniai neuroniniai tinklai

Tradicinis būdas dviejų dimensijų šablonų atpažinimui yra grindžiamas savybių ištraukėju, kurio išeiga yra paduodama į neuroninį tinklą [SKP10]. Šis savybių ištraukėjas dažnai yra nekintantis ir nėra neuroninio tinklo dalis. Tokiu būdu kuriant savybių ištraukėją, ši užduotis gali pasidaryti labai sudėtinga, nes savybių ištraukėjas nėra neuroninio tinklo dalis ir jis negali adaptuotis prie neuroninio tinklo, kuris dinamiškai kinta jį apmokant.

Konvoliuciniai neuronai tinklai šią sudėtingą problemą pavertė neuroninio tinklo dalimi ir veikia kaip apmokomi savybių ištraukėjai, kurie ištraukia savybes su tam tikra poslinkio,

mastelio ir deformacijos paklaida [LBB+98]. Šie neuroniniai tinklai tą pasiekia naudodami 3 pagrindines architektūrines idėjas:

1. Lokalius jautrumo laukus.
2. Svorių dalinimąsi tarp neuronų.
3. Konvoliucinių sluoksnių rezoliucijų mažinimą.

KNT buvo naudojamas atpažinti eismo dalyvius ir kitus potencialiai pavojingus objektus [CH07]. Autoriai naudojo KNT LeNet [LBB+98]. Šio neuroninio tinklo modifikacijos buvo sėkmingai naudojamos atpažinti įvairiems objektams [LHB04] ar net autonominių robotų kliūčių išvengimo sistemoms kurti [LMB+05].

Tačiau eismo dalyvių atpažinimo modelio vykdymas buvo labai lėtas ir netenkino realaus laiko skaičiavimų reikalavimų [CH07]. Viena iš spartinimo galimybių yra pritaikyti atpažinimo modelį bei KNT veikimui su GP.

1.4.3. Konvoliucinių neuroninių tinklų spartinimas su GP

Per pastaruosius metus, kuomet GP labai ištobulėjo ir tapo kur kas spartesni už CP, daug pastangų buvo dedama, norint pagreitinti neuroninius tinklus su GP. Vieni iš pirmųjų darbų mėgino spartinti neuroninių tinklų klasifikavimo dalį naudodami viršūnių ir pikselių šešėliavimo programas, kiti mėgino paspartinti ir neuroninio tinklo mokymąsi [SKP10]. Pirmoji konvoliucinių neuroninių tinklų GP realizacija pasiekė 4 kart didesnį našumą [CPS06], tačiau ši realizacija naudojo pasenusią viršūnių ir pikselių šešėliavimo architektūrą – buvo pasitelktas DirectX grafikos API.

Atsiradus unifikuotai GP architektūrai bei CUDA, buvo mėginta pritaikyti neuroninius tinklus, panašius į konvoliucinius, ir šioms technologijoms, tačiau buvo mėginama spartinti tik neuroninio tinklo atpažinimo dalį [SKP10].

Nors ir konvoliuciniai neuroniniai tinklai ir yra vienas iš pačių moderniausių būdų atpažinti dvimačius šablonus, tačiau buvo dedama mažai pastangų kuriant lygiagrečiąsias realizacijas. Norint užpildyti šią spragą, buvo siūloma pilna konvoliucinių neuroninių tinklų GP realizacija naudojantis CUDA [SKP10]. Tiek apmokymo, tiek klasifikavimo dalis buvo spartinama, ir buvo pasiektas nuo 2 iki 24 kartų didesnis našumas lyginant su CP. Tačiau didžiausias CUDA technologijos trūkumas yra tai, kad CUDA realizacija veikia tik su NVIDIA technine įranga, ir jos nepavyktų paleisti kompiuteryje, kuriame yra AMD GP ar koks nors kitas procesorius. Tokie standartai, kaip OpenCL, gali padėti išvengti šio prisirišimo prie vieno GP gamintojo, todėl, beveik nekeičiant kodo, būtų galima paleisti OpenCL programą daugelyje šį standartą palaikančių procesorių.

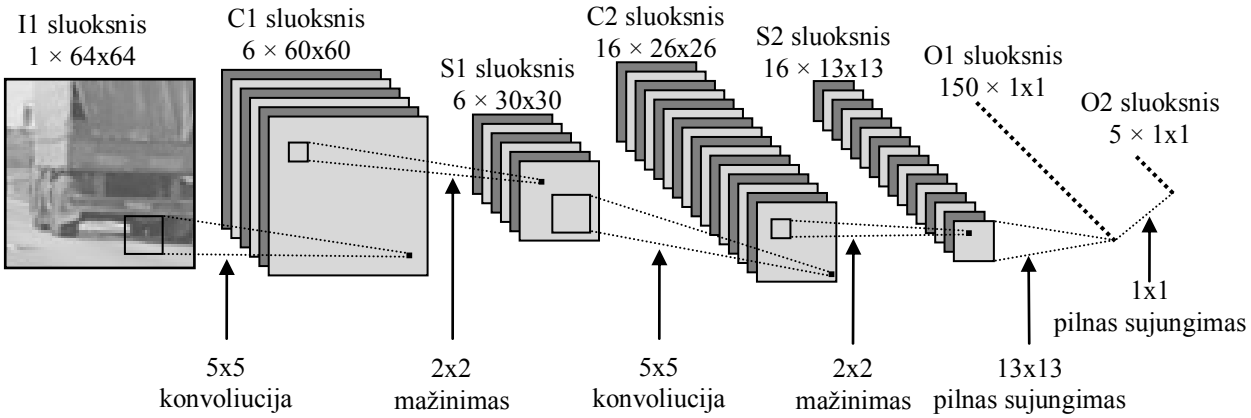
2. KONVOLIUCINIS NEURONINIS TINKLAS EISMO DALYVIAMS ATPAŽINTI

2.1. Konvoliuciniai neuroniniai tinklai

Konvoliuciniai tinklai gali turėti trijų tipų sluoksnius:

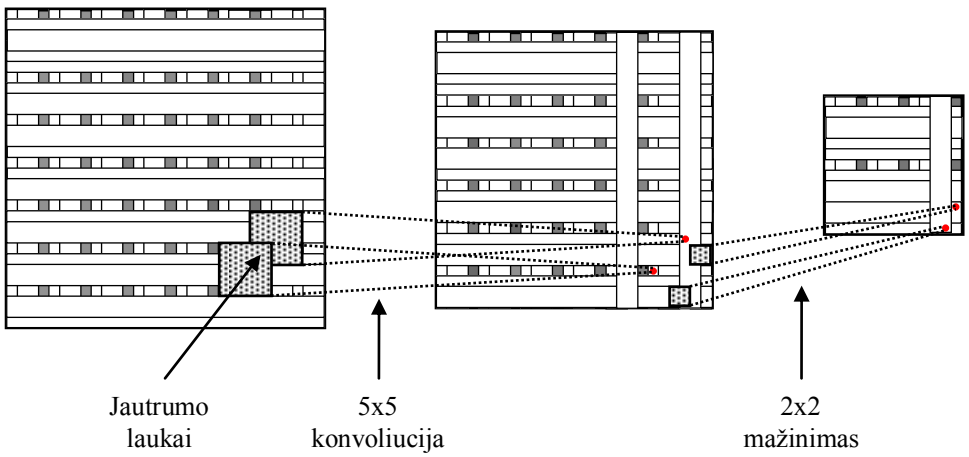
1. Konvoliucinius sluoksnius.
2. Mažinimo sluoksnius.
3. Visiškai sujungtus sluoksnius.

Visi šie sluoksniai yra organizuojami taip, kad rezultatas iš ankstesnio sluoksnio būtų perduodamas į sekantį sluoksnį (4 pav.).



4 pav. Modifikuotas LeNet konvoliucinis neuroninis tinklas, skirtas atpažinti eismo dalyviams keliuose

2.1.1. Konvoliuciniai sluoksniai



5 pav. Konvoliucijos ir mažinimo detalizacija

Konvoliuciniai sluoksniai yra pagrindinė konvoliucinio neuroninio tinklo dalis. Kiekvienas konvoliucinis sluoksnis turi keletą dvimačių plokštumų su neuronais. Šios plokštumos yra vadinamos savybių žemėlapiais. Kiekvienas savybių žemėlapio neuronas yra sujungtas su maža grupe neuronų iš prieš tai buvusio sluoksnio. Ši neuronų grupė yra vadinama jautrumo lauku (5 pav.). Kiekvienas neuronas iš to pačio savybių žemėlapio dalinasi tais pačiais svoriais jungdamiesi su neuronais jautrumo lauke iš prieš tai buvusio sluoksnio. Jei, pavyzdžiui, jautrumo lauko dydis yra 5x5, tuomet visi savybių žemėlapio neuronai naudosis tais pačiais 25 svoriais. Savybių žemėlapiai tarp šalia esančių sluoksnių gali būti arba pilnai sujungti, arba dalinai sujungti. Dalinis sujungimas gali būti tiek generuojamas atsitiktinai, tiek apibrėžiamas iš anksto.

Kuomet yra atliekamas skaičiavimas, pirmiausiai yra suskaičiuojamos konvoliucijos tarp kiekvieno įeities savybių žemėlapio ir atitinkamo neurono sumuojant įeities reikšmes iš jautrumo laukų. Jautrumo lauke kiekviena jungtis tarp neuronų turi svorį, todėl kiekviena įeiga prieš atliekant sumavimą yra padauginama iš šio svorio. Prie sumos tuomet yra pridedama paklaida, kuri taip pat gali būti keičiama (apmokoma) neuroninio tinklo mokymo procese. Tuomet rezultatas yra paduodamas aktyvavimo funkcijai (pavyzdžiui, tanh).

2.1.2. Mažinimo sluoksniai

Savybių žemėlapių sumažinimui yra naudojami mažinimo sluoksniai, kurie dažniausiai yra naudojami tarp dviejų konvoliucinių sluoksnių. Savybių mažinimo sluoksnis sumažina savybių žemėlapių iš prieš tai buvusio sluoksnio neuronų skaičių, apjungdami gretimus neuronus į vieną neuroną (paprastai naudojant 2x2 kvadratą, šie kvadratai nepersidengia). Po to rezultatas yra padauginamas iš svorio, pridedama paklaida ir rezultatas yra paduodamas aktyvavimo funkcijai. Mažinimo sluoksniai turi tokį patį savybių žemėlapių skaičių kaip ir prieš tai buvęs sluoksnis ir kiekvienas savybių žemėlapis yra jungiamas tik su atitinkamu savybių žemėlapiu ir ankstesnio sluoksnio. Yra naudojamas tik 1 su 1 jungimas.

2.1.3. Visiškai sujungti sluoksniai

Po konvoliucinių ar mažinimo sluoksnių būna vienas ar daugiau visiškai sujungtų sluoksnių. Šie sluoksniai visuomet privalo būti naudojami, nes jie atlieka klasifikavimą. Paskutinis konvoliucinio neuroninio tinklo sluoksnis turi tiek neuronų, kiek įvairių skirtingų klasių norima atpažinti šiuo neuroniniu tinklu. Šiuose sluoksniuose kiekvienas neuronas yra sujungiamas su kiekvienu ankstesniame sluoksnyje esančiu neuronu.

2.2. Modifikuotas LeNet konvoliucinis neuroninis tinklas eismo dalyviams atpažinti

Šio darbo vienas iš tikslų yra sukurti konvoliucinį neuroninį tinklą, kuris gebėtų atpažinti eismo dalyvius kelyje. Kuomet norima suprojektuoti ir sėkmingai apmokyti konvoliucinį neuroninį tinklą, reikia priimti daugybę sprendimų, kurie gali įtakoti ar konkretus konvoliucinis neuroninis tinklas galės būti sėkmingai apmokytas ir ar jis veiks. Reikia gerai apgalvoti kiek ir kokių tipų sluoksnių naudoti, kokius mokymo koeficientus naudoti, kokio dydžio turi būti jautrumo laukai, kiek savybių žemėlapių turėtų būti kiekviename sluoksnyje, kokius mokymo ir testavimo duomenis naudoti ir pan. Konvoliucinio neuroninio tinklo sėkmingas veikimas labai nuo to priklauso. Tačiau nėra jokių metodų, kurie griežtai pasakytų kaip konvoliucinį neuroninį tinklą reikėtų suprojektuoti. Viskas labai priklauso nuo situacijos, kam jie bus naudojami. Jei pridėsime per daug sluoksnių ar per daug savybių žemėlapių, neuroninis tinklas išsipūs ir bus lėtas, reikalaus daugiau atminties. Jei bus per mažai savybių žemėlapių ir sluoksnių, tuomet tokio konvoliucinio neuroninio tinklo negalėsime sėkmingai apmokyti. Šis konvoliucinis neuroninis tinklas, skirtas eismo dalyviams atpažinti, buvo sukurtas eksperimentuojant bei skaitant kitų autorių straipsnius, atkreipiant dėmesį į tai, kokie konvoliuciniai neuroniniai tinklai buvo naudojami.

Šis konvoliucinis neuroninis tinklas yra skirtas atpažinti 4 eismo dalyvių klases: pėsčiuosius, motociklininkus, lengvuosius automobilius ir sunkvežimius. Taigi, turime 4 klases. Kuomet yra bandoma iš vaizdo atpažinti objektus, visų pirma labai svarbu yra suklasifikuoti visą vaizdą į dvi pradines klases – fono ir ne fono, kur ne fono klasė gali būti bet kurios kitos objektų klasės, kurias yra norima atskirti. Taigi, prie jau anksčiau minėtų 4 klasių prisideda dar viena kasė – fonas. Neuroninis tinklas eismo dalyviams atpažinti paskutiniame pilnai sujungtame sluoksnyje turi turėti 5 neuronus – po vieną kiekvienai klasei.

Eismo dalyviams atpažinti skirtas neuroninis tinklas yra grindžiamas LeNet neuroniniu tinklu, kuris buvo sėkmingai panaudotas ranka rašytų skaitmenų atpažinimui [LBB+98]. Modifikuotas LeNet neuroninis tinklas yra sudarytas iš 7 sluoksnių (4 pav.):

1. **I1 įeitis.** Šiame sluoksnyje neuroniniui tinklui yra paduodamas vaizdas, kurį neuroninis tinklas mėgins atpažinti. Naudojama 64x64 rezoliucija [CH07]. Tokios įeities rezoliucijos reikia norit apdoroti didesnės rezoliucijos vaizdus. Mažesnės rezoliucijos įeitis apimtų mažesnį plotą vaizde ir to gali nepakakti, o didesnės rezoliucijos įeitis reikalautų daugiau skaičiavimų arba į vaizdą galėtų patekti keli objektai, kuriuos reikia suklasifikuoti. Yra tik vienas įeities žemėlapis.
2. **C1 konvoliucinis sluoksnis.** Šis sluoksnis turi 6 savybių žemėlapius, kurių dydis yra 60x60. Naudojamas 5x5 konvoliucijos kernelis. I1 sluoksnyje yra tik vienas

savybių žemėlapis, todėl šis sluoksnis iš viso turi 21600 neuronų, 561600 jungčių ir 156 apmokomų parametrų.

3. **S1 mažinimo sluoksnis.** Šis sluoksnis turi lygiai tiek savybių žemėlapių kiek ir C1 sluoksnis – 6. Naudojamas 2x2 mažinimo kernelis, todėl savybių žemėlapių dydis yra 30x30. Šis sluoksnis turi 5400 neuronų, 27000 jungčių ir 12 apmokomų parametrų. Šio sluoksnio savybių žemėlapiai yra sujungti tik su atitinkamais savybių žemėlapiais iš C1 sluoksnio (1 su 1 sujungimas).
4. **C2 konvoliucinis sluoksnis.** Šis sluoksnis yra sudarytas iš 16 savybių žemėlapių. Naudojamas 5x5 konvoliucijos kernelis, todėl kiekvieno savybių žemėlapio dydis yra 26x26. Šio sluoksnio savybių žemėlapiai yra pilnai sujungti su S1 sluoksnio savybių žemėlapiais, todėl šis sluoksnis turi 1633216 jungčių, 10816 neuronų ir 2416 apmokomų parametrų.
5. **S2 mažinimo sluoksnis.** Šis sluoksnis turi 16 savybių žemėlapių. Naudojamas 2x2 mažinimo kernelis, todėl savybių žemėlapių dydis yra 13x13. Šio sluoksnio savybių žemėlapiai yra sujungti tik su atitinkamais savybių žemėlapiais iš C2 sluoksnio (1 su 1 sujungimas), todėl šis sluoksnis turi 13520 jungčių, 2704 neuronų ir 32 apmokomus parametrus.
6. **O1 sluoksnis.** Šis sluoksnis turi 150 savybių žemėlapių. Yra naudojamas 13x13 konvoliucijos kernelis, todėl savybių žemėlapių dydis yra 1x1. Yra pilnai sujungtas su S2 sluoksniu, todėl turi 405750 jungčių, 150 neuronų ir 405750 apmokomų parametrų.
7. **O2 sluoksnis.** Šis sluoksnis yra pilnai sujungtas su O1 sluoksniu ir turi 755 jungtis, 755 apmokomus parametrus ir 5 savybių žemėlapius, kurie kiekvienas reprezentuoja įeigos panašumo laipsnį į konkrečią bandomą atpažinti klasę. Savybių žemėlapių dydis yra 1x1, todėl kuo didesnė yra konkrečios klasės reikšmė, tuo įeiga yra panašesnė į konkrečią ieškomą klasę. Šio darbo atveju, tai:
    a. 1 savybių žemėlapis yra skirtas fonui atpažinti.
    b. 2 savybių žemėlapis yra skirtas sunkvežimiams atpažinti.
    c. 3 savybių žemėlapis yra skirtas motociklininkams atpažinti.
    d. 4 savybių žemėlapis yra skirtas lengviesiems automobiliams atpažinti.
    e. 5 savybių žemėlapis yra skirtas pėstiesiems atpažinti.

2.3. Apmokymas

Tam, kad būtų galima tinkamai kurti bei testuoti konvoliucinį neuroninį tinklą, reikėjo pradinių duomenų, kuriais būtų galima neuroninį tinklą apmokyti ir testuoti. Buvo nufilmuota

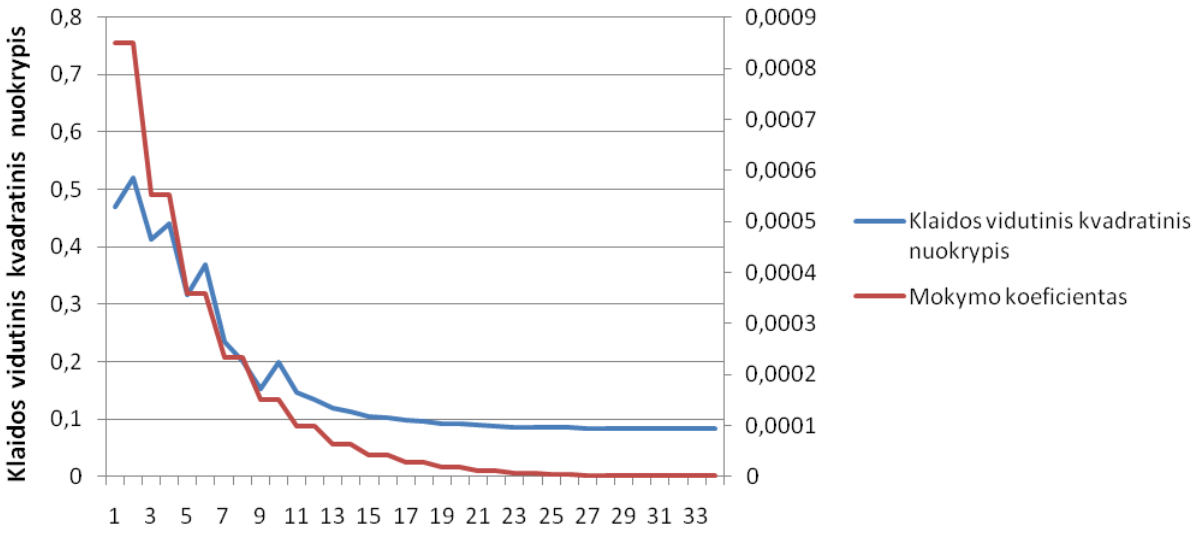
serija stereo video įrašų automagistralėje Vilnius-Kaunas-Klaipėda. Šie duomenys ir buvo naudojami konvoliuciniam neuroniniam tinklui apmokyti ir testuoti.

Neuroninio tinklo apmokymui buvo panaudotas atgalinio propagavimo *Stochastic Diagonal Levenberg–Marquardt* metodas [LBB+98], kurio metu apmokymas yra vykdomas pradedant nuo paskutinio sluoksnio ir baigiant pirmuoju. Šio proceso metu yra koreguojamos mokomų parametrų (svorių ir paklaidos) reikšmės, taip, kad neuroninis tinklas po truputi gebėtų atskirti klases.

Apmokymui buvo panaudoti paveikslėliai gauti iš stereo video įrašų. Buvo nufilmuoti sunkvežimiai bei lengvieji automobiliai. Iš viso buvo panaudota:

- 1602 fono paveikslėlių.
- 606 sunkvežimių paveikslėlių.
- 564 lengvųjų automobilių paveikslėlių.

Sunkvežimių ir lengvųjų automobilių paveikslėliai buvo dubliuojami keletą kartų, nes fono paveikslėlių buvo kur kas daugiau, todėl neuroninis tinklas negalėjo pakankamai greitai išmokti atskirti kitų klasių.



6 pav. Konvoliucinio neuroninio tinklo mokymo eiga

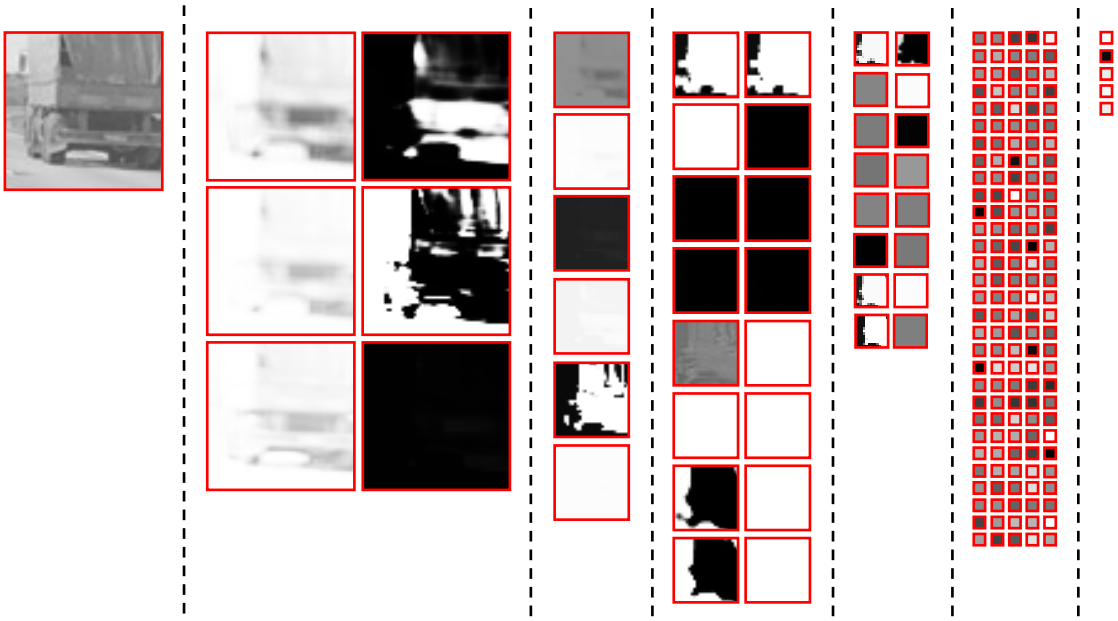
5 lentelėje galime pamatyti apmokymo proceso eigą. Apmokymo procesas užtruko 34 iteracijas. Mokymo koeficientas buvo po truputi mažinamas tam, kad neuroninis tinklas sėkmingai mokytųsi (6 pav.). Jei mokymo koeficientas būtų per didelis vėlesnėse iteracijose, tai galėtų lemti klaidų procento augimą vietoj mažėjimo.

5 lentelė. Apmokymo procesas

Iteracija	Klaidos vidutinis kvadratinis nuokrypis	Klaidingai atpažintų paveiksliukų skaičius	Klaidos procentas	Mokymo koeficientas
1	0,4702328	620	22,37%	0,000850000
2	0,5201132	610	22,01%	0,000850000
3	0,4130086	533	19,23%	0,000552500
4	0,4413999	592	21,36%	0,000552500
5	0,3158827	422	15,22%	0,000359125
6	0,3698114	533	19,23%	0,000359125
7	0,235118	303	10,93%	0,000233431
8	0,2003371	263	9,49%	0,000233431
9	0,1530704	177	6,39%	0,000151730
10	0,1982795	248	8,95%	0,000151730
11	0,1474922	167	6,02%	0,000098625
12	0,1339289	157	5,66%	0,000098625
13	0,1189855	131	4,73%	0,000064106
14	0,1130594	125	4,51%	0,000064106
15	0,105036	112	4,04%	0,000041669
16	0,1030617	113	4,08%	0,000041669
17	0,0977121	109	3,93%	0,000027085
18	0,0957753	110	3,97%	0,000027085
19	0,09275129	103	3,72%	0,000017605
20	0,09182656	103	3,72%	0,000017605
21	0,08927735	105	3,79%	0,000011443
22	0,08822068	102	3,68%	0,000011443
23	0,08661175	102	3,68%	0,000007438
24	0,08583189	103	3,72%	0,000007438
25	0,08559904	102	3,68%	0,000004835
26	0,0849477	103	3,72%	0,000004835
27	0,08446866	103	3,72%	0,000003143
28	0,0842115	104	3,75%	0,000003143
29	0,08350588	102	3,68%	0,000002043
30	0,08361492	101	3,64%	0,000002043
31	0,08332893	101	3,64%	0,000001328
32	0,08317942	101	3,64%	0,000001328
33	0,08309883	102	3,68%	0,000001000
34	0,08293314	101	3,64%	0,000001000

7 pav. matome konvoliucinio neuroninio tinklo sluoksnių skaičiavimo rezultatus (savybių žemėlapius). Kiekvienas konvoliucinio neuroninio tinklo sluoksnis yra atskirtas punktyrine linija. Pirmasis (įeities) sluoksnis yra pavaizduotas kairėje pusėje. Toliau iš kairės į dešinę pavaizduoti antrasis, trečiasis, ketvirtasis, penktasis bei šeštasis sluoksniai. Paskutinis (rezultatų) sluoksnis yra pavaizduotas dešinėje pusėje. Po apmokymo, neuroninis tinklas sėkmingai

suklasifikavo sunkvežimio paveiksliuką – 7 sluoksnyje 2 savybių žemėlapyje matome didžiausią (ryškiausią) reikšmę.



7 pav. Konvoliucinio neuroninio tinklo sluoksnių skaičiavimo rezultatų pavaizdavimas

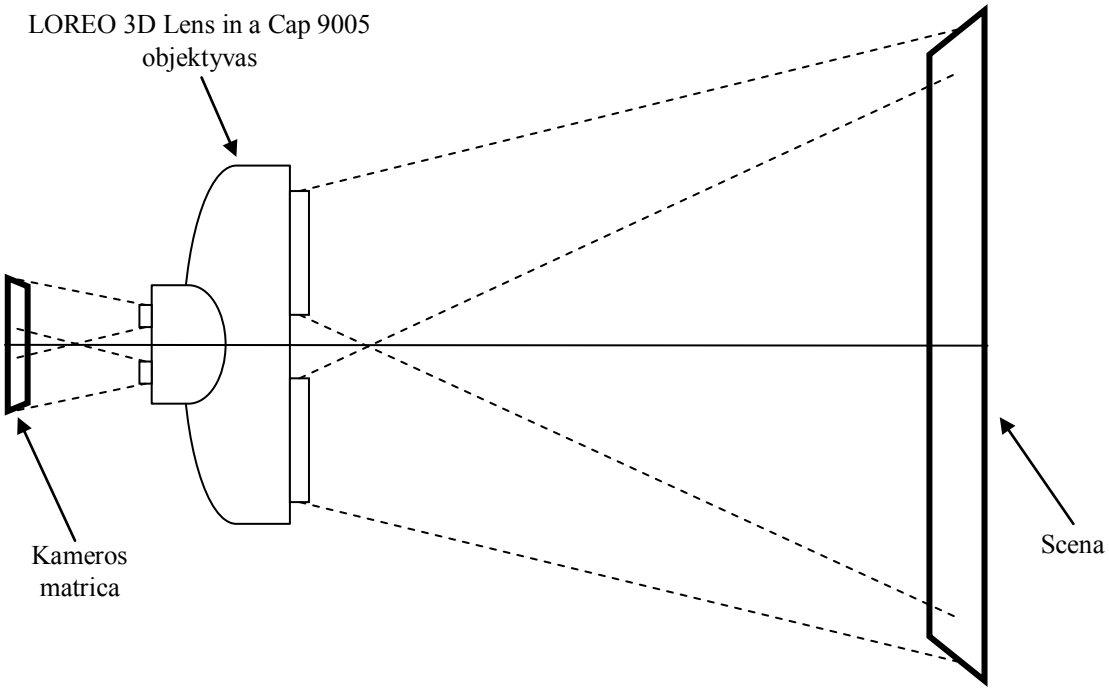
3. EISMO DALYVIŲ ATPAŽINIMAS

3.1. Stereo vaizdų gavimas

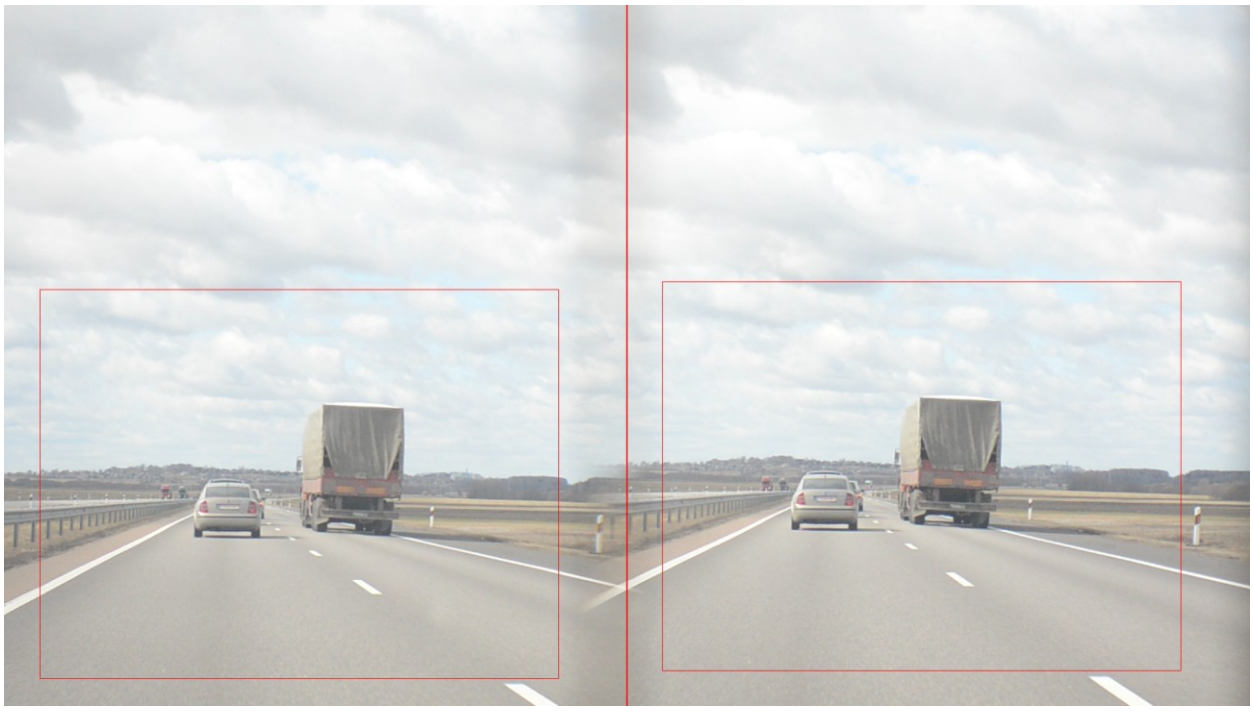
Stereo vaizdams gauti buvo naudojamas Nikon D7000 fotoaparatas ir Loreo 3D Lens in a Cap 9005 objektyvas (8 pav.). Stereo video medžiaga buvo filmuojama automagistralėje Vilnius-Kaunas-Klaipėda. Fotoaparatas buvo pritvirtintas prie trikojo, kuris buvo padėtas mašinos priekyje, priešais keleivio sėdynę ir pritvirtintas.

8 pav. Stereo filmavimui naudotas Nikon D7000 fotoaparatas ir Loreo 3D Lens in a Cap 9005 objektyvas

Loreo 3D stereo objektyvas užsideda ant fotoaparato kaip ir visi kiti paprasti objektyvai, tačiau skirtumas yra tas, kad šis objektyvas vietoj vieno vaizdo fiksuoja iškarto du ir fokusuoja šiuos du vaizdus ant matricos kairės ir dešinės pusės (9 pav.)



9 pav. Scenos projektavimo į Nikon D7000 fotoaparato matricą schema



10 pav. Stereo vaizdas bei raudonai pažymėti kairiojo ir dešiniojo kadro regionai, kurie buvo naudojami nesugretinamumo žemėlapio skaičiavimui

Filmuojama buvo naudojant 1920x1080 rezoliuciją, 24 kadrų per sekundę greičiu. Paprastai stereo vaizdams gauti yra naudojamos dvi kameros, kurios privalo būti sukalibruotos. Tai sukelia daug problemų, nes kameros privalo būti nukreiptos į priekį tokiu pačiu kampu, privalo būti pritvirtintos viena prie kitos, kad nepriklausomai nejudėtų, kamerų matricų parametrai turi būti suderinti tarpusavį, taip pat kadrai turi būti sinchronizuojami. Turint objektyvą, kuris gali fokusuoti iškarto du vaizdus ant tos pačios matricos, šių problemų nelieka.

10 pav. matome scenos pavyzdį, nufilmuotą su stereo objektyvu. Raudona linija centre žymi, kur kairysis vaizdas pasibaigia, o kur dešinysis prasideda. Kadangi vaizdas yra fokusuojamas ant tos pačios matricos, ties vaizdų susikirtimo vieta atsiranda artefaktai, t.y. kadrai persidengia. Tačiau šis vaizdo regionas yra pakankamai nedidelis. Raudoni stačiakampiai žymi, kairįjį ir dešinįjį vaizdus, kurių kiekvieno rezoliucija yra 800x640. Ši rezoliucija yra proporcinga paveiksliukų rezoliucijoms, kurie yra apdorojami su konvoliuciniu neuroniniu tinklu, tam, kad pradinį vaizdą būtų galima mažinti jo neiškraipant. Norint gauti nesugretinamumo žemėlapį, reikia šiuos vaizdus iškirpti iš bendro vaizdo. Galime pastebėti, kad dešiniojo vaizdo stačiakampis yra šiek tiek aukščiau. Taip yra dėl to, kad naudoto Loreo 3D objektyvo dešinysis veidrodis fokusuoja vaizdą šiek tiek aukščiau. Tačiau šis nedidelis defektas yra stabilus, todėl buvo galima nuolatos iškirpti vaizdą iš tos pačios vietos.

Atstumas tarp kairiojo ir dešiniojo Loreo 3D stereo objektyvo veidrodžių yra 9 centimetrai. To pakanka, kad, skaičiuojant nesugretinamumo žemėlapį, būtų galima užfiksuoti pakankamai toli esančius objektus.

3.2. Pradinis kadro apdorojimas

Modifikuotas LeNet konvoliucinis neuroninis tinklas kaip įeitį priima 64x64 dydžio paveiksliuką. Tačiau visa greitkelio su eismo dalyviais scena tikrai negali sutilpti į tokį mažą paveiksliuką, be to konvoliucinis neuroninis tinklas pateikia tik vieną atsakymą – konvoliucinis neuroninis tinklas negali atpažinti iš karto dviejų klasių objektų. Konvoliucinio neuroninio tinklo užduotis yra klasifikuoti pateiktus paveiksliukus, nustatant kuriai paveiksliukų klasei pateiktas paveiksliukas priklauso. Jokių kitų užduočių konvoliucinis neuroninis tinklas neatlieka.

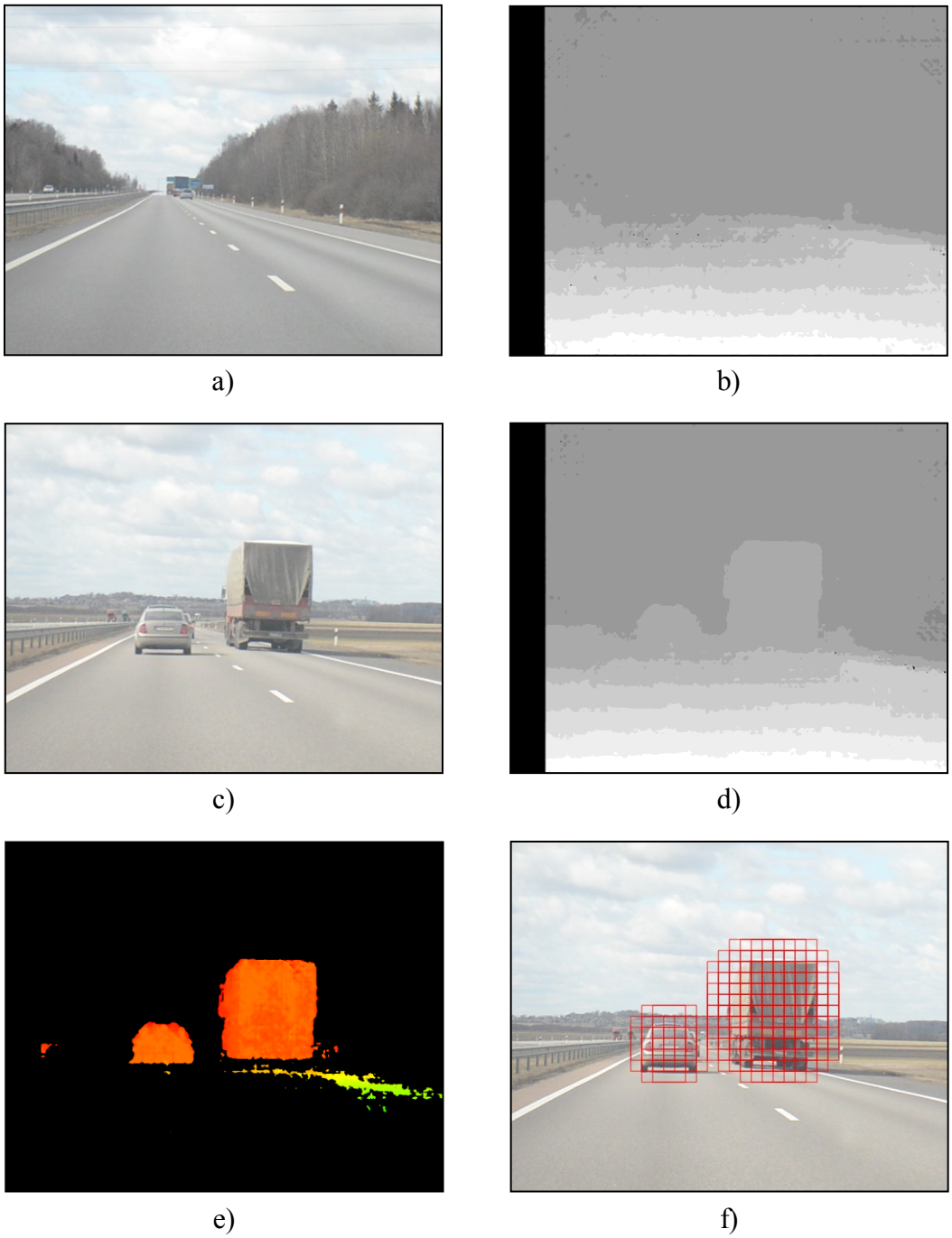
12 pav. matome eismo dalyvių atpažinimo vaizde modelį. Kuomet sistema gauna neapdorotą vaizdą, pirmiausiai reikia išskirti iš šio vaizdo kairįjį ir dešinįjį vaizdus. Šie vaizdai yra iškerpami pagal iš anksto numatytas koordinates pradiniame vaizde.

Tuomet kairysis ir dešinysis vaizdai būna apdorojami OpenCV biblioteka ir gaunamas nesugretinamumo žemėlapis (11 pav. d paveikslas). 11 pav. a ir b paveiksluose matome tuščią sceną ir atitinkamą nesugretinamumo žemėlapį. Galime pastebėti, kad kelias yra žymimas baltai, o fonas tamsiai pilkai. Nesugretinamumo žemėlapyje kuo objektas yra arčiau, tuo balčiau jis yra žymimas. Todėl matome tolygų perėjimą nuo balto iki pilko fono, kur pilkas fonas užima daugiau kaip pusę scenos. Kiekviena nesugretinamumo žemėlapio eilutė tuščioje scenoje turi labai panašias reikšmes. Galime suskaičiuoti vidutinį atstumą kiekvienoje eilutėje ir po to, šias reikšmes naudoti išskiriant iškilius objektus kituose vaizduose. Kad neišskirti nereikalingų regionų ir išskirtume iškilius objektus tiksliau, galime įsivesti atstumo slenkstinę reikšmę kiekvienoje nesugretinamumo žemėlapio eilutėje, kuri priklauso nuo anksčiau suskaičiuotų vidurkių tuščioje scenoje (pvz. kad konkretus pikselis būtų registruojamas kaip iškiliojo objekto pikselis, jo atstumas turi būti bent 5% mažesnis nei tos eilutės tuščios scenos vidutinis atstumas). 11 pav. e paveiksle pavaizduoti aptikti iškilieji objektai (iškilieji pikseliai žymimi spalvomis nuo žalios iki raudonos, kur žalia yra arčiausiai kameros, o raudona – toliausiai). Keičiant slenkstines reikšmes, galime išskirti daugiau arba mažiau iškilių pikselių, tačiau jei nenaudotume slenkstinės reikšmės, tuomet būtų išskiriama daug neteisingų regionų vaizde. Taip atsitiktų todėl, kad nesugretinamo žemėlapis gali turėti šalutinio triukšmo ar kai kuriems monotoniškiems regionams negali būti nustatytas atstumas arba jis yra nustatomas blogai.

Tam, kad atpažinti tiek toliau važiuojančius eismo dalyvius, tiek arčiau važiuojančius, pradinį paveiksliuką galima nuskaityti dviem rezoliucijomis [CH07]: 320x256 ir 640x512. Galime pasirinkti ir kitokias rezoliucijas ar daugiau rezoliucijų, tačiau kiekvieno sukurto

paveikslo kiekvienos dimensijos dydis turi būti dalus iš slenkančio langelio poslinkio žingsnio, kad galėtume suformuoti bendrą klasių intensyvumo žemėlapį (13 pav. e paveikslas).

Toliau kiekvienas paveiksliukas yra suskaldomas į 64x64 dydžio paveiksliukus, kurie yra tinkamo dydžio, kad juos būtų galima paduoti konvoliuciniam neuroniniui tinklui. 320x256 paveiksliukas yra skaldomas su 8 pikselių žingsniu, o 640x512 paveiksliukas yra skaldomas 16 pikselių žingsniu. Jei nenaudotume nesugretinamumo žemėlapio ir neišskirtume potencialių regionų, tuomet po suskaldymo turėtume iš viso 1898 64x64 paveiksliukų, kuriuos visus reikėtų suklasifikuoti ir nustatyti, kurioje pozicijoje yra koks objektas. Tačiau galime atmesti tuos 64x64 paveiksliukus, kurie neturi iškiliųjų pikselių arba turi nepakankamą procentą. Galime įsivesti slenkstinę reikšmę, kad 64x64 paveiksliukas yra paduodamas konvoliuciniam neuroniniam tinklui tik tuomet, kuomet tam tikras procentas pikselių yra iškilūs (pvz. 50% pikselių turi būti aptikti nesugretinamumo žemėlapio kaip iškilūs). 11 pav. f paveiksle matome kas 16 pikselių persidengiančius 64x64 raudonus kvadratus, kurie žymi vaizdo regionus, kurie bus paduodami konvoliuciniam neuroniniam tinklui apdoroti. Vietoj 1898 paveiksliukų, atmetus neiškilius vaizdo regionus, liko tik 150 64x64 paveiksliukų, kas yra 92% mažiau. Žinoma, jei slenkstinė reikšmė būtų didesnė, tuomet turėtume dar mažiau paveiksliukų, o jei slenkstinė reikšmė būtų mažesnė – turėtume daugiau.



11 pav. Nereikalingų vaizdo regionų atmetimas naudojant nesugretinamumo žemėlapį

3.3. Neuroninio tinklo rezultatų skaičiavimas pasitelkus GP

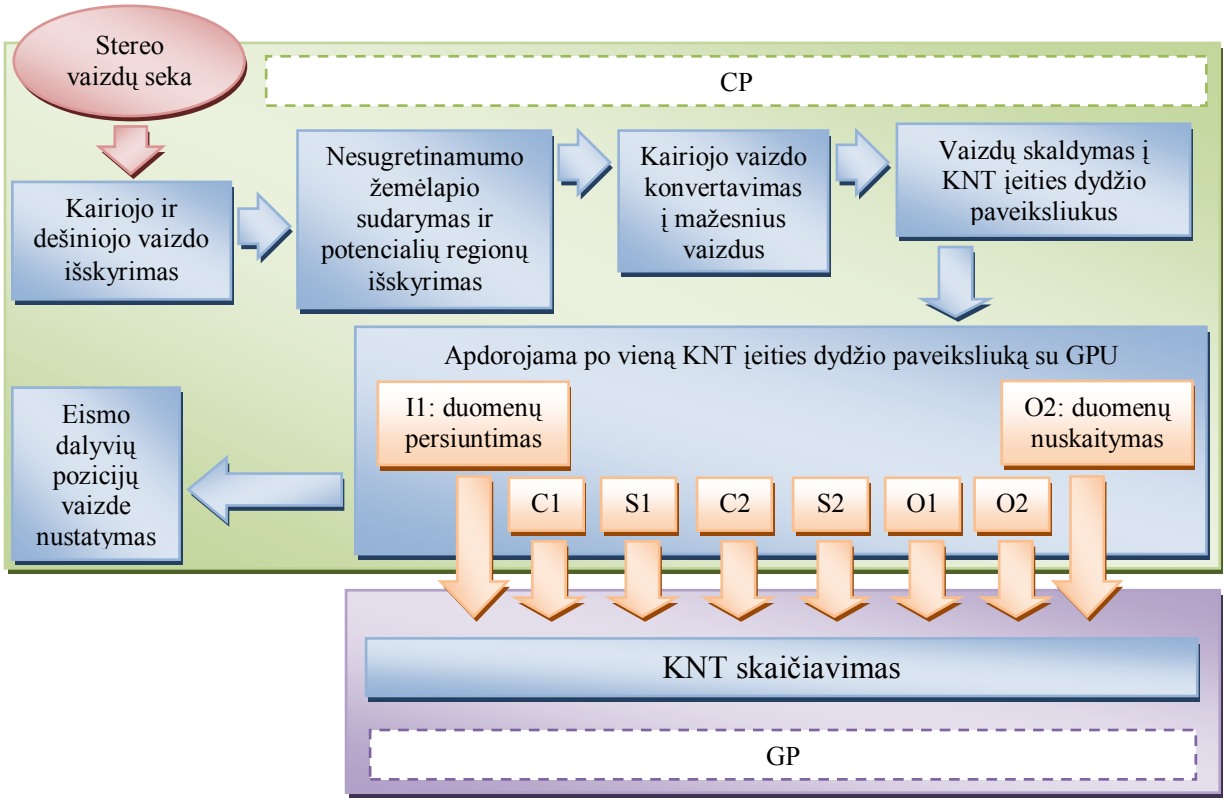
Modifikuotas LeNet konvoliucinis neuroninis tinklas eismo dalyviams atpažinti turi pakankamai didelę struktūrą, daug jungčių. Todėl suskaičiuoti neuroninio tinklo rezultatą su CP užtrunka šiek tiek laiko. Galima mėginti spartinti procesą pasitelkus grafikos procesorius.

Yra keletas būdų, kaip galima rašyti programas, kurios išnaudotų grafikos procesoriaus pajėgumus. Galima taikyti tradicinį būdą naudojant grafikos bibliotekų API bei rašyti šešėliavimo programas. Tačiau tai jau pasenęs ir nepatogus būdas. Konvoliucinio neuroninio tinklo sluoksnių rezultatams suskaičiuoti pasitelkta OpenCL [Khr10b].

Eismo dalyvių atpažinimo proceso modelis pateiktas 12 pav. Visų pirma paleidus sistemą, apmokyto neuroninio tinklo duomenų struktūros yra nusiunčiamos į GPU operatyviąją atmintį, nes šie duomenys yra statiniai ir nesikeičia sistemos vykdyme. Gavus pradinį paveikslių bei atlikus pradinio kadro apdorojimo žingsnius, gaunami 64x64 paveiksliukai, kuriuos reikia suklasifikuoti. Šie 64x64 paveiksliukai yra paruošiami apdorojimui naudojant grafikos procesorių su iš anksto apmokytu konvoliuciniu neuroniniu tinklu. 64x64 paveiksliukai yra apdorojami po vieną:

1. Iš pradžių yra persiunčiami paveikslio duomenys į I1 neuroninio tinklo sluoksnį.
2. Tuomet paeiliui po vieną yra kviečiama atskira OpenCL programa kiekvienam sluoksniui apdoroti. Prieš paleidžiant sekančią OpenCL programą, yra laukiama kol pasibaigs prieš tai ėjusi.
3. Nuskaitomi O2 sluoksnio duomenys iš grafikos procesoriaus. O2 sluoksnis yra sudarytas iš 5 neuronų, kurie identifikuoja eismo dalyvio klasę arba foną.

Apdorojus visus 64x64 paveiksliukus, sekantis žingsnis yra rezultatų apdorojimas – eismo dalyvių pozicijų vaizde nustatymas.



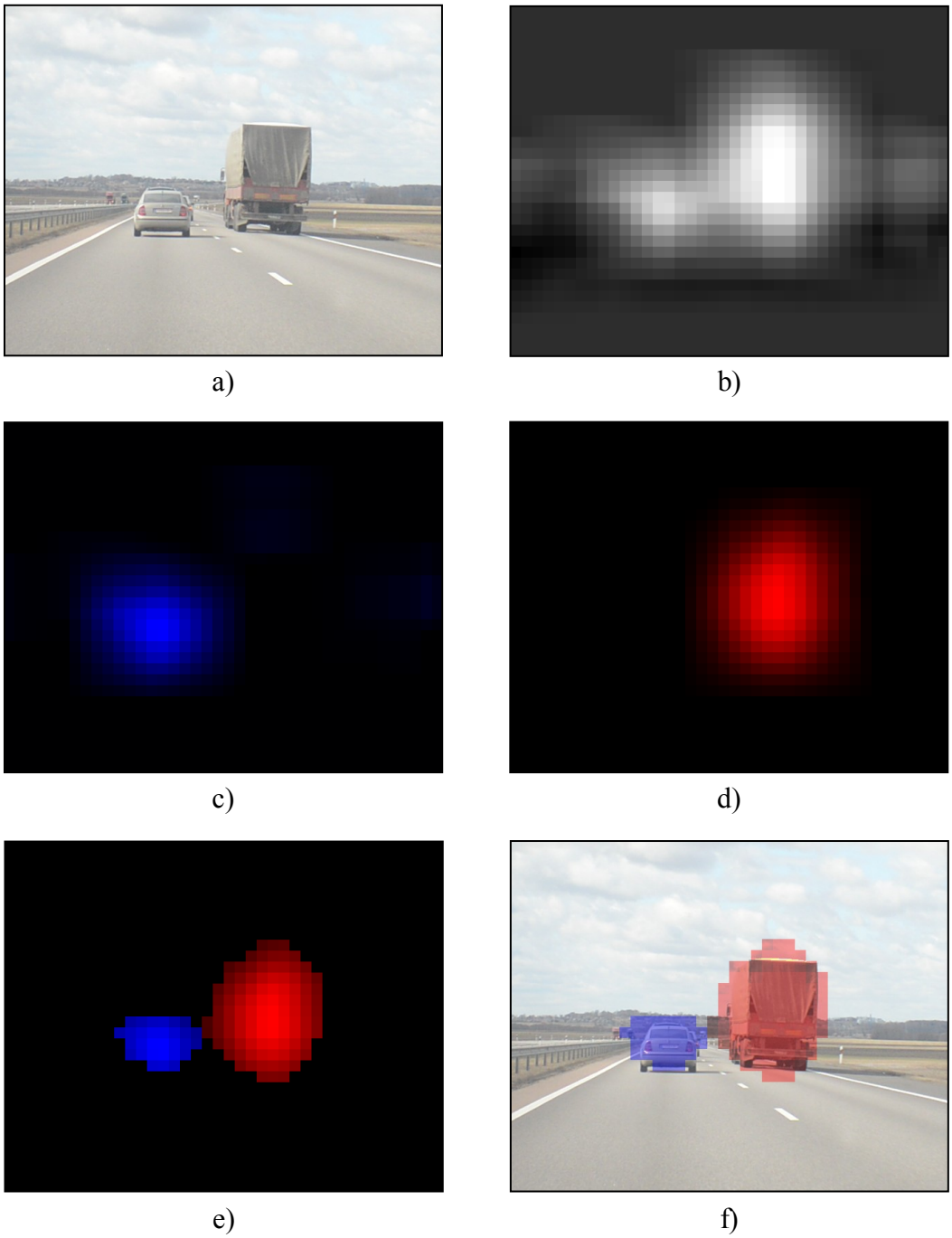
12 pav. Eismo dalyvių atpažinimo analizuojant vaizdą modelis

3.4. Rezultatų apdorojimas bei eismo dalyvių pozicijos vaizde nustatymas

Kaip minėjau, vaizdas yra apdorojamas dviem rezoliucijomis (320x256, 640x512) ir naudojamas 64x64 slenkantis langas paveiksliukams gauti. Nuskaičius 320x256 vaizdą su 8

pikseliais slenkančiu 64x64 langu, gauname 33x25 dydžio matricą, o nuskaičius 640x512 vaizdą su 16 pikseliais slenkančiu 64x64 langu, gauname 37x29 dydžio matricą. Kiekvienas elementas šiose matricose saugo atsakymą gautą iš neuroninio tinklo apie tai, kokia klasė buvo aptikta šioje pozicijoje ir kokia intensyvumo reikšmė buvo gauta. Tokiu būdu galime gauti konkrečios klasės intensyvumą konkrečioje pozicijoje ir pasinaudojus šia informacija suformuoti klasių pasirodymo žemėlapius (13 pav.):

a. Tai – pradinis vaizdas, kuris būna apdorojamas 320x256 ir 640x512 rezoliucijomis bei išskirti regionai suskaldomi į 64x64 dydžio paveiksliukus.
b. Kuomet neuroninis tinklas apdoroja visus paveiksliukus ir gražina rezultatus, yra suformuojamas fono ir ne-fono žemėlapis. Tai yra padaroma tiesiog stebint pirmos (fono) klasės intensyvumą kiekviename paveikslėlio pikselyje. Kadangi paveikslėliai yra nuskaitomi su 16 pikselių poslinkiu, todėl ir intensyvumo žemėlapis nėra tolygus, o susiskirstęs 16x16 kvadratėliais. Kuo tamsesnis kvadratėlis, tuo fono klasės intensyvumas yra didesnis.
c. Kaip ir fono klasės atveju, yra sudaromas ir lengvųjų automobilių klasės intensyvumo žemėlapis. Vaizdumo dėlei, lengvųjų automobilių klasės intensyvumas yra žymimas mėlyna spalva.
d. Sunkvežimių klasės intensyvumas yra žymimas raudona spalva. Raudona spalva parinka vaizdumo dėlei. Spalvos gali būti keičiamos.
e. Sudėjus visas klases į vieną bendrą žemėlapį bei palyginus kiekvienos klasės intensyvumą visose pozicijose ir palikus tik dominuojančią klasę konkrečioje pozicijoje, gaunamas bendras klasių intensyvumo žemėlapis, kuris sumažina kiekvienos klasės užimamą plotą žemėlapyje ir tiksliau nurodo poziciją.
f. Sujungus pradinį vaizdą su bendru klasių žemėlapiu (iš kurio yra pašalinama fono klasė), yra sužymimos eismo dalyvių klasės, ir parodomos jų pozicijos vaizde.



13 pav. Eismo dalyvių atpažinimas naudojant konvoliucinį neuroninį tinklą ir GPU

3.5. Sistemos greitaveikos analizė

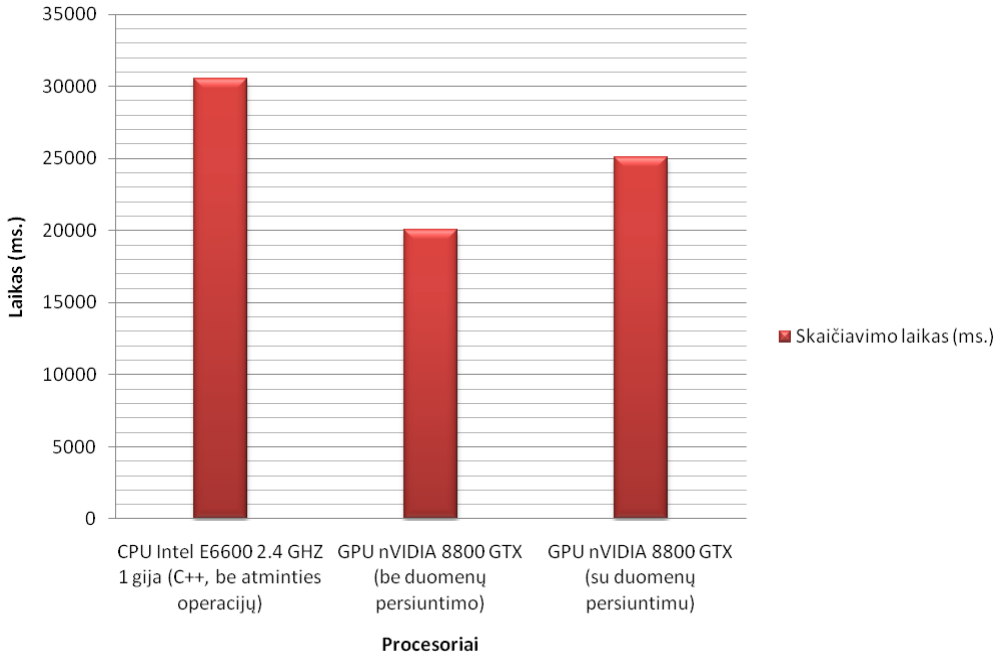
Eismo dalyvių atpažinimo analizuojant vaizdą problema yra iš ties sudėtinga. Tačiau pasitelkus konvoliucinius neuroninius tinklus galime apmokyti šį neuroninį tinklą klasifikuoti objektus vaizde. Norint apdoroti didesnės rezoliucijos vaizdus, reikia, kad ir neuroninis tinklas galėtų apdoroti didesnės rezoliucijos įeitį. 64x64 įeitį turintis neuroninis tinklas reikalauja labai daug resursų, o blogiausiu atveju gali tekti apdoroti net 1898 64x64 paveiksliukus, gautus iš vieno vaizdo.

Pasiūlytas eismo dalyvių atpažinimo modelis suteikia galimybę sparčiau apdoroti vaizdą su GP. Yra naudojamas OpenCL standartas, o tai leidžia sistemą naudoti ne vien su nVIDIA grafikos procesoriais, bet ir su AMD grafikos procesoriais ar kitais procesoriais, kurie palaiko OpenCL. Naudojant GP konvoliucinio neuroninio tinklo rezultatams suskaičiuoti, CP lieka laisvas ir gali tuo metu atlikti kitas užduotis. Taip pat pasiūlytas būdas kaip galima išskirti potencialius vaizdo regionus pasinaudojus nesugretinamumo žemėlapiais. Išskyrus potencialius pradinio vaizdo regionus, kadro apdorojimo greitis išauga, priklausomai nuo to, kiek procentų pradinio vaizdo buvo atmesta.

6 lentelė. CP ir GP greičių palyginimas skaičiuojant 1898 įeitis

Procesorius	**Skaičiavimo laikas (milisekundės)**
CP Intel E6600 2.4 GHZ 1 gija (C++, be atminties operacijų)	31248.72
GP nVIDIA 8800 GTX OpenCL (be duomenų persiuntimo)	20812,81
GP nVIDIA 8800 GTX OpenCL (su duomenų persiuntimu)	25208.37

6 lentelėje pateikti blogiausio atvejo testavimo rezultatai, kuomet yra neatmestas nei vienas 64x64 paveiksliukas. 1898 64x64 įeitis centrinis procesorius (1 gija) suskaičiavo per 31,2 sekundes, tačiau centrinis procesorius neatliko jokių atminties operacijų – visos duomenų struktūros buvo iš anksto paruoštos skaičiavimui. Grafikos procesorius visa tai suskaičiavo per 20,8 sekundžių (33,4% greičiau, 14 pav.), tačiau prieš atliekant skaičiavimus, neuroniniam tinklui reikalingi duomenys turi būti nusiunčiami į grafikos procesoriaus operatyviąją atmintį, o tai užtrunka dar papildomas 5 sekundes.



14 pav. CP ir GP skaičiavimo greičių palyginimas

CP Intel E6600 2.4 GHZ (1 gija) vieną 64x64 įeitį vidutiniškai suskaičiuoja per 16,5 milisekundes, o GP nVIDIA 8800 GTX – per 11 milisekundžių. Nesugretinamumo žemėlapio suskaičiavimas iš dviejų 800x640 vaizdų, iškiliųjų pikselių nustatymas ir nereikalingų vaizdo regionų atmetimas vidutiniškai užtrunka apie 1 sekundę su Intel E6600 2.4 GHZ CP. Tačiau atmetus daug 64x64 paveiksliukų, konvoliucinis neuroninis tinklas būna kur kas mažiau apkraunamas. Jei atmetus nereikalingus vaizdo regionus lieka, pavyzdžiui, 150 64x64 paveiksliukų, tuomet šiuos paveiksliukus konvoliucinis neuroninis tinklas suskaičiuos per 2,48 sekundes su Intel E6600 2.4 GHZ CP (1 gija) arba per 1,65 sekundes su nVIDIA 8800 GTX GP. Taigi CP atveju, vietoj 31,2 sekundžių vaizdas yra apdorojamas per 3,48 sekundes, o tai yra beveik 9 kartus greičiau. GP atveju, vietoj 20,8 sekundžių vaizdas yra apdorojamas per 2,65 sekundes, o tai yra beveik 8 kartus greičiau.

Iš rezultatų matyti, kad ši sistema dar toli nuo gebėjimo apdoroti vaizdo srautą realiu laiku. Norint video srautą apdoroti bent 15 kadrų per sekundę greičiu, reikia, kad sistema vieną kadrą apdorotų per 66,67 milisekundes. Buvo galima tikėtis geresnių rezultatų iš GP, tačiau neuroninio tinklo sudėtinga vidinė struktūra reikalauja, kad dideli masyvai būtų globalioje GP operatyviojoje atmintyje, kuri yra lėta, ir nepastovios kreiptys į šią atmintį viską procesą labai sulėtina. Sekančiuose darbuose galima mėginti optimizuoti atminties naudojimą grafikos procesoriuje ir taip sistemą paspartinti labiau. Taip pat galima mėginti sukurti lankstesnę neuroninio tinklo vidinę duomenų struktūrą, kuri labiau išnaudotų grafikos procesoriaus pranašumus.

IŠVADOS IR REZULTATAI

Išvados:

- Pasitelkus grafikos procesorių galima paspartinti konvoliucinius neuroninius tinklus eismo dalyviams klasifikuoti.
- Panaudojant nesugretinamumo žemėlapį bei išskyrus potencialius regionus iš vaizdo galima paspartinti eismo dalyvių atpažinimo procesą.

Rezultatai:

- Sukurtas konvoliucinio neuroninio tinklo modelis, kurio pagalba galima klasifikuoti eismo dalyvius.
- Pasiūlytas būdas, kaip gauti stereo vaizdus nesugretinamumo žemėlapio skaičiavimui.
- Sukurtas eismo dalyvių atpažinimo analizuojant vaizdą metodas pasitelkus konvoliucinius neuroninius tinklus, nesugretinamumo žemėlapį bei grafikos procesorių.
- Atlikta greitaveikos analizė lyginant centrinį procesorių bei grafikos procesorių.

ŠALTINIAI

[Amd10] AMD. ATI Radeon™ HD 5870 GPU Feature Summary. [žiūrėta 2010.06.17]. Prieiga per internetą: <http://www.amd.com/uk/products/desktop/graphics/ati-radeon-hd-5000/hd-5870/Pages/ati-radeon-hd-5870-specifications.aspx>

[BF10] J.Breitbart, C.Fohry. OpenCL – An effective programming model for data parallel computations at the Cell Broadband Engine. IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, 19-23 April 2010, pp.1-8.

[CH07] N.Chumerin, M.M.Van Hulle. An Approach to On-Road Vehicle Detection, Description and Tracking. IEEE Workshop on Machine Learning for Signal Processing, 27-29 Aug. 2007, pp.265-269.

[CPS06] K.Chellapilla, S.Puri, P.Simard. High Performance Convolutional Neural Networks for Document Processing. 10th International Workshop on Frontiers in Handwriting Recognition, 23-26 October 2006.

[DAL+07] L.Duvieubourg, S.Ambellouis, S.Lefebvre, F.Cabestaing. Obstacle Detection Using a Single Camera Stereo Sensor. Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, 16-18 Dec. 2007, pp.979-986.

[Dom07] M.Dominic K. Vector drawing: OpenGL shaders and cairo. [žiūrėta 2010.06.17]. Prieiga per internetą: <http://www.mdk.org.pl/2007/8/6/vector-drawing-opengl-shaders-and-cairo>

[Hay99] S.Haykin. Neural Networks – A Comprehensive Foundation. Second Edition. Prentice Hall, 1999.

[Khr10a] Khronos Group. OpenCL Overview (June 2010). [žiūrėta 2010.06.17]. Prieiga per internetą: <http://www.khronos.org/developers/library/overview/opencl_overview.pdf>

[Khr10b] Khronos Group. OpenCL 1.1 Specification (revision 33, June 11, 2010). [žiūrėta 2010.06.17]. Prieiga per internetą: <http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf>

[KK09] P.Kinderis, R.Krasauskas. Efektyvus NURBS paviršių vaizdavimas. Vilnius, 2009.

[LBB+98] Y.Lecun, L.Bottou, Y.Bengio, P.Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, vol.86, no.11, Nov. 1998, pp.2278-2324.

[LGT+97] S.Lawrence, C.L.Giles, A.C.Tsoi, A.D.Back. Face recognition: a convolutional neural-network approach. IEEE Transactions on Neural Networks, vol.8, no.1, Jan. 1997, pp.98-113.

[LH07] D.Luebke, G.Humphreys. How GPUs Work. Computer, vol.40, no.2, Feb. 2007, pp.96-100.

[LHB04] Y.LeCun, F.J.Huang, L.Bottou. Learning methods for generic object recognition with invariance to pose and lighting. Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol.2, no.2, 27 June-2 July 2004, pp.97-104.

[LMB+05] Y.LeCun, U.Muller, J.Ben, E.Cosatto and B.Flepp: Off-Road Obstacle Avoidance through End-to-End Learning. Advances in Neural Information Processing Systems, MIT Press, 2005.

[ND10] J.Nickolls, W.J.Dally. The GPU Computing Era. Micro, IEEE, vol.30, no.2, March-April 2010, pp.56-69.

[Nvi10] NVIDIA. CUDA Community Showcase – HTML Version.
[žiūrėta 2010.06.17]. Prieiga per internetą:
<http://www.nvidia.com/object/cuda_showcase_html.html>

[OHL+08] J.D.Owens, M.Houston, D.Luebke, S.Green, J.E.Stone, J.C.Phillips. GPU Computing. Proceedings of the IEEE, vol.96, no.5, May 2008, pp.879-899.

[SBM06] Z.Sun, G.Bebis, R.Miller. On-road vehicle detection: a review. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.28, no.5, May 2006, pp.694-711.

[SKP10] D.Strigl, K.Kofler, S.Podlipnig. Performance and Scalability of GPU-Based Convolutional Neural Networks. 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, 17-19 Feb. 2010, pp.317-324.

[SVS10] Spille, Vötter, Sauter. Geforce GTX 480 and GTX 470 reviewed: Fermi performance benchmarks. PC Games Hardware, Mar 27 2010.
[žiūrėta 2010.06.17]. Prieiga per internetą:
<http://www.pcgameshardware.com/aid,743498/Geforce-GTX-480-and-GTX-470-reviewed-Fermi-performance-benchmarks/Reviews/ >

SANTRUMPOS

ALU – angl. *Arithmetic Logic Unit*, tai skaitmeninės grandinės vienetas, kuris atlieka aritmetines ir logines operacijas

CP – centrinis procesorius

CUs – angl. *Compute Units*, skaičiavimo vienetai

ECC – angl. *Error Correcting Code*, atminties apsauga nuo iškraipymų

FBO – angl. *Frame Buffer Objects*, kadro buferio objektai

FLOPS – angl. *FLoating point Operations per Second*, slankaus kablelio operacijų skaičius per sekundę

GP – grafinis procesorius

KNT– konvoliucinis neuroninis tinklas

LIDAR – angl. *LIght Detection and Ranging*, aktyvusis šviesos jutiklis, skirtas aptikti objektams erdvėje

PEs – angl. *Processing Elements*, apdorojimo elementai

SIMD – angl. *Single-Instruction Multiple-Data*, vienos instrukcijos kelių duomenų vienetų programavimo modelis

SMs – angl. *Streaming Multiprocessors*, srauto multiprocesoriai

SPMD – angl. *Single-Program Multiple-Data*, vienos programos kelių duomenų vienetų programavimo modelis

VBO – angl. *Vertex Buffer Objects*, viršūnių buferio objektai