KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACINIŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

GINTAUTAS BIELIAUSKAS

Erdvinių objektų kolizijų aptikimo metodų tyrimas

Magistro darbas

Darbo vadovas:
doc. dr. Antanas Lenkevičius

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACINIŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

GINTAUTAS BIELIAUSKAS

Erdvinių objektų kolizijų aptikimo metodų tyrimas

Magistro darbas

Darbo vadovas:
doc. dr. Antanas Lenkevičius
2015-05-25

Recenzentas:
prof. dr. Kęstutis Motiejūnas
2015-05-25

Atliko:
IFM-3/1 gr. studentas
Gintautas Bieliauskas
2015-05-25

KAUNAS, 2015

AUTORIŲ GARANTINIS RAŠTAS
DĖL PATEIKIAMO KŪRINIO

2015 - ..... - ..... d.

Kaunas

Autoriai, Gintautas Bieliauskas

patvirtina, kad Kauno technologijos universitetui pateiktas baigiamasis magistro (toliau vadinama - Kūrinys) Erdvinių objektų kolizijų aptikimo metodų tyrimas.

pagal Lietuvos Respublikos autorių ir gretutinių teisių įstatymą yra originalus ir užtikrina, kad

1) jį sukūrė ir parašė Kūrinyje įvardyti autoriai;

2) Kūrinys nėra ir nebus įteiktas kitoms institucijoms (universitetams) (tiek lietuvių, tiek užsienio kalba);

3) Kūrinyje nėra teiginių, neatitinkančių tikrovės, ar medžiagos, kuri galėtų pažeisti kito fizinio ar juridinio asmens intelektinės nuosavybės teises, leidėjų bei finansuotojų reikalavimus ir sąlygas;

4) visi Kūrinyje naudojami šaltiniai yra cituojami (su nuoroda į pirminį šaltinį ir autorių);

5) neprieštarauja dėl Kūrinio platinimo visomis oficialiomis sklaidos priemonėmis.

6) atlygins Kauno technologijos universitetui ir tretiesiems asmenims žalą ir nuostolius, atsiradusius dėl pažeidimų, susijusių su aukščiau išvardintų Autorių garantijų nesilaikymu;

7) Autoriai už šiame rašte pateiktos informacijos teisingumą atsako Lietuvos Respublikos įstatymų nustatyta tvarka.

Autorius

Gintautas Bieliauskas _____

(parašas)

TURINYS

Paveiksliukų sąrašas 6

Santrauka....7

Summary....8

TERMINŲ BEI SANTRUMPŲ ŽODYNAS....9

ĮVADAS....10

Tyrimo tikslas ir uždaviniai 13

1 Erdvinių objektų kolizijų aptikimo metodų analizė 14

1.1 Analizės tikslas....14

1.2 Tyrimo objektas, sritis ir problema....14

1.3 Plačioji fazė....17

1.4 Siauroji fazė 19

1.5 Siauroji fazė (atstumas, nuolatinis kolizijos nustatymas, įsiskverbimo gylis)....22

1.5.1 Atstumas....23

1.5.2 Nuolatinis kolizijos nustatymas 23

1.5.3 Įsiskverbimo gylis....24

1.6 Kritinio laiko kolizijos nustatymas 25

1.6.1 Kolizijos nustatymas liečiamosiose aplinkose 26

1.7 Deformuojamų objektų kolizijos nustatymas ir GPU pagrįsti metodai 27

1.8 Kolizijos uždavinių klasifikacija 29

1.9 Algoritmų palyginimas....29

1.9.1 „V-COLLADE“ algoritmas 31

1.9.2 „PQP“ algoritmas....32

1.9.3 „SOLID“ algoritmas....33

1.9.4 „OPCODE“ algoritmas....34

1.9.5 „Dop-Tree“ algortimas....35

1.9.6 „Box-Tree“ algoritmas 35

1.10 Išvados....37

2 Projektinė dalis....38

2.1 Projekto planas....38

2.2 Techninės ir programinės įrangos reikalavimai....38

2.3 Algoritmų realizavimas 39
2.4 Eksperimentiniai išvesties duomenys 40
3 Tyrimo eksperimentinė dalis 42
3.1 Atliekamo tyrimo metodologija 42
3.2 Pasirinktų tyrimų paskirtis 42
3.3 Tyrimui naudojamos įrangos bei parametrų aprašas 42
3.4 Tyrimo rezultatai 43
3.5 Išvados 51
4 Išvados 52
5 Literatūros sąrašas 53

Paveiksliukų sąrašas

1.1 pav. tipinis kolizijos nustatymo specializuotų modulių sekos projektas 16
1.2 pav. Skirtingi ribojantys tūriai 18
1.3 pav. *BVH* principas: geometriniai objektai suskirstomi rekursyviai į jų geometrinių bazinių elementų poaibius (kairėje) ir kiekvienas mazgas ant medžio realizuoja ribojantį tūrį visiems baziniams elementams jo pomedyje (dešinėje) 19
1.4 pav. Vienalaikė rekursyvi dviejų *BVH* paieška kolizijos metu tikrina rezultatus ribojančio tūrio testo medyje 20
1.5 pav. *The Voxmap-Pointshell 6 DOF* haptinio vaizdavimo metodas naudoja dvi skirtingas duomenų struktūras: vokselizacija (angl. *voxelization*) (kairėje) ir objektų paviršiaus taškų mėginių ėmimą (dešinėje).....27
1.6 pav. Deformuojamieji objektai, kaip pavyzdžiui audinys, reikalauja specialių algoritmų, kadangi iš anksto apskaičiuotos duomenų struktūros tampa klaidingomis po deformacijos. Be to, gali susidaryti kolizija tarp objekto dalių. 27
1.7 pav. Skirtingos *BVH's* atnaujinimo strategijos 28
1.8 pav. Trimačių geometrinių modulių klasifikacija....29
1.9 pav. Pirmojoje nuotraukoje raudona spalva žymimos vietos kur susikerta du objektai. 32
1.10 pav. Antroje nuotraukoje raudona linija pažymėti du artimiausi taškai tarp triušio ir riestainio. 32
1.11 pav. Paskutinėje nuotraukoje yra parodyti trys žingsniai, kur susidūrimas tarp objektu neįvyksta. Kelias skaičiuojamas su kelio planuotoju (angl. *path-planner*), naudojant tolerancijos patikrinimą. 32
2.1 pav. Darbo eiga, nustatant kolizijas....41
3.1 pav. Naudojamo kompiuterio techninių duomenų paveiksliukas 42
3.2 pav. [14] Pilies ir grotelių paveiksliukai, kurie buvo naudojami tyrime. 44
3.3 pav. Algoritmų rezultatai, naudojant pilies paveiksliukus. 44
3.4 pav. Algortmų rezultatai, naudojant grotelių paveiksliukus. 45
3.5 pav. [14] „Apollo 13" ir „ATST walker" paveiksliukai, kurie buvo naudoti tyrime. 46
3.6 pav. Algoritmų rezultatai, naudojant „Apollo 13" kapsulės paveiksliukus. 46
3.7 pav. Algoritmų rezultatai, naudojant „ATST" ėjiko paveiksliukus. 47
3.8 pav. Kolizijos aptikimo laiko priklausomybė nuo paveikslėlio sudėtingumo su šviestuvo paveikslėliais. 47
3.9 pav. [14] Šviestuvo ir kėdės paveiksliukai, kurie buvo naudoti tyrime. 48
3.10 pav. Kolizijos aptikimo laiko priklausomybė nuo paveikslėlio sudėtingumo su grotelių paveikslėliais. 48
3.11 pav. Kolizijos aptikimo laiko priklausomybė nuo paveikslėlio sudėtingumo su pilies paveikslėliais. 49
3.12 pav. Kolizijos aptikimo laiko priklausomybė nuo paveikslėlio sudėtingumo su kėdės paveikslėliais. 50

Santrauka

Erdvinių objektų kolizijų aptikimo metodų tyrimas

Kompiuterių moksle, įskaitant fiziškai paremtas simuliacijas, kelio planavime ir lietimo atvaizdavime, kolizijos aptikimas yra fundamentali problema. Per paskutinius dešimtmečius, buvo pasiūlyta daugybė algoritmų kolizijos užklausų pagreitinimui. Tačiau, vis dar yra keletas neužbaigtų iššūkių.

Siekiant užtikrinti teisingą skirtingų susidūrimo algoritmų palyginimą, rekomenduoju keletą metodų. Tai, apima teorinio hierarchinės kolizijos aptikimo algoritmų laiko veikimo modelį ir atvirą lyginimo šaltinį, kuris vertina kolizijos atlikimą. Šiame darbe yra tiriamas šių algoritmų veikimas, kurie veikia naudojant CPU (*Central processing unit*) ir GPU (*Graphics processing unit*), analizuojami erdvinių objektų kolizijų aptikimo metodai. Darbe iškeltų tikslų pasiekimui yra realizuotos bazinės algoritmų versijos, kurios buvo integruotos į vieną paketą tam, kad tikslingai būtų galima palyginti visų algoritmų veikimo laiką. Šiame darbe atlikti tyrimai parodė, kad nėra vieno įrankio tinkančio visiems scenarijams. Tačiau bendriniu atveju galima būtų siūlyti „Dop-Tree“, kadangi jo atveju derinami glaudūs BV ir greiti BV testai. Taip pat tyrimo rezultatai parodė, kad kartais didinant objekto sudėtingumą, kolizijos aptikimo laikas greitėja, tačiau tai tik išskirtiniai variantai.

Raktiniai žodžiai

Kolizijos aptikimas, centrinis procesorius, grafinis procesorius, algoritmų palyginimas, BV.

Summary

Research of methods for collision detection of spatial objects

In computer science, including physically based simulations, road planning and in representing the issues in the collision detection is a fundamental problem. Over the past decades, many algorithms have been proposed to speed up collision detection queries. However, there are still a number of open challenges.

In order to ensure a fair, different collision algorithms comparison I recommend several methods. This includes theoretical hierarchical collision detection algorithms time operating model and open source compareing which the performance of the collision detection. In this work has studied the performance of algorithms that operate using the CPU (Central processing unit) and GPU (Graphics processing unit), analyzing spatial objects conflict detection methods. The work objectives achievement is realized in the base version of algorithms have been integrated into a single package in order to specifically allow for comparison all algorithms time. This paper studies have shown that there is no single tool suitable for all scenarios. However, the generic would be readily offer Dop-Tree, as in his case combines close BV and quick BV tests. Another study showed that while increasing the complexity of the subject, the collision detection time is improving, but it's just outstanding options.

Keywords

Collision detection, Central processing unit, Graphics processing unit, algorithms comparison, BV

TERMINŲ BEI SANTRUMPŲ ŽODYNAS

Cpu	(orig. *Central processing unit*) loginis įtaisas, apdirbantis duomenų srautą
Gpu	(orig. *Graphics processing unit*) loginis įtaisas, apdirbantis ir atvaizduojantis duomenų srautą.
„OpenSG“	Grafikos sistema skirta kurti realaus laiko grafines programas.
SMS	(org. *Pipeline*) specializuotų modelių seka, naudojama kolizijos aptikimui
GCC	(org. *GNU Compiler Collection*) laisvųjų kompiliatorių rinkinys, šiuo metu turintis C, C++, Java, Fortran ir Ada kompiliatorius
BVH	(org. *bounding volume hierarchy*) ribojančio tūrio hierarchijos, tai medžio tipo struktūra su geometrinių objektų rinkiniais
BV	(org. *bounding volume*) ribojantys tūriai, tai BVH medžio lapai ir mazgai
AABB	(org. axis-aligned bounding box) pagal ašį sulygiuotos ribojančios dėžės
OBB	(org. oriented bounding box) orientuotos ribojančios dėžės

ĮVADAS

Problemos aktualumas

0.1 pav. Teniso simuliacijos 1972 ir 2012 metais

Interaktyvių kompiuterių, imituojančių aplinkas laipsnis žymiai išaugo per pastaruosius dešimtmečius. Stulbinantys vaizdo ir garso pristatymo patobulinimai yra akivaizdūs. Realaus laiko sekimo sistemos, kurios tik prieš kelerius metus buvo slepiamos „VR“ laboratorijose, dabar randamos kiekvieno vaiko kambaryje. Šios naujos įvesties technologijos, pavyzdžiui, „Nintendo Wii“, „Sony Move“ ar „Microsoft Kinect“, atvėrė visai naują ir natūralesnį sąveikos 3D aplinkoje būdą plačiajai auditorijai.

Tačiau, svaiginanti patirtis interaktyvioje, virtualioje aplinkoje reikalauja ne tik realistinių garsų, grafikos ir sąveikos metaforų, bet ir tikėtino objektų, su kuriais sąveikaujame, elgesio. Pavyzdžiui, jei realiame pasaulyje objektai sąveikauja, tai yra, susiduria, jie gali atšokti vienas nuo kito arba sutrupėti į gabalėlius (jei jie standūs). Objektams esant nestandiems, jie deformuojasi. Akivaizdu, kad ir kompiuterinėje aplinkoje mes tikimės panašaus elgesio.

Iš tiesų, suvokimo psichofiziniai eksperimentai parodė, kad dėl neįprasto fizinio elgesio mes išsiblaškome, daugiausiai dėl vizualiųjų ženklų. Pavyzdžiui, „Reitsma“ ir „O’Sullivan“ [1] parodė, kad susidūrimo uždelsimas ženkliai sumažina atsako suvokimą. Laimei, tolimesni eksperimentai rodo, kad mes, sąveikaudami su pasauliu, taip tiksliai neatkartojame Niutono dėsnių, tačiau kolizijos sprendimus atliekame remiantis objektų kinematiniais duomenimis. Vadinasi, pakanka pateikti fiziškai tikėtiną elgesį vietoj fiziškai teisingo elgesio.

Tačiau, kompiuterio generuojamame pasaulyje objektai dažniausiai atstovauja abstraktų geometrinį modelį. Pavyzdžiui, buvo suderinti paviršiai poligonais kuriuos apibūdinam matematinėmis funkcijomis, tokiomis kaip „NURBS“. Tokie abstraktūs pareiškimai savaime neturi fizinių savybių. Iš

tiesų, jie tiesiog plūduriuoja vienas per kitą. Taigi, reikia pridėti tinkamą algoritminį kontaktų tvarkymui.

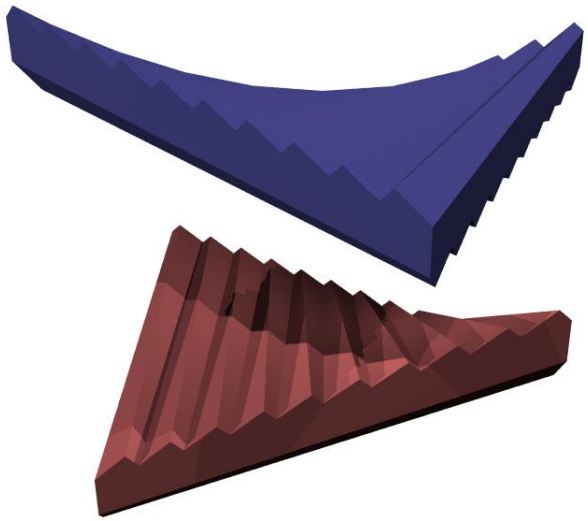
Smulkiau, pirmiausia turime rasti ryšius tarp judančių objektų. Šis procesas vadinamas susidūrimo aptikimu. Antrajame etape, reikia išspręsti šiuos susidūrimus fiziškai logišku būdu. Tai vadinama susidūrimo atsakymu.

Ši fundamentali technika nėra apibrėžta realaus laiko modeliavimo, kuris paprastai naudojamas kompiuterinėje grafikoje, kompiuteriniuose žaidimuose, virtualioje realybėje ar virtualaus surinkimo užduotyse. Faktiškai, reikia visų šių užduočių, apimančių objektų judesio imitavimą, kurios neleidžia objektams prasiskverbti vienas pro kitą. Tai apima realaus laiko animaciją, animacijos „CGI“ (*Computer-Generated Imagery*) filmus, robotų paraiškas, kur aptikus susidūrimą, padeda išvengti kliūčių ir savekolizijos tarp roboto dalių. Be to, tai yra naudojama kelių išplanavimui, molekulinio sujungimo užduotims ir daugiaašiam NC (*Numerical control*) apdirbimui.

Šis platus kolizijos aptikimo aplikacijų spektras įrodo, kad šiai temai jau buvo skirta nemažai laiko. Tiesą sakant, šimtai, o gal net tūkstančiai, skirtingų tyrimų buvo atlikta apie susidūrimo aptikimo problemų sprendimus. Pavyzdžiui, „Google-Scholar“ apie kolizijos aptikimą turi daugiau nei 44000 rezultatų.

Tai reikalauja iškelti keletą klausimų:

- Kodėl kolizijos aptikimas toks sunkus, kad jam jau yra skirta tiek daug laiko?
- Ar dar yra kur tobulėti? Ar viskas šia tema jau išsakyta?



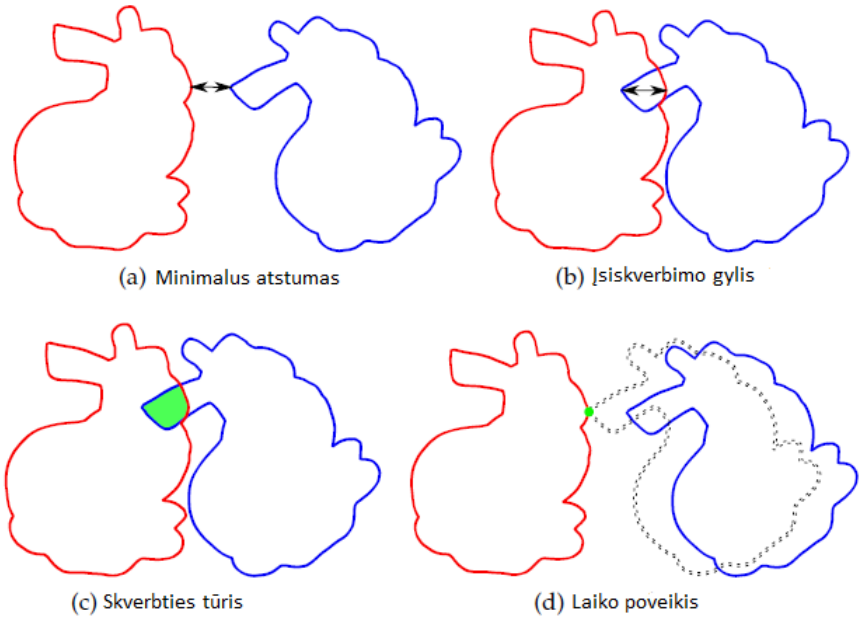
0.2 pav. Dviejų „Chazelle“ daugiakampių susidurimas yra blogiausias atvejis kolizijos aptikimui

Tiesą sakant, rasti susidūrimus tarp geometrinių objektų yra labai sunki užduotis. Daugumoje programų, susidūrimo paieška, dėl sau būdingo sudėtingumo, tampa kliūtimi. Tiesiog, pagalvokite apie dviejų objektų esančių ant daugiakampio paviršiaus atstovavimą. Kiekvienas iš jų yra sumodeliuotas iš „n“ poligonų. Brutalios jėgos požiūriu, susidūrimo aptikimo algoritmas gali būti paprastai išbandytas su dvejais daugiakampiais objektais. Tai lemia O (n^2) sudėtingumas. Jei objektai yra netinkamos formos, norint sudurti daugiakampius, galima sukurti konfigūracijas su O (n^2) (pav. 0.2 [11]). Šie atvejai yra pakankamai dirbtini ir negali dažnai atsitikti praktiškai atitinkamose situacijose. Daugeliui realaus pasaulio scenarijų, galėtume įrodyti O (n log n) sudėtingumą. Tačiau, susidūrimo aptikimo algoritmai turi veikti tinkamai net ir blogiausiais scenarijais. Taigi, teorinis daugelio susidūrimų aptikimų algoritmų sudėtingumas blogiausiu atveju yra O (n^2).

Dauguma susidūrimo aptikimo algoritmų remiasi protingomis duomenų struktūromis, užtikrinančiomis susidūrimo aptikimo užklausų jautrų išvesties pagreitį. Sekančiam skyriuje pateiksiu šio lauko klasikinių ir neseniai atliktų tyrimų apžvalgą.

Paprastai, šios duomenų struktūros apdorojimas užtrunka ganėtinai ilgai. Deja, jei objektai nėra standūs, t.y. jie laikui bėgant deformuojasi, šios iš anksto pateiktos duomenų struktūros tampa neteisingomis ir turi būti iš naujo apskaičiuojamos arba atnaujinamos. Beveik visi ankstesni susidūrimo aptikimo metodai buvo paremti išankstinio rėmo pagrindu. Tai reiškia, jog jie atnaujindavo savo duomenų struktūras prieš kiekvieną susidūrimo užklausimą. Savaime suprantama, kad tai užėmėlabai daug laiko ir buvo viena iš prastos kokybės ir deformuotų objektų priežasčių.

Visgi, surasti susidūrimus tėra tik vienas būdas. Kaip jau buvo minėta anksčiau, susidūrimai turi būti išspręsti per susidūrimų tvarkymo procesą. Kad būtų galima apskaičiuoti fiziškai patikimus susidūrimus, reikalingi duomenys turi būti pateikti pagal susidūrimo aptikimo algoritmus. Iš esmės, egzistuoja keturi skirtingų rūšių kontaktinės informacijos tipai, kurie gali būti naudojami įvairių susidūrimų sprendime: galima sekti *minimalius atstumus* tarp objektų, nustatyti tikslų *poveikio laiką*, apibrėžti minimalius taikomuosius vektorius, skirtus apibrėžti objektus, vadinamąjį *įsiskverbimo gylį*, arba galima apskaičiuoti *skverbties tūrį* (0.3 pav). Kitame skyriuje detaliau aptarsiu skirtingų skverbties priemonių privalumus ir trūkumus.



pav. Kontaktinės informacijos tipai

Pagal Fisher ir Lin [2] skverbties apimtis yra pats sudėtingiausias, tačiau veiksmingiausias metodas, skirtas apibrėžti sankirtos apimtis. Tačiau, mano žiniomis, dar nėra algoritmo galinčio tai apskaičiuoti realiu laiku, t.y. daugiau negu tuziną poligonų.

Taip pat, egzistuoja ir kiti susidūrimo aptikimo metodai. Tačiau programuotojas, norėdamas integruoti susidūrimo aptikimą, turi pasirinkti iš šimtų skirtingų prieigų. Akivaizdu, jog tai yra beveik neįmanoma nestudijuojant visos literatūros. Bet net ir ekspertams yra sunku spręsti apie susidūrimo aptikimą vien tik iš jų aprašų, nes kiekvienas autorius rašo apie specifinius scenarijus. Norint tai ištaisyti, buvo sukurtas standartizuotas objektų žymėjimas susidūrimo aptikimui, kurį pristatysiu šiame darbe. Jis leidžia atpažinti teisingą ir realų palyginimą, skirtingiems algoritmams ir scenarijams.

Tyrimo tikslas ir uždaviniai

Šios darbo tikslas yra išanalizuoti realizuotų tradicinių susidūrimų paieškos algoritmų našumo rezultatus. Sprendžiami uždaviniai:

- Išanalizuoti kolizijos aptikimo problemą.
- Išanalizuoti įvairius kolizijų metodus.
- Palyginti juos tarpusavyje, nustatant jų pranašumus ir trūkumus.

1 Erdvinių objektų kolizijų aptikimo metodų analizė

1.1 Analizės tikslas

Išsiaiškinti pagrindines kolizijos aptikimo metodų savybes, skirtumus, panašumus ir trūkumus.

Tyrimas susideda iš trijų pagrindinių dalių ir jų smulkesnių uždavinių:

1. Erdvinių objektų kolizijų aptikimo metodų analizė:

- Išnagrinėti esamus erdvinių objektų kolizijų aptikimo metodus („V-COLLIDE“, „PQP, SOLID“ ir kt.).
- Išnagrinėti esamus erdvinių objektų kolizijų aptikimo formavimo būdus.

2. Erdvinių objektų kolizijų aptikimo metodų sudarymas:

- Sudaryti erdvinių objektų kolizijų aptikimo metodo algoritmą.

3. Erdvinių objektų kolizijų aptikimo metodų prototipas:

- Sukurti algoritmą, realizuojantį prototipą.
- Ištestuoti ar prototipo veikimas atitinka iškeltus reikalavimus.

1.2 Tyrimo objektas, sritis ir problema

Tyrimo objektas:

Erdvinių objektų kolizijų aptikimo metodai.

Tyrimo sritis:

- Realaus laiko kompiuterinėje grafikoje.
- Kompiuterinėje animacijoje.
- Virtualios realybės modeliavimuose.
- Kompiuteriniame projektavime ir gamyboje („CAD“/„CAM“).
- Skaičiuojamojoje geometrijoje.
- Roboto technikoje, kibernetikoje.
- Kitose haptinės sąveikos sistemose (medicinoje, fizikoje).

Problema:

Problemos esmė ta, kad du nepersidengiantys modeliai negali užimti tos pačios erdvės dalies. Kolizijų aptikimo mechanizmas leidžia imitaciniu būdu spręsti daugelį modeliavimo uždavinių, tokių kaip maršruto planavimą, inžinerinę analizę, montažą, tolerancijos tikrinimą.

Šiame skyriuje pateiksiu kolizijos nustatymo tyrimo apžvalgą. Įvade jau paminėjau bendrą

kolizijos nustatymo sudėtingumo problemą, atsirandančią dėl jos teorinio kvadratinio daugiakampių modelio (tokių, kaip „Chazelle“ briaunainis – žr. Pav. 0.2) veikimo laiko.

Tačiau, tai yra dirbtinis pavyzdys ir daugeliu realaus pasaulio atvejų yra tik labai nedaug susiduriančių daugiakampių. Taigi, kolizijos nustatymo algoritmų tikslas yra suteikti išvesčiai jautrų veikimo laiką. Tai reiškia, kad jie bando eliminuoti kiek įmanoma daugiau O (n^2) primityvių testų, pavyzdžiui pasitelkiant ankstyvą objektų, kurie negali susidurti, didelių dalių pašalinimą. Todėl kolizijos nustatymo problema gali būti laikoma filtravimo procesu.

Pastarosios fizikos simuliacijos bibliotekos kaip antai „PhysX“ [3] , „Bullet“ [4] ir „ODE“ [5], kurios įgyvendina keletą filtravimo lygių, yra vadinamos kolizijos nustatymo specializuotų modelių seka.

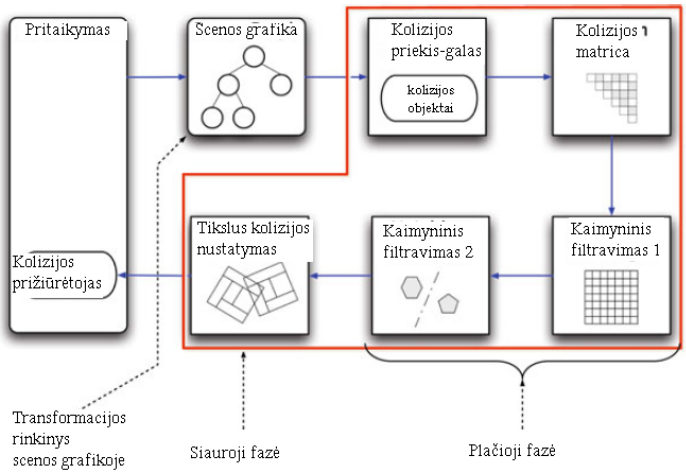
Dažnai scena susidaro ne iš vienintelės objektų poros, bet iš didesnio organizuoto 3D modelių rinkinio. Pirmajame filtravimo žingsnyje, *plačiajame etape* (angl. *broadphase*) ar *N-kūnų atrinkime* (angl. *N-Body culling*), greitas testas išvardija visas, potencialiai susiduriančių objektų poras (taip vadinamus potencialios kolizijos rinkinius („PCS“)) tam, kad patikrinti tikslų susikirtimą antrajame žingsnyje, kuris vadinamas *siauraja faze* (angl. *narrowphase*). Siauroji fazė paprastai skirstoma į dvi dalis: pirmojoje yra taikomas filtras, skirtas išgauti potencialiai susiduriančių geometrinių bazinių elementų poras, o antrojoje dalyje šios poros yra patikrinamos dėl kolizijos. Norint pagreitinti kolizijos nustatymo procesą, priklausomai nuo scenos tarp šių pagrindinių žingsnių gali būti panaudota daugiau filtravimo lygių. Paveikslėlis 1.1 [9] vaizduoja kolizijos aptikimo projekto algoritmą. Visi šiame darbe naudoti algoritmai buvo integruoti į šią struktūrą.

Tačiau, chronologinė kolizijos nustatymo modulių seka yra vienintelis būdas klasifikuoti kolizijos nustatymo algoritmus ir čia egzistuoja daug daugiau savitų veiksnių. Kitos klasifikacijos yra, pvz.: standieji kūnai prieš deformuojamos objektus. Dažniausiai filtravimo žingsniai būna pagrįsti geometrinių pagreitinimo duomenų struktūromis, kurios yra sudarytos prieš apdorojimo žingsnį. Jei objektai yra deformuojami, šios iš anksto apskaičiuotos duomenų struktūros gali tapti klaidingomis. Todėl deformuojami objektai reikalauja kitų duomenų struktūrų ar mažiausiai papildomų žingsnių tam, kad atnaujinti ar perskaičiuoti iš anksto apdorotas struktūras.

Be to, deformuojami objektai reikalauja savųjų kolizijų patikrinimo. Kai kurie iš šių metodų bus aprašyti toliau.

Kitas skiriamasis bruožas yra geometrinių objektų atvaizdavimas. Ypatingai kompiuterinėje grafikoje objektų ribos yra paprastai priartinamos daugiakampiais. Taigi, dauguma kolizijos nustatymo algoritmų yra sukurti daugiakampiams objektams. Tačiau, „CAD“/„CAM“ (angl. *Computer Aided*

Designing/Computer Aided Manufacturing) aplikacijos taip pat ir lenkto paviršiaus atvaizdavimai, kaip antai svarbios nevienarūšės racionalaus pagrindo kreivės (angl. *non-uniform rational B-splines (NURBS)).* Kita „CAD“/„CAM“ dažnai naudojama objektų modeliavimo technika yra *konstruktyvi kieta geometrija* (angl. *constructive solid geometry (CSG)).* Objektai yra rekursyviai apibrėžti pagrindinių formų, kaip sferų ir cilindrų sąjungos, susikirtimo ar skirtumo operacijų.



1.1 pav. tipinis kolizijos nustatymo specializuotų modulių sekos projektas

Taškų debesys tampa vis labiau ir labiau populiarus, dėl pigių gylio kamerų, kurios gali būti panaudotos 3D skanavimui, kaip, pavyzdžiui „Microsoft Kinect“. Vienas iš pirmųjų būdų aptikti koliziją tarp taškų debesų buvo išvystytas Klein ir Zachmann [16]. Jie naudojo ribojančio tūrio hierarchiją kartu su sfera, apimančia paviršiaus dalis. Klein ir Zachmann [16] pasiūlė dviejų numanomų funkcijų interpoliacijos tyrimo metodą artumo diagramoje kartu su atsitiktinėmis imtimis.

Kiti algoritmai palaiko kolizijas tik tarp vienintelio taško zondo ir taškų debesies. Dėl to jie užpildo „AABB‘s“ tarpus, supančius taškus ir naudoja aštuonmedį tolimesniam greitinimui. Arba naudojo „R-medžius“ (angl. *R-trees*), hierarchinę duomenų struktūrą, kuri keliose dimensijose saugo geometrinius objektus su intervalais, kartu su plačiosios fazės tinkleliu. Stochastiniame metode naudojamas ribojančio tūrio hierarchijos skersinis. Naudojant mašinines mokymosi technikas, jų būdas sugeba tvarkyti triukšmingus taškų debesis. Papildomai, paprastiems kolizijos testams, jie palaiko minimalių atstumo skaičiavimą.

Tai, tiesiogiai veda prie kitos klasifikacijos ypatybės: informacijos rūšies, kuri yra suteikiama kolizijos nustatymo algoritmo. Faktiškai, beveik visi simuliacijos metodai veikia diskretiškai: tai reiškia, kad jie tikrina tik atskirais taškais laike ar susiduria simuliuoti objektai. Ko pasekoje, vidinis-įsiskverbimas tarp simuliuotų objektų yra dažnai neišvengiamas. Tačiau, tam, kad simuliuoti fiziškai

patikimą pasaulį, objektai neturėtų pereiti vienas per kitą ir turėtų judėti kaip tikimasi, kai yra stumiami ar traukiami. Kaip rezultatas, egzistuoja tam tikras skaičius kolizijos atsako algoritmų, skirtų išspręsti vidinį įsiskverbimą. Pavyzdžiui, bauda pagrįstas metodas apskaičiuoja nesiskverbiančias suvaržymo jėgas, pagrįstas skvarbos kiekiu. Kiti metodai, kaip impulsu pagrįstas metodas ar suvaržymu pagrįsti algoritmai, reikalauja informacijos apie tikslų kontakto laiką, skirtą pritaikyti impulsyvias jėgas.

Paprasti kolizijos nustatymo algoritmai dažniausiai praneša ar du objektai susikerta ar ne. Papildomai, kai kurie iš šių būdų suteikia prieigą prie vienintelės susikertančių daugiakampių poros ar jie duoda visų susikertančių daugiakampių rinkinį. Deja, tai nėra pakankama informacija, reikalinga daugumai kolizijos atsako schemų. Taigi, ten taip pat egzistuoja metodai, kurie sugeba apskaičiuoti, kai kurios rūšies *įsiskverbimo gylį*, pvz.: minimalų pereinamąjį vektorių, skirtą atskirti objektams. Labiau pažengę metodai suteikia *įsiskverbimo tūrį*. Ypatingai kelio planavimo užduotyse, bet taip pat ir suvaržymu pagrįstose (angl. *constraint-based*) simuliacijose yra naudinga sekti *minimalų atskyrimo atstumą* tarp objektų, kad išvengti kolizijų. Pagaliau, *nuolatinis kolizijų nustatymas*, apskaičiuoja tikslų tašką laiku, kai kolizija įvyksta tarp dviejų objektų konfigūracijų. Paprastai, kuo daugiau informacijos kolizijos nustatymo algoritmas suteikia, tuo ilgesnis yra užklausos laikas.

Gali būti daugiau kolizijos nustatymo algoritmų klasifikacijų. Pavyzdžiui, realaus laiko prieš neprisijungusį, hierarchinis prieš nehierarchinį, išgaubtas prieš neišgaubtą, GPU pagrįsti metodai prieš CPU, ir t.t. Tai rodo skirtingų metodų įvairovę. Tiesą sakant, kolizijos nustatymas buvo tirtas beveik tris dešimtmečius. Pilna, visų egzistuojančių metodų, apžiūra užpildytų bibliotekas ir yra už šio darbo ribų.

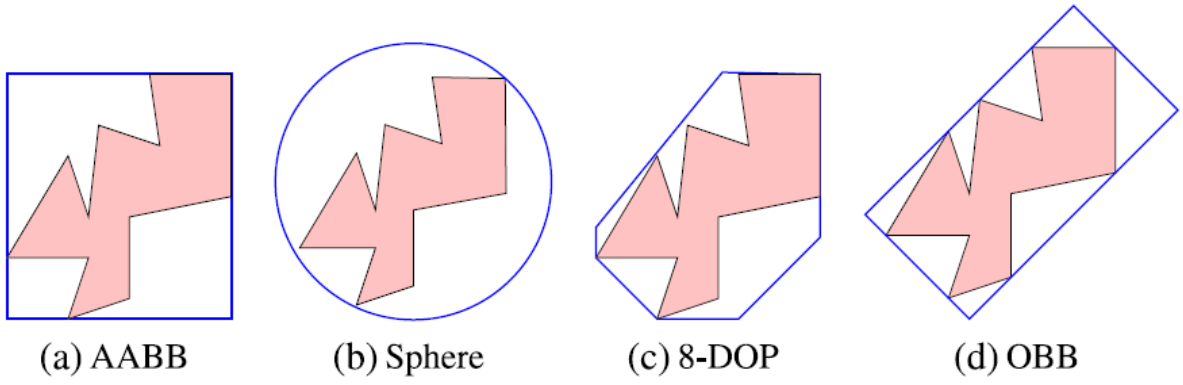
1.3 Plačioji fazė

Pirmoji specializuotų modulių sekos dalis, vadinama plačiąja faze, turėtų atlikti efektyvų dalių, kurios nėra kolizijoje, pašalinimą. Todėl, objektai dažniausiai yra „įdedami“ į pagrindines formas, kurios gali būti greitai ištestuotos ir nepersidengia. Tipinės pagrindinės formos yra *ašies sulygiuotos ribojančios dėžės (AABB),* sferos, *diskretieji orientuoti politopai (k-DOP)* ar *orientuotos ribojančios dėžės (OBB)* (žr. Pav. 1.2 [8]).

Pats paprasčiausias kaimyno radimo fazės metodas yra brutalios jėgos būdas, kuris palygina kiekvieno objekto ribojantį tūrį su kitais ribojančiais tūriais. Šio metodo sudėtingumas yra O (n^2), kur *n žymi objektų skaičių scenoje.* Pasiūlyti du pagrindiniai metodai: erdvinis dalinimas ir topologiniai metodai.

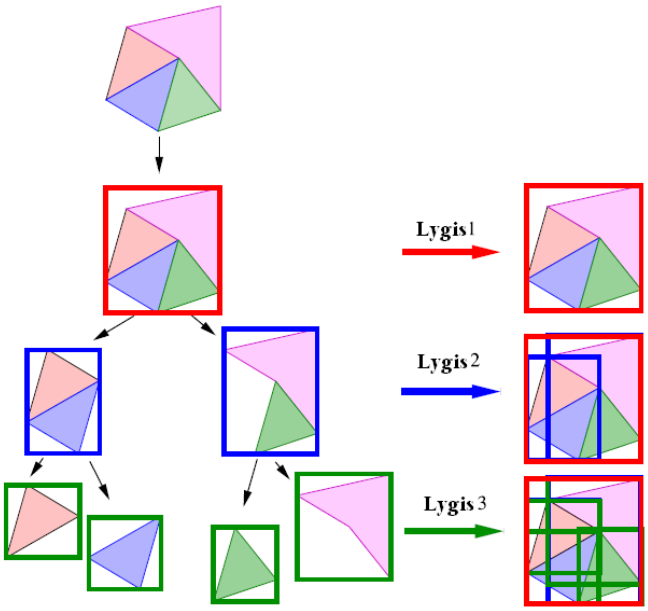
Erdvinio dalinimo algoritmai suskirsto erdvę į ląsteles. Objektai, kurių ribojantys tūriai dalinasi ta pačia ląstele, yra atrenkami siaurojoje fazėje. Tokio erdvinio dalijimo duomenų struktūrų pavyzdžiai yra reguliarūs tinkleliai, hierarchinės erdvinės dėstymo lentelės, aštuonmedžiai, kd-medžiai ir binarinės erdvės pertvaros (BSP-medžiai). Pagrindinis kolizijos nustatymo erdvinių padalinimo schemų trūkumas - jų statinis būdas: jie turi būti atstatyti arba atnaujinti kaskart, kai objektai keičia savo konfigūraciją. Vienodiems tinkleliams tokie atnaujinimai gali būti atlikti nuolatiniu laiku ir tinkleliai tobulai tinka paralelizacijai. Tačiau, vienodų tinklelių efektyvumas išnyksta, jei objektai labai skiriasi dydžiais.

Priešingai, nei erdvės dalinimo būdams, topologiniai metodai yra pagrįsti objekto pozicija kitų objektų atžvilgiu. Pats žinomiausias metodas yra vadinamas *nuskaitymo ir apkarpymo* algoritmu (angl. *Sweep-and-Prune)*. Pagrindinė idėja yra projektuoti objektų ribojantį tūrį ant vieno ar daugiau ašių (pvz.: trijų koordinačių ašių (x, y, z)). Siaurojoje fazėje turi būti aptartos tik tos objektų poros, kurių projektuoti ribojantys kūnai persidengia ant visų ašių. Dažniausiai, šis metodas nekonstruoja jokios vidinės struktūros, bet prasideda kiekvienos kolizijos tikrinime.



1.2 pav. Skirtingi ribojantys tūriai

1.4 Siauroji fazė



1.3 pav. *BVH* principas: geometriniai objektai suskirstomi rekursyviai į jų geometrinių bazinių elementų poaibius (kairėje) ir kiekvienas mazgas ant medžio realizuoja ribojantį tūrį visiems baziniams elementams jo pomedyje (dešinėje)

Kai plačioji fazė išvardina galimų susiduriančių objektų poras, siaurosios fazės tikslas yra nustatyti tikslius kolizijos patikrinimus tarp šių porų.

Siaurosios fazės brutalios jėgos sprendimas galėtų paprasčiausiai patikrinti visus vieno objekto geometrinius bazinius elementus su visais kito objekto baziniais elementais. Žinoma, tai dar kartą sukeltų kvadratinį sudėtingumą. Dėl greitų šiuolaikinės grafikos technikos įrangos pokyčių, šiandien objektai gali būti sudaryti iš milijonų daugiakampių ir kvadratinis veikimo laikas nėra tinkamas pasirinkimas. Taigi, reikia pažangesnių algoritmų.

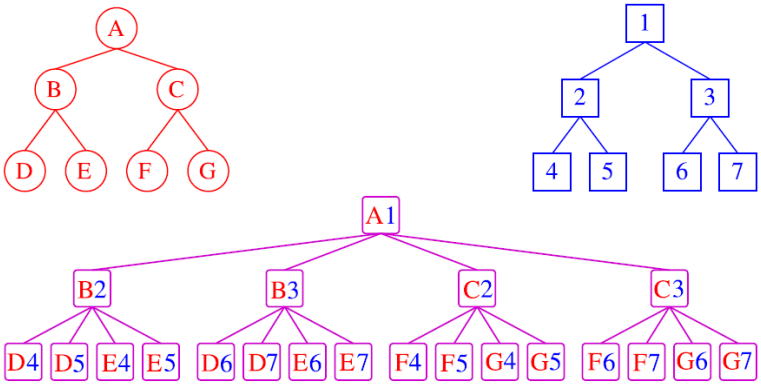
Iš tikrųjų, siauroji fazė gali būti suskirstyta į dvi fazes. Pirmojoje fazėje yra atrenkamos nepersidengiančios objektų dalys; antrajame žingsnyje atliekamas tikslus kolizijos ir pirmojoje fazėje neatrinktų geometrinių bazinių elementų porų nustatymas.

Vietoj duomenų struktūrų, kurios padalina pasaulio erdvę plačiojoje fazėje, siaurojoje fazėje dažniausiai naudojamos objektus dalinančios technikos, skirtos atrinkimo etapui.

Bendrosios duomenų struktūros šiai užduočiai yra *ribojančio tūrio hierarchijos (BVH's).* Visam objektui pritaikoma ribojančių tūrių technika, žinoma iš ankstesnio skyriaus (Paveikslėlis 1.2 [8]). Rezultatas – panaši į medį struktūra. Kiekvienas mazgas tokiame medyje yra susijęs su ribojančiu

tūriu, kuris apima visus bazinius elementus pomedyje (žr. pav. 1.3 [8]).

BVH dažniausiai konstruojama žingsnyje prieš apdorojimą, kuris gali būti šiek tiek pigesnis. Vykdymo metu vienalaikė rekursyvi dviejų objektų *BVH* medžio paieška (angl. *traversal*) leidžia konservatyvų nesusikirtimo apkarpymą: jei susikirtimas yra nustatomas *BVH* šaknyje, paieška veikia tikrinant ribojančius šaknies mazgo vaikų tūrius ir taip toliau, kol pasiekiami lapų mazgai ir gali būti atliktas tikslus kolizijos testas tarp geometrinių bazinių elementų. Nepersidengiantys BV yra išmetami iš tolimesnio svarstymo. Visas paieškos algoritmas duoda ribojančio tūrio testo medį (*BVTT*) (žr. Pav. 1.4 [8]).



1.4 pav. Vienalaikė rekursyvi dviejų *BVH* paieška kolizijos metu tikrina rezultatus ribojančio tūrio testo medyje

Optimalus ribojantis tūris turėtų:

- tiksliai atitikti pagrindinę geometriją
- suteikti greitus susikirtimo testus
- nekisti per kietą judesį (nepakeisti savo formos)
- nenaudoti per daug atminties
- būti galimas pastatyti automatiškai ir greitai

Deja, šie veiksniai yra prieštaringi. Pavyzdžiui, sferos greitai persidengia ir atstumo testai gali būti labai efektyvūs atminties atžvilgiu, bet prastai atitinka plokščias geometrijas. „AABB‘s“ taip pat siūlo greitus susikirtimo testus, bet jie reikalauja būti perlygiuoti po sukimų. Arba, jei nenaudojamas perlygiavimas, reikia daug brangesnio „OBB“ persidengimo testo. Šiuo atveju, sklandžiau atitinkantis „OBB“ galėtų būti panaudotas tiesiogiai. Tačiau jie taip pat reikalauja daugiau atminties. Iškilieji korpusai siūlo geriausią atitikimą tarp išgaubto BV, tačiau persidengimo testas yra labai sudėtingas ir atminties vartojimas priklauso nuo pagrindinės geometrijos.

Todėl, teisingo *BVH* pasirinkimas reikalauja kompromiso ir priklauso nuo scenarijaus. Iš esmės, *BVH* pagrįstų algoritmų kokybė gali būti matuojama kaštų funkcija, kuri buvo pristatyta „Weghorst", išanalizuoti hierarchinius metodus spindulio sekimui ir vėliau „Gottschalk" pritaikė ją hierarchiniams kolizijos nustatymo metodams:

$T = N_v C_v + N_p C_p$,kur

T = Iš viso išlaidų testuojant porą modelių susikirtimui

Nv = BV Testų skaičius

Cv = BV Testų išlaidos

N_p = bazinių elementų testų skaičius

Cp = bazinių elementų testų išlaidos

Papildomai prie BV formos, yra daugiau veiksnių, kurie veikia *BVH* efektyvumą, apimant hierarchijos aukštį, kuris neturėtų daryti įtakos masyvingumui ar paieškos eilei per kolizijos užklausas.

Iš esmės, egzistuoja dvi pagrindinės *BVH* statymo strategijos: iš apačios į viršų ir iš viršaus į apačią. Metodas iš apačios į viršų prasideda elementariu lapų mazgų BV ir sujungia juos rekursyviai kartu iki pasiekiamo šaknų BV. Labai paprastas euristinis sujungimas yra aplankyti visus artimiausius kaimynus ir minimalizuoti tame pačiame lygyje sujungtų tėvinių mazgų dydį. Mažiau gobšios strategijos suderina BV naudojant čerpes (angl. *tilings*).

Tačiau, pats populiariausias metodas yra nuo viršaus į apačią. Pagrindinė idėja yra pradėti su pilnu elementaraus BV rinkiniu, tada padalinti jį į kelias dalis ir kiekvienai daliai rekursyviai sukurti *BVH*. Didžiausia problema - parinkti gerą skėlimo kriterijų. Klasikinis skaldymo kriterijus yra paprasčiausiai parinkti ilgiausią ašį ir padalinti ją per vidurį. Kita paprasta euristika yra padalinti palei elementarių ribojančių dėžių medianą pagal ilgiausią ašį. Tačiau, yra lengva statyti blogiausio atvejo scenarijus šiems paprastiems metodams. *Paviršiaus ploto euristika* (angl. *surface area heuristic (SAH))* stengiasi išvengti šių blogiausių atvejų, optimizuojant paviršiaus plotą ir geometrinių bazinių elementų skaičių per visus galimus skėlimo plokštumos kandidatus. Originaliai išvystytas spindulio ieškojimui, jis šiandien yra naudojamas kolizijos nustatymui. Skaičiavimo sąnaudos gali būti sumažintos iki $O(n \log n)$ ir lygiagrečiai egzistuoja algoritmai, skirti greitai konstrukcijai GPU.

Kita svarbi geometrinių bazinių elementų klasė yra išgaubti politopai. Ne todėl, kad jie yra plačiai naudojami fizika pagrįstose simuliacijose, bet istoriniu požiūriu: kai kurie pirmųjų kolizijos nustatymo algoritmų yra pagrįsti būtent jais. Taip pat, jie gali būti naudojami tiek kaip geometriniai baziniai kintamieji, tiek kaip ribojantys tūriai. Iš tikrųjų, egzistuoja du pagrindiniai išgaubtų politopų metodai: ypatybe pagrįsti algoritmai ir simpleksu pagrįsti algoritmai.

Pirmasis, ypatybe pagrįstas, metodas buvo pasiūlytas Lin ir Canny [7]. Išgaubto daugiasienio bruožai yra viršūnės, kraštai ir sienos. „Lin-Canny“ algoritmas yra lokali, šiomis savybėmis pagrįsta paieška, naudojanti iš anksto apskaičiuotą „Voronoi“ diagramą. Išgaubtumas garantuoja, kad lokalūs minimumai yra vengiami. Be to, algoritmas naudoja erdvinį ir laiko suderinamumą tarp dviejų skiriamųjų užklausų: dažniausiai objektai nejuda per daug tarp dviejų fizika pagrįstų simuliacijų kadrų.

Taigi, arčiausia ypatybė dabartiniame kadre yra arčiau už arčiausią savybę iš kito kadro. Pagrindinis algoritmo trūkumas yra tas, kad jis negali valdyti susidūrimų. Šiuo atveju, jis veikia begalę ciklų. „V-Clip“, klasikinio „Linn-Canny“ metodo išplėtimas, kuris eliminuoja šį svarbų trūkumą.

Geriausiai žinomas „simplex“ pagrįstas algoritmas vietoj „Voronoi“ diagramų, „GJK-algoritmas“ yra pagrįstas „Minkowski“ skirtumais. Papildomai prie „Būlio“ kolizijų nustatymo, kuris paprastai praneša ar du objektai susiduria ar ne, „GJK-algoritmas“ taip pat grąžina skvarbos įvertinimą. Be to, jis pasiekia tą patį, beveik pastovų, laiko kompleksiškumą kaip ir „Lin-Canny“.

Abi algoritmų rūšys yra sukurtos išgaubtiems briaunainiams. Tačiau, naudojant išgaubtą gerai besielgiančių įgaubtų daugiasienių dekompoziciją, jie gali būti taip pat praplėsti į jų objektus. Tačiau, rasti geras išgaubtas dekompozicijas nėra paprastas metodas ir yra vis dar aktyvi tyrimo sritis.

1.5 Siauroji fazė (atstumas, nuolatinis kolizijos nustatymas, įsiskverbimo gylis)

Fizika pagrįstoms simuliacijoms paprastas „Būlio“ atsakymas į klausimą ar atskiruose taškuose laikę objektų pora susiduria ar ne, yra dažnai nepakankamas. Dažniausiai, norint apskaičiuoti atmušimo jėgas arba nesusikertančius suvaržymus, reikalinga tam tikro tipo kontaktinė informacija.

Tol, kol objektų pora ilsisi laisvoje nuo kolizijos konfigūracijoje, paprastas būdas charakterizuoti atmušimo jėgas - naudoti *minimalų atstumą* tarp jų. Tačiau, kolizijos dažnai yra neišvengiamos dėl simuliacijos proceso atskirose struktūrose. Todėl konfigūracijoms, kur objektai persidengia, svarbus skvarbos išmatavimas. Kai kurie autoriai pasiūlė mažiausią slenkamąjį vektorių, tam kad atskirti objektus. Tai, dažnai vadinama *įsiskverbimo gyliu*. Sudėtingiausias, bet ir vienintelis fiziškai patikimas skvarbos išmatavimas yra *įsiskverbimo tūris*, kuris tiesiogiai atitinka vandens kiekį, pakeičiantį persidengiančias objektų dalis. Ir galiausiai, galima apskaičiuoti tikslų tašką laike tarp dviejų atskirų kolizijos patikrinimų, tai vadinama *nuolatiniu kolizijos nustatymu*. Iš tiesų, tai nėra skvarbos masto matmuo, bet technikos, kurios yra panaudotos jo apskaičiavimui yra labai panašios į kitus įsiskverbimo gylio apskaičiavimus.

1.5.1 Atstumas

„Lin-Canny“ algoritmas, aprašytas ankstesniame skyriuje, yra minimalaus atstumo skaičiavimų pavyzdys. Artimiausių savybių sekimas tiesiogiai pristato reikiamus atstumus. Tiesą sakant, minimalių atstumų skaičiavimai gali būti atlikti panašiu būdu kaip atliekant tradicinį „Būlio“ kolizijos nustatymą naudojant *BVH's*.

Tradicinis rekursinis *BVH* paieškos algoritmas, aprašytas anksčiau, testuoja du BV – vienas iš kiekvieno *BVH* – persidengia. Jei, tai tas atvejis, rekursija tęsiasi į jų vaikus. Jei ne – rekursija užbaigiama. Jei pasiekiami du lapai, atliekamas bazinių elementų susikirtimų testas.

Paprasta rekursyvi schema gali būti lengvai modifikuota minimalaus atstumo skaičiavimams: tik bazinių elementų susikirtimo testas turi būti pakeistas atstumo skaičiavimu tarp bazinių elementų ir susikirtimo testas tarp BV turi būti pakeistas atstumo testu tarp BV's. Paieškos metu viršutinė atstumo riba tarp bazinių elementų yra palaikoma kintamuoju S. Šis kintamasis gali būti inicializuotas kartu su atstumu tarp bet kurios bazinių elementų poros. S turi būti atnaujintas, jei randama pora bazinių elementų su mažesniu atstumu.

Akivaizdu, kad BV's su ilgesniu atstumu nei S gali būti atrinkti, kadangi jei BV's turi ilgesnį atstumą, tai privalo būti teisinga ir visiems pridedamiems baziniams elementams. Tai yra būtent tas būdas, kuriuo dauguma autorių naudoja BVH's tam, kad įgyvendinti savo algoritmus.

Kita alternatyva atstumo skaičiavimams yra atstumo laukai, kurie taip pat gali būti kartu su BVH's.

1.5.2 Nuolatinis kolizijos nustatymas

Atmušimo jėgų skaičiavimas skiriančiame atstume gali vesti prie vizualių artefaktų fizika pagrįstose simuliacijose, pvz.: kai objektai atšoka prieš tai, kai jie realiai yra vizualiame kontakte. Be to, jei objektai juda per greitai ar jei laiko žingsnis tarp dviejų kolizijos užklausų yra per didelis, objektai gali praeiti vienas per kitą. Norint išvengti klaidų, tokių kaip tunelio efektas, būtų geriausia apskaičiuoti tikslų abiejų objektų susidūrimo laiką. Norint išspręsti šią *tęstinio kolizijos nustatymo problemą*, kartais vadinamą *dinaminiu kolizijos nustatymu*, buvo pasiūlytos kelios technikos.

Paprasčiausias būdas yra tiesiog pakartotinai panaudoti gerai ištirtus ir stabilius algoritmus, žinomus iš statinio kolizijos nustatymo. Vizualinės interaktyvios taikomosios programos dažniausiai reikalauja 30 kadrų per sekundę atnaujinimo normų, t. y. ten praeina apie 30 milisekundžių laiko tarp

dviejų statinių kolizijos patikrinimų. Pastarieji, „Būlio“ kolizijos nustatymo algoritmai, reikalauja tik kelių milisekundžių, priklausomai nuo objektų konfigūracijos. Taigi, yra daug laiko atlikti daugiau nei vieną užklausą tarp dviejų kadrų. Paprastesnis metodas, vadinamas *pseudo tęstine kolizija*, kuris realizuoja būtent šią strategiją: jis atlieka statinį kolizijos nustatymą su mažesniais laiko žingsniais. Net su didesniu mėginių ėmimo dažnumu yra įmanoma praleisti kontaktus tarp objektų.

Konservatyvi pažanga yra kita paprasta technika, kuri išvengia šių problemų. Objektai pakartotinai paankstinami tam tikru laiko žingsniu, kuris garantuoja neįsiskverbimo suvaržymą. Dažniausiai, norint keletą kartų apskaičiuoti naujas pažangos viršutines ribas naudojamas minimalus atstumas. Konservatyvi pažanga, taip pat suvokiama kaip diskretus kinetinių duomenų struktūrų protėvis.

Poslinkio tūriai garantuoja konservatyvias ribas jų pagrindiniams baziniams elementams ir, todėl poslinkio *BVH‘s* gali būti peržiūrėti panašiai į diskretų *BVH‘s*. Tačiau, reikalingas papildomas tęstinis kolizijos testas baziniams elementams, kad pasiekti tikslų susidūrimo laiką. Tiesą sakant, šie testai (taip pat testai tarp BV‘s) priklauso nuo bazinių elementų trajektorijų, kurie paprastai yra nežinomi tarp simuliacijos žingsnių. Dažnai naudojama paprasta tiesinė interpoliacija, siekiant priartinti judesį, esantį tarpe. Trikampių porai, tai atneša šešis paviršius - viršūnę ir devynis briauna-briauna testus. Kiekvienas iš šių elementarių testų reikalauja vieno - išspręsti kubinę lygtį. Toks skaičiavimais gana brangus užsiėmimas. Todėl, kai kurie autoriai papildomai pasiūlė savybėmis pagrįstus išankstinius testus, kaip pavyzdžiui poerdvinius „Tang et al.filtrus“ ar papildomus BV‘s, kaip „k-DOPs“ briaunoms.

Tačiau, besitęsiančio kolizijos nustatymo apskaičiavimas vis dar yra per brangus realaus laiko taikomosios programoms, ypatingai, kai vienu metu yra simuliuojama daug kompleksinių dinaminių objektų.

1.5.3 Įsiskverbimo gylis

Minimalus atstumas nėra geras matas nustatyti atmušimo jėgas ir tikslaus susidūrimo laiko apskaičiavimas, naudojant nuolatinį kolizijos nustatymą, yra per daug laiko reikalaujantis realaus laiko aplikacijų darbas. Todėl, vienas mokslininkas tirdamas išvystė kitą įsiskverbimo matą: *įsiskverbimo gylį*. Plačiai naudojamas apibrėžimas apibūdina jį, kaip atstumą, kuris atitinka trumpiausią slenkamąjį judėjimą, reikalingą atskirti du susikertančius objektus.

Kai kurie autoriai apskaičiavo lokalias įsiskverbimo gylio aproksimacijas, jei objektai susikerta

keliose atskirose zonose. Todėl, įsiskverbimo zonos buvo atitvertos į susijusius regionus ir atskirai apskaičiuotas kiekvieno šio regiono lokalus įsiskverbimo gylis.

Įsiskverbimo gylio apskaičiavimas yra aktyviai tyrinėjamas, kadangi jis yra brangus ir yra tinkamas tik labai greitose mašinose. Tačiau, klasikinio įsiskverbimo gylio naudojimas turi kitą svarbų trūkumą: slenkamojo judėjimo vektorius nėra tęstinis taškuose, esančiuose ant vidurinės ašies. Taip pat, nėra paprasta modeliuoti daug vienalaikių kontaktų

1.6 Kritinio laiko kolizijos nustatymas

Nepaisant prieinamos didelės apskaičiavimo galios, kolizijos nustatymo algoritmų veikimas yra vis dar kritinis daugelyje aplikacijų, ypatingai jei reikalingas laiko biudžetas negali būti viršytas. Ši problema dažniausiai iškyla interaktyviose realaus laiko aplikacijose, kur sklandžiai vaizdo atsakomajai reakcijai reikalingas mažiausiai 30 fps kadravimas. Todėl, lieka tik 30 ms fizika pagrįstos simuliacijos pavaizdavimui. Vaizdavimo žingsniui egzistuoja lygių ir detalių (angl. *levels-of-details (LOD)*) technika, skirta sumažinti grafikos specializuotų modulių darbo krūvį. Pagrindinė idėja yra saugoti geometrinius duomenis keliose mažėjančiose rezoliucijose ir pasirinkti atvaizdavimui tinkamą *LOD* pagal požiūrio atstumą. Fizika pagrįstai simuliacijai gali būti panaudotos panašios technikos, tiksliau kolizijos nustatymo žingsniui. Taigi, taip vadinamas *kritinio laiko kolizijos nustatymas* sumažina skaičiavimo laiką tikslumo kaina.

Paprastai kritinio laiko kolizijos nustatymo metodai pagrįsti kompleksinių objektų supaprastinimais, kaip, pavyzdžiui vizualios *LOD* atvaizdavimais. Tai, galima padaryti tiesiogiai arba netiesiogiai. Be to, jie dažnai naudoja kadras į kadrą sąryšį, kadangi fizika pagrįstose simuliacijose paprastai neturėtų būti trūkių ir kontakto informacija tarp dviejų kolizijos patikrinimų labai nesiskiria.

Pavyzdžiui, iš vienalaikės *BVH* paieškos išvestas „BVTT“ (žr. Pav. 1.4 prieš tai buvusiame skyriuje) kiekviename mazge laiko užklausos tarp dviejų BV‘s rezultatą. Tos BV poros, kur paieška nustoja statyti „BVTT“ *atskyrimo sąrašą*. Esant dideliam darnumui, nereikia iš naujo pradėti paieškos *BVH‘s* šaknyse kiekvienai užklausai, bet šis sąrašas gali būti tiesiogiai panaudotas iš naujo.

Tiesą sakant, klasikinė vienalaikė *BVH* paieška gerai „paskolina save“ kritinio laiko kolizijos nustatymui: paieška gali būti paprastai pertraukta, kai laiko biudžetas yra išeikvotas.

1.6.1 Kolizijos nustatymas liečiamosiose aplinkose

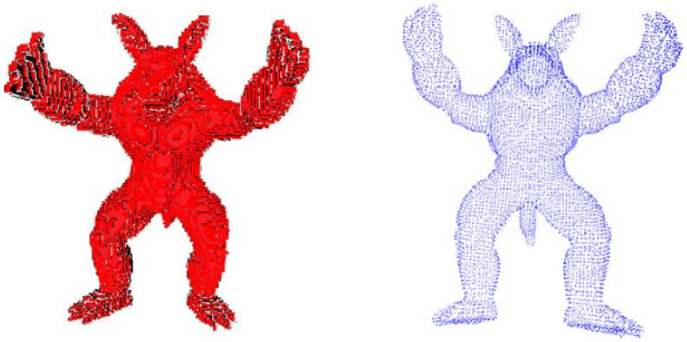
Beveik visi kolizijos nustatymo metodai, aprašyti anksčiau, yra visų pirma sukurti dirbti bent vizualiame realiame laike. Kaip jau buvo minėta įvade, sklandžiam vaizdo pojūčiui reikalingos 30 Hz atnaujinimo normos, tuo tarpu lietimo perteikimas reikalauja 1000 Hz dažnio atnaujinimo realistiškam liečiamajam pojūčiui.

Nei vienas iš anksčiau aprašytų metodų, ypatingai tų, kurie apskaičiuoja įsiskverbimo gylius ar susidūrimo laikus, negalėtų būti pagreitintas 30-ies veiksniu dėl pagrįsto scenos sudėtingumo liečiamosiose aplinkose. Todėl, kolizijos nustatymas haptikoje dažnai veda prie tolimesnių supaprastinimų tam, kad garantuoti aukštą dažnį, bet taip pat ir apskaičiuoti patikimas jėgas.

1.6.1.1 „3DOF“, „6DOF“

Ankstyvojoje haptinės žmogaus kompiuterijos istorijoje, 20 a. devyniasdešimtųjų pradžioje pagrindinis supaprastinimas paveikė tiek haptinės techninės įrangos sąsajų dizainą, tiek kolizijos nustatymą: vietoj sudėtingo standžių kūnų sąveikos simuliavimo, šioje sąveikoje buvo panaudotas tik vienintelio taško zondas. Tai reikalavo tik trijų jėgos komponentų apskaičiavimo zondo galiuke. To rezultate daug „3 DOF“ įrenginių, kaip pavyzdžiui „SensAble Phantom Omni Massie“ ir „Salisbury“, pateko į rinką ir buvo atlikta daug „3 DOF“ haptinio vaizdavimo algoritmų tyrimų.

Daugelis aplikacijų, kaip treniravimasis ar virtualaus prototipavimas reikalauja sąveikos su sudėtingais virtualiais instrumentais vietoj vienintelio zondo taško tam, kad užtikrinti pakankamą realizmo laipsnį. Kai tik haptinis zondas apima 3D objektus, papildomas sukimo momentų pavaizdavimas tampa svarbus. Taip pat, gali atsirasti vienalaikiai kontaktai su aplinka. Tai, ryškiai padidina kolizijos nustatymo ir kolizijos atsako sudėtingumą. Iš tikrųjų išbaigta „6 DOF“ standaus kūno simuliacija, apimančią jėgas ir sukimosi momentus, turi įvykti per 1 ms. Statinė aplinka yra diskretizuota į vokselių (angl. *voxel*) rinkinį, kur dinaminis objektas yra aprašomas taškų rinkinių, kuris reprezentuoja jo paviršių (žr. Pav. 1.5). Užklausos metu, kiekvienam šių taškų jis nustatomas paprastu „Būlio“ testu ar jis yra užpildytame tūrio elemente, ar ne. Šiandien vokselizacija gali būti efektyviai apskaičiuojama naudojant GPU.

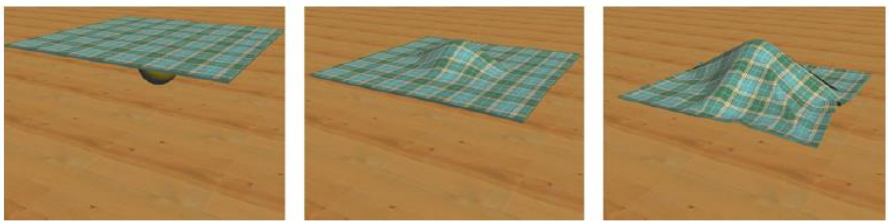


1.5 pav. *The Voxmap-Pointshell 6 DOF* haptinio vaizdavimo metodas naudoja dvi skirtingas duomenų struktūras: vokselizacija (angl. *voxelization*) (kairėje) ir objektų paviršiaus taškų mėginių ėmimą (dešinėje)

1.7 Deformuojamų objektų kolizijos nustatymas ir GPU pagrįsti metodai

Dažniausiai kolizijos nustatymo algoritmai priklauso nuo iš anksto apskaičiuotų duomenų struktūrų kaip *BVH's*. Tai, veikia gerai, kol objektų geometrija nepakinta, t. y. jei objektai yra standūs. Tačiau, mūsų pasaulis egzistuoja ne tik iš standžių objektų bet ir iš deformuojamų objektų, kaip audinys (žr. Pav. 1.6). Todėl realistinė simuliacija turėtų gebėti valdyti deformuojamus modelius. Be audinio simuliacijos, populiarios deformuojamos aplikacijos apima personažų animaciją, chirurgijos simuliaciją ir lūžius.

Papildomas iššūkis deformuojančių objektų kolizijos nustatymui yra tikimybė, kad vieno objekto dalis susikerta su kitomis to pačio objekto dalimis, taip vadinamos kolizijos su savimi (angl. *self-collision*).

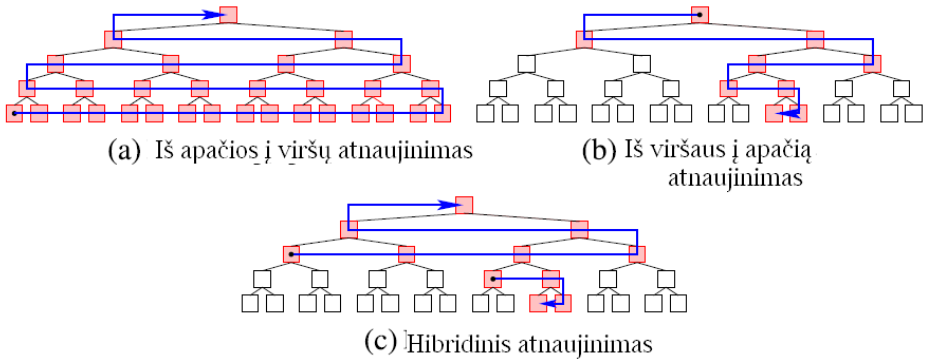


1.6 pav. Deformuojamieji objektai, kaip pavyzdžiui audinys, reikalauja specialių algoritmų, kadangi iš anksto apskaičiuotos duomenų struktūros tampa klaidingomis po deformacijos. Be to, gali susidaryti kolizija tarp objekto dalių.

Faktiškai, *BVH's* gali būti lengvai įdarbintas rasti kolizijas su savimi, dažniausiai tikrinant objekto *BVH* su savimi ir atmetant kolizijas tarp gretimų bazinių elementų. Papildomai, tolimesniam pagreitinimui, gali būti panaudotos tokios technikos, kaip normalių kūgių hierarchijos arba galios diagramos.

Yra įrodyta, kad *BVH's* yra labai efektyvūs standiems objektams ir be to jie gali būti laisvai praplėsti kolizijos su savimi nustatymui. Dėl šių priežasčių tyrėjai taip pat norėtų juos panaudoti deformuojamiems objektams. Kadangi *BVH's* tampa netinkami po deformacijų, keli metodai išsprendė šią problemą: lengviausias metodas atstatyti *BVH* nuo pradžių po kiekvienos deformacijos. Deja, paaiškėjo, kad tas skaičiavimais per brangus metodas. Net modernaus GPU pagreitinimas negali garantuoti realaus laiko *BVH* konstrukcijos veikimo pagrįstai sudėtingose scenose.

Kaip kompromisas buvo pasiūlyta hibridinė atnaujinimo strategija: medžiui, su gyliu n, iš pradžių pirmas n/2 turėtų būti atnaujinamas iš apačios į viršų. Apatiniai mazgai turėtų būti atnaujinti iš viršaus į apačią, kolizijos paieškos metu (žr. pav 1.7).



1.7 pav. Skirtingos *BVH's* atnaujinimo strategijos

GPU pagrįsti metodai

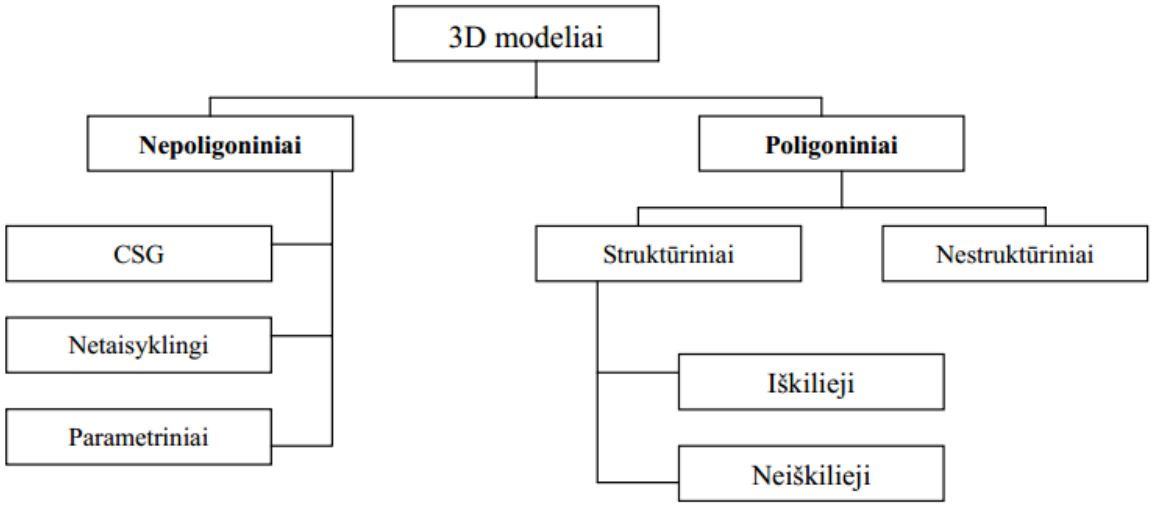
Populiarūs deformuojamų objektų, kaip, pavyzdžiui, masės ir spyruoklių sistemų realaus laiko simuliacijos metodai, bet ir kelių kūnų simuliacijos, gali būti lengvai sulygiagretintos. Todėl, jos yra idealiai pritaikytos moderniai GPU architektūrai. Vadinasi, tai yra akivaizdu išvystyti kolizijos nustatymo schemas, kurios dirba tiesiogiai grafikos techninėje įrangoje, o ne kopijuoja duomenis atgal ir pirmyn tarp pagrindinės atminties ir GPU atminties.

Tiesą sakant, GPU - pagrįsti algoritmai buvo pasiūlyti visoms kolizijos nustatymo specializuotų metodų sekos dalims: plačiajai fazei, siaurajai fazei ir net bazinių elementų testams.

Pirmieji metodai rėmėsi mažiausiai „OpenGL 1.6“ fiksuotos funkcijos grafikos specializuotų metodų seka ir naudojo vaizdo erdvės technikas.

1.8 Kolizijos uždavinių klasifikacija

Vienas iš kolizijos aptikimo uždavinių klasifikavimo kriterijų yra trimačių modelių geometrija, kurią būtų galima suskirstyti kaip parodyta 1.8 pav [15].



1.8 pav. Trimačių geometrinių modulių klasifikacija

Poligoniniai objektai – dažniausiai naudojami modeliai kompiuterinėje grafikoje ir modeliavime. Jie gana paprasti ir universalūs. Bendriausias jų tipas – *nestruktūriniai* poligonai. Tai poligonų rinkiniai, kurie geometriškai nesujungti ir neturi jokios topologinės informacijos. Jei poligoninis modelis turi ryškiai apibrėžtą vidų ir išorę – tai jis yra *struktūrinis* modelis, kuris savo ruožtu gali būti *iškilusis* ir *neiškilusis*. *„CSG“ (Constructive Solid Geometry)* - tai objektai sudaryti iš tokių primityvų kaip blokai, sferos, cilindrai, kūgiai. Jie formuojami „sujungiant“ primityvus tam tikromis matematinėmis operacijomis (sąjunga, sankirta). Netaisyklingi paviršiai apibrėžiami netaisyklingomis funkcijomis. Tai uždari „vamzdžiai“, algebriniai paviršiai, sukimosi paviršiai, „quadrics“. Parametriniai paviršiai, priešingai nei „CSG“ ar netaisyklingi paviršiai, nesudaro pilno uždaro objekto, tai tiesiog paviršiaus riba.

1.9 Algoritmų palyginimas

Dauguma viešų duomenų sistemų yra priskiriamos poligoniniams modeliams ir tik kai kurie yra priskiriami didelėms aplinkoms, kurios sudarytos iš kelių judančių objektų. Beveik neįmanoma

sulyginti skirtingų algoritmų ir sistemų, nes jų veikimo principai skiriasi, priklausomai nuo modeliavimo aplinkos (modelių, įvairių kontaktų, užklausos tipų, judesio aprašymų ir t.t.) bei kitų faktorių. Pateikiami algoritmai yra surašyti chronologiška tvarka pagal jų išleidimą bei trumpai aprašyti jų ypatumai.

Apžvelgus visą eilę kolizijos aptikimo metodų ir algoritmų matyti, jog poligoniniams objektams skirtų algoritmų yra žymiai daugiau nei kitiems nepoligoninių objektų algoritmams. Visus metodus galima suskirstyti į fundamentalius (t.y. bazinius) ir į išvestinius (sukurtus naudojantis baziniais metodais).

Pabandysiu apibendrinti algoritmus skirtus poligoniniams objektams. Vieni algoritmai pasinaudoja iškiliųjų poliedrų savybėmis, kurios leidžia pagreitinti užklausų apdorojimą, kiti algoritmai gali dirbti su bet kokiais trikampių rinkiniais ir poliedrais.

Trumpas palyginimas:

- **„Solid“.** „Solid“, susidūrimų aptikimui naudoja pagal ašį sulygiuotomis gaubiančiomis dėžėmis (angl. *axis-aligned bounding boxes*). Paskutiniam susidūrimų testui tarp „AABB“ hierarchijų naudojama van den Bergen aprašyta schema. „Solid“ biblioteka taip pat tinka geometrijos deformacijų nustatymui.
- **„V-Collide“**. „V-Collide“, pasiūlyta Hudson , yra biblioteka su paprasta sąsaja „I-Collide“, remiamasi „RAPID“ biblioteka. Pirmajame žingsnyje „sweep-and-prune“ algoritmas naudotas potencialiai sutampančių objektų porų aptikimui. Antrajame žingsnyje „RAPID“ biblioteka naudota tiksliam porinio testo tarp objektų porų atlikimui. Galimai susiduriančių trikampių porų radimui naudojamas orentuotu gaubiančių dėžių testas.
- **„PQP“.** „PQP“ taip pat paremta „RAPID“ biblioteka. Kaip ir „RAPID“ atveju, „PQP“ naudoja orientuotas gaubiančiasias dėžes (angl. *oriented bounding boxes (OBB)).* Be to, „PQP“ taip pat gali apskaičiuoti atstumą tarp artimiausios taškų poros. Atstumo ir nuokrypių užklausoms naudojamas kitas BV tipas, vadinamosios sferos.
- **„Opcode“.** Terdiman pristatyta „Opcode“ biblioteka – tai porinių susidūrimų testų biblioteka. Joje naudojamos „AABB“ hierarchijos, o ypatingas dėmesys skiriamas atminties optimizavimui. Dėl šios priežasties pasitelkiamos belapės *BVH*, t.y., hierarchijos, iš kurių pašalinti lapų mazgai. Siekiant didesnės spartos, rekursyvios paieškos metu naudojami pirminiai-BV sutapimo testai, tuo tarpu visos kitos šiame skirsnyje aprašytos bibliotekos naudoja tik pirminius-pirminius testus ir BV- BV testus. Kaip ir „Solid“ ar „Opcode“ taip pat palaiko deformacijas.

- **„Dop-Tree“.** Zachmann aprašyta „BoxTree“ biblioteka – tai optimizuotos atminties „AABB“ medžių versija. Vietoj 6 reikšmių išsaugomos tik dvi plokštumos. Siekiant paspartinti simuliaciją, bibliotekos palaiko tinklelį.
- **„Box-Tree“. „**Dop-Tree“ [Zachmann] bibliotekoje naudojami diskretūs orientuoti daugiakampiai (kur k yra orientacijų skaičius). Biblioteka palaiko įvairias orientacijas. Autorius paminėjo, kad k= 24 garantuoja didžiausią efektyvumą. Todėl, savo matavimams pasirinkau būtent šį skaičių. Orientacijų rinkinys yra fiksuotas. Ši biblioteka taip pat palaiko n-kūnų simuliaciją per tinklelius.

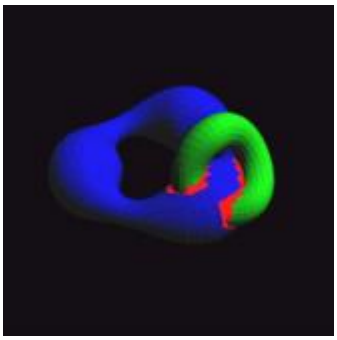
1.9.1 „V-COLLADE“ algoritmas

„V-COLLIDE“ [13] yra n-kūno procesorius, pastatytas ant „RAPID“ sistemos. Kai jūsų kliento programa liepia „V-COLLIDE“, kur visi modeliai yra pasaulio erdvėje, „V-COLLIDE“ atlieka greitą šlavimo-ir-genėjimo operaciją, nuspręsti, kuri modelių pora yra potencialiai kontakte ir tada kiekvienai potencialiai kontaktų porai ji naudoja „RAPID“ nustatyti tikrąjį kontaktų statusą.

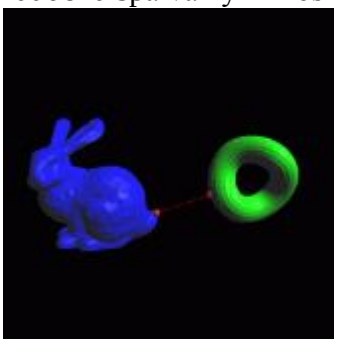
„V-COLLIDE“ prisimena, kur yra visi modeliai, ir jūs galite atnaujinti kai kurias arba visas modelių vietas pasakant „V-COLLIDE“ jų naujas vietas – „RAPID“ daro laipsniškus pakeitimus potencialioje kontakto informacijoje su kiekviena vietos modifikacija.

„V-COLLIDE“ taip pat dirba su poligonų „sriubomis“, ir praneša tik kontakto būseną (bet ne atstumą). Jei turite daug sudėtingų objektų, daugelis kurių gali susidurti su kitais objektais, tada „V-COLLIDE“ yra tikriausiai teisingas pasirinkimas. Papildomai, atnaujinant modelio pozicijas, kliento programa taip pat galit pridėti arba ištrinti modelius iš „V-COLLIDE“ susidūrimo aptikimo variklio valdomos kolekcijos. „V-COLLIDE“ taip pat palaiko daugialypius nepriklausomus susidūrimo nustatymo variklius.

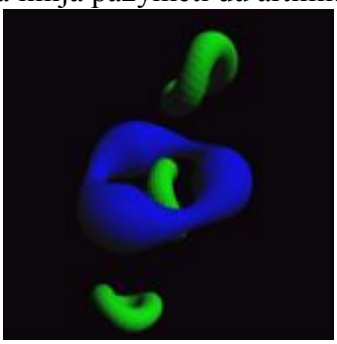
1.9.2 „PQP“ algoritmas



1.9 pav. Pirmojoje nuotraukoje raudona spalva žymimos vietos kur susikerta du objektai.



1.10 pav. Antroje nuotraukoje raudona linija pažymėti du artimiausi taškai tarp triušio ir riestainio.



1.11 pav. Paskutinėje nuotraukoje yra parodyti trys žingsniai, kur susidūrimas tarp objektu neįvyksta. Kelias skaičiuojamas su kelio planuotoju (angl. *path-planner*), naudojant tolerancijos patikrinimą.

Palyginti su „RAPID“, „PQP“ [13] taip pat teikia paramą atstumo skaičiavimui ir leistino nuokrypio (tolerancijos) patikrinimo užklausoms (papildomai susidūrimo nustatymui). Jo „API“ panašus į „RAPID API“.

Jis taikomas bendriesiems daugiakampio modeliams ir nereikalauja jokios topologinės informacijos. Lyginant du modelius, „PQP“ palaiko daug įvairių užklausų. Jis panaudoja nuvalytus sferinius tūrius kaip BV pasirinkimą nuotolio užklausoms. Be to, tai leidžia klientui programuoti lankstumą, naudojant daugiau nei vieną ribojantį tūrį duotai užklausai.

Tikslas

„PQP“ yra biblioteka, atliekanti trijų rūšių artumo užklausas, naudojanti porą geometrinių modelių, susidedančių iš trikampių:

- *susidūrimo aptikimo* - nustatyti ar du modeliai persidengia ir pasirinktinai visi trikampiai, kurie persidengia.
- *Atstumo skaičiavimas* - apskaičiuojant minimalų atstumą tarp modelių poros, t.y. atstumą tarp artimiausios poros taškų.
- *Tolerancijos patikrinimas* - nustatyti ar du modeliai yra arčiau, ar toliau nei tolerancijos atstumas.

1.9.3 „SOLID“ algoritmas

„SOLID“ [13] yra trimačių objektų, patiriančių griežtą judesį ir deformaciją, susidūrimo aptikimo biblioteka. „SOLID“ - skirtas būti naudojamas interaktyvios 3D grafikos aplikacijose ir ypač tinka „VRML“ aprašytų objektų ir pasaulių susidūrimo aptikime.

Kai kurios jo savybės:

- Objektų figūros yra reprezentuojamos primityvių formų (dėžutė, kūgis, cilindras, sfera) ir politopų kompleksų (atkarpos, išgaubti daugiakampiai, išgaubti briaunainiai). Gali būti naudojama vienintelė forma parodyti kelis objektus.
- Judesys yra detalizuojamas pavertimu, sukimu, nevienodais kiekvieno judančio objekto lokalios koordinačių sistemos masteliais. Šie pokyčiai gali būti absoliutūs arba santykiniai palyginus su prieš tai buvusiu kadru. Lokali koordinačių sistema, taip pat gali būti nustatyta pagal šešiolikos plūdžių ar dvigubą masyvą, reprezentuojantį 4x4 stulpelių transformacijos matricą, kaip naudojama „OpenGL“.

Sudėtingų formų deformacijos gali būti detalizuojamos naudojant kliento apibrėžtus viršūnių masyvus.

- Susidūrimo atsakas yra apibrėžiamas kliento pagal atgalinių funkcijų priemones. Atsakymas gali būti apibrėžtas už objektų porą, už visas poras, turinčias specifinį objektą ir kaip numatyta visų objektų porų.
- Atsako atgalinės funkcijos gali naudoti susidūrimo duomenis, aprašančius susiduriančių objektų poros konfigūraciją. Kadangi, susidūrimo duomenys gali būti naudojami kaip taškas, bendras abiem objektams ir artimiausia objektų iš prieš tai buvusio kadro taškų pora. Pastarasis atsako tipas gali būti naudojamas priartinant susidūrimo plokštumą fizikos pagrindo simuliacijose.

- Kadro darnumas išnaudojamas išlaikant artimų objektų porų rinkinį ir šių porų atskiriančių ašių spartinimą. Ši funkcija yra neprivaloma ir gali būti įjungta/išjungta bet kuriuo simuliacijos metu.

1.9.4 „OPCODE“ algoritmas

„OPCODE“ [12] yra nauja maža kolizijos nustatymo biblioteka. Ji yra panaši į populiarius paketus, kaip pavyzdžiui „SOLID“ ar „RAPID“, bet labiau „draugaujanti“ su atmintimi ir dažniausiai greitesnė.

SAVYBĖS

- C++ „interfeisas“, išvystytas operacinei sistemai „Windows“, naudojant VC++ 6.0
- Dirba ant arbitriškų tinklelių (išgaubtų ir neišgaubtų) net daugiakampių sriubų
- Dabartinis įgyvendinimas naudoja „AABB-medžius“
- Pristato bazinių elementų-BV persidengimo testus rekursinės kolizijos užklausų metu (kadangi standartinės bibliotekos pasikliauna tik bazinių elementų – bazinių elementų ir BV-BV testais)
- Pristato belapius medžius, t.y. kolizijos medžius, kurių lapų mazgai buvo pašalinti
- Palaiko kolizijos užklausas ant kvantizuotų medžių (išskleistų skrydyje)
- Palaiko „pirmasis kontaktas" ar „visi kontaktai" režimus („à la RAPID“)
- Naudoja laiko nuoseklumą „pirmojo kontakto" režimui (~nuo 10 iki 20 kartų greičiau, naudingas standaus kūno simuliacijoje perpjovimo metu)
- Atminties pėdsakas yra 7.2 kartų mažesnis nei „RAPID“, kuris yra idealus konsolės žaidimams su limituotais „RAM“ (tiesą sakant, jei jūs naudojate nepakeistą „RAPID“ kodą, naudojantį dvigubą tikslumą, jis daugiau nei 13 kartų mažesnis...)
- Ir pagaliau, jis dažnai veikia greičiau nei „RAPID“ (pagal „RDTSC“, kartais daugiau nei 5 kartus greičiau, kai objektai yra giliai persidengiantys)
- Atlikimas dažniausiai artimas „RAPID“ artumo situacijose.
- Vėrimų, plokštumų ir tūrių užklausos (sfera, „AABB“, „OBB“, „LSS“)
- Nuskaitymo ir apkarpymo (angl. *Sweep-and-prune*) algoritmas ir „radix“ - pagrįstas dėžės apkarpymas
- Dabar dirba su deformuojamais tinkleliais („OPCODE 1.3“)

- Hibridiniai medžiai („OPCODE 1.3“) laiko maksimaliai 16 trikampių lapui ir reorganizuoja kliento trikampių sąrašus, kas reikalauja grubiai 16 kartų mažiau „RAM“ nei „OPCODE's“ standartiniai medžiai. Geriausiu atveju, tai sumažėja iki 1.25 baitų / trikampiui, kas yra *115 kartų mažiau* nei „RAPID OBB“ medžių. Greitis yra dažnai nežymus ir tūrių užklausos gali iš tikrųjų veikti greičiau nei „OPCODE 1.2“ dėl mažesnių „cache“ atminties praradimų (šiuo atžvilgiu, padeda ir klientų masyvų reorganizavimas). Jie taipogi greitesni deformuojamiems tinkleliams.

1.9.5 „Dop-Tree“ algortimas

Diskrečios orientacijos politopai (angl. Discrete orientation polytopes (DOP's)) [6] yra išgaubti politopai, kurių paviršiai turi vieninteles normales, kurios ateina iš fiksuotų orientacijų rinkinių (k-DOP's). Tam, kad *DOP's* pritaikyti intervalo testą, jis taip toliau apriboja orientacijų rinkinį, kad kiekvienai rinkinio orientacijai yra taip pat ir anti-lygiagreti. Todėl, šis specialus „k-DOP's“ tipas gali būti laikomas ašies sulygiuotų dėžių apibendrinimu. Atliekant testus buvo nustatytas optimalus orientacijos skaičių, kuris yra 24.

Remiantis bendrąja hierarchine duomenų struktūra, yra pristatomas greitas algoritmas, skirtas savavališkų daugiakampių standžių objektų tiksliam kolizijos nustatymui. Gali būti patikrinta objektų, sudarytų iš šimtų tūkstančių daugiakampių, interaktyvių normų kolizija. Iš anksto paskaičiuota hierarchija yra diskrečiųjų orientuotų politopų (DOP's) medis. Efektyvus DOP's perlygiavimo šių medžių paieškos metu metodas leidžia panaudoti intervalo testus, skirtus nustatyti persidengimą tarp DOP's. Duomenų struktūra yra labai efektyvi, kalbant apie atmintį ir konstravimo laiką. Buvo atlikti platūs eksperimentai su sintetiniais ir realaus pasaulio „CAD“ duomenimis, kad išanalizuoti duomenų struktūrų veikimą ir atminties naudojimą. Palyginimas su „OBB“ medžiais rodo, kad *DOP* medžiai yra efektyvūs kolizijos užklausos laiko požiūriu ir labiau efektyvūs atminties naudojimo ir konstravimo laiko požiūriu.

1.9.6 „Box-Tree“ algoritmas

Šie, ašimi lygiuoti ribojantys tūriai yra vadinami dėžiniais medžiais (angl. *Box Tree*) [6]. Žinoma, kai objektai juda, jie nebėra sulygiuoti, atsižvelgiant į pasaulius koordinuojantį kadrą. Todėl,

naudojant šio tipo BV‘s iškyla užduotis sugalvoti algoritmus efektyviems besivartančių ribojančių dėžių persidengimo testams. Aštuonmedžiai (angl. *octree*) yra specialus „Box-Tree“ atvejis. Tačiau, „Box-Tree“ leidžia geresnę subalansuotumo kontrolę. Kiekvienas „Box-Tree“ mazgas saugo tik: skeliančios ašies vardą, dvi pozicijas, dvi rodykles (į kairę ir dešinę sub-dėžę) ir tik lapams, rodyklę į poligoną (us), susijusį su lapu.

Dėžinio medžio duomenų struktūra yra binarinis medis, kuris nėra vienodas, prisitaikantis erdvinis poskyris. Kadangi jis yra nevienodas, yra manoma, kad jis gali lenkti metodus, naudojančius vienodus poskyrius, tokius kaip tinkleliai. Be to, kadangi tai yra pritaikoma, tai pagerina laukiamą veikimo laiko atlikimą. „Box-Tree“ lapai turi kraštus ir daugiakampius, kurie sudaro susijusį daugiasienį. Medžio konstravimo algoritmas bando sukurti optimalų medį, atsižvelgiant į kolizijos nustatymo algoritmą. Rezultatai rodo, kad „Box-Tree“ algoritmas veikia daug geriau nei paprasti (potencialiai O (n2)) algoritmai, kai objekto sudėtingumas yra virš tam tikro lygio (apie 200 daugiakampių/objektui). Dėl rekursyvaus algoritmo įmantrumo pobūdžio, jis bet kuriuo metu gali būti pertraukiamas, kai aplikacija pasirenka padaryti tai, kad apdrausti konstantinę kadro normą. Todėl, šis algoritmas yra geras kandidatas pritaikomam krūvio balansavimui. Kiekvienam objektui tik kartą sukuriama hierarchinė duomenų struktūra. Ji neturi būti transformuojama, kai objektas juda.

1.10 Išvados

Kolizijų nustatymo tarp objektų problema buvo ištirta ir dažnai motyvuota patobulinto veikimo poreikio principu. Nors, kai kurie mokslininkai daugiausia dėmesio skyrė skaičiavimo efektyvumui, kiti sutelkė dėmesį į rezultatų tikslumą. Kai kuriais atvejais šis tyrimas vedė prie atviro kodo programinės įrangos bibliotekų ir įrankių išleidimo, įgalinančio kitus tyrėjus tobulinti šią sritį. Kolizijos nustatymo algoritmų lyginamosios analizės plėtojimas yra reikalaujantis ypatingų pastangų dėl įvairių faktorių, kurie gali žymiai paveikti algoritmų veikimą. Įtrauktų objektų tipai ir jų santykinės transformacijos, ir dydžiai gali turėti įtakos užklausų skaičiavimo laikui. Todėl, taikymo tipas yra vienas iš lemiančių veiksnių pasirenkant tinkamą kolizijos aptikimo strategiją. Šiame darbe yra aprašyti pagrindiniai kolizijos nustatymo algoritmai.

Vienas būdas, kuris apskritai gerai veikia yra scenos išskaidymas į atskiras erdvines sritis tam, kad įgalinti veiksmingą strategiją sumažinti problemą. Tačiau, ši strategija paprastai reikalauja išankstinio apdorojimo etapo ir efektyvių technikų dinaminių objektų stebėjimui. Ribojančiojo tūrio hierarchija gali būti sukonstruota kiekvienam trijų dimensijų objektui tam, kad palengvinti artumo užklausų tarp kiekvienos objektų poros gretinimą. Susikirtimų tarp ribojančiųjų tūrių, tokių kaip sferos, su ašimi sulygiuotų ribojančių dėžių ir orientuotų ribojančių dėžių nustatymas yra santykinai nebrangi procedūra, palyginti su individualių daugiakampių testavimu, kuris gali būti įtrauktas į procedūras. Norėdami nustatyti tikslius kontakto ir paviršiaus statmenų taškus, galime apsvarstyti ar naudoti individualius daugiakampius viduje ribojančių tūrių, kurie suformuoja hierarchijos lapus.

Kelia susirūpinimą tai, kaip galima gauti geriausią kompromisą tarp greičio ir tikslumo. Potenciali kolizijų nustatymo tyrimų ateitis gali apimti tolimesnį tęstinių kolizijų naudojimą ir lygiagrečių struktūrų išnaudojimą. Didėjantis GPU atlikimas užtikrina, kad jie tampa galingu įrankiu ne tik atvaizdavime, bet ir kitoms reikmėms, pvz. kolizijų aptikimui. Tačiau, reikia stengtis užtikrinti pusiausvyrą tarp CPU ir GPU, apsisaugoti nuo nepageidaujamų uždelsimų abiejuose procesoriuose.

2 Projektinė dalis

Teorinis sudėtingumas beveik visų hierarchinių susidūrimo aptikimo požiūrių yra blogiausiu atveju poligonų skaičiaus kvadratas. Tai tiesa, nes yra įmanoma statyti dirbtinius objektus, pavyzdžiui, „Chazelle“ briaunaininį (žr 0.2 pav), kur kiekvienas poligonas susiduria su visais poligonais iš kito objekto (žr. įvado skyrių). Tačiau, praktikoje susidūrimo nustatymo sudėtingų objektų kvadratinis įdirbimo laikas susideda iš milijonų poligonų. Tokiu atveju, nėra prasmės naudoti kvadratinius algoritmus realaus laiko scenarijuose. Tiesą sakant, tokios situacijos nėra dažnos realiuose scenarijuose, nes tikri objektai taip nesielgia. Tai kelia klausimą: kas verčia objektus elgtis tinkamai?

Tačiau teoriškai „geras elgesys“ yra tik viena medalio pusė. Šiandien vartotojas gali pasirinkti tarp įvairių skirtingų susidūrimo aptikimo bibliotekų, kurios remiasi įvairių *BVH's* (žr 1 skyrių).

2.1 Projekto planas

Šiame darbe bus atliekamas erdvinių objektų kolizijų metodų palyginimas. Palyginti algoritmus ir jų realizacijas viena su kitu yra ne trivialu. Tam reikia standartizuoto susidūrimo aptikimo palyginimo paketo. Tad, šiame darbe bus naudojamas lyginamosios analizės rinkinys, kuris leidžia sistemingai lyginti standžių objektų statinius susidūrimus ir nustato atstumą tarp objektų. Į šį įrankį buvo įdiegti visi aukščiau išvardinti kolizijos aptikimo algoritmai, tad visu algoritmų rezultatus galėsime matyti vienoje diagramoje. Tam, kad analizės rezultatai būtų tikslesni, kolizijos bus tikrinamos su skirtingais objektais ir naudojant skirtingas rezoliucijas.

2.2 Techninės ir programinės įrangos reikalavimai

Techninės įrangos reikalavimai

Visi šiuolaikiniai kompiuteriai atitinka techninės įrangos reikalavimus.

Programinės įrangos reikalavimai

- „OpenSG 1.6“ arba naujasnės versijos
- „Gcc 3.5“ arba „Visual Studio 2003“ ar naujesnių versijų

- „Gnuplot“ (naudojamas automatinėms diagramoms paišyti)
- „Microsoft Windows XP“, „Vista“, 7, 8 arba „Windows Server 2003“ arba 2008.

2.3 Algoritmų realizavimas

Projektuojant pasirinktus algoritmus buvo vadovaujamasi trimis pagrindiniais žingsniais:

1. Integracija. Sukuriami visi reikalingi komponentai atvaizduoti simuliaciją. Taip pat, šiame žingsnyje integruojami objektų parametrai (greitis ir pozicija), reikalingi judėjimui simuliuojamoje erdvėje.

2. Pasirinkto algoritmo struktūros kūrimas. Šiame etape realizuojasi testavimo įrankis, į kurį integruojami pasirinkti algoritmai (*V-Collide, PQP, Solid, Opcode, BoxTree, Dop-Tree*).

3. Susidūrimų paieškos vykdymas. Šis, paskutinysis žingsnis, yra reikalingas skaičiavimams.

Taip pat yra laikomasi darbo eigos principų, kurie pavaizduoti iliustracijoje žemiau ([10] žr. 2.1 pav.)

Specializuotų modelių seka (toliau SMS). Duomenų seka kolizijos nustatymo specializuotų modelių seka. Objekto prižiūrėtojas pasako kolizijos nustatymo moduliui apie, bet kokius objektus, kurie buvo pašalinti, bet kurio kito modulio. Šie pridedami į sąrašo buferį. Pagaliau buferis yra ištuštinamas ir, bet kuri globali randanti kaimynus duomenų struktūra yra atnaujinama pagal naujas pozicijas. Tada, iš kolizijos susidomėjimo matricos yra sugeneruojamas porų sąrašas. Šis porų sąrašas yra filtruojamas vieno ar kelių kaimynus randančių algoritmų (pvz., tinklelio ir/ar jų išgaubtų korpusų kolizijos nustatymo).

Tai, pagamina tarpinį objektų, artimų vienas kitam, porų sąrašą. Kiekvienam jų atliekamas tikslus kolizijos nustatymas. Taip, pagaliau, pagaminamas susiduriančių objektų sąrašas.

Pagaliau, kiekvienai susiduriančiai porai atliekamas vienas ar daugiau pakartojimų.

Kolizijos susidomėjimo matrica. „VR“ sistemoje kolizijos nustatymo modulis naudojamas skirtingų aukšto lygio modulių (kaip interakcijos tvarkytojas, atvirkštinė kinematika, standaus kūno dinamika, ir t. t.). Kiekvienas modulis yra susidomėjęs skirtingomis kolizijomis. Papildomai, ten dažniausiai yra daug objektų porų, kuriomis nėra susidomėjęs joks modulis. Galiausiai, kartais modulis tiesiog „žino“, kad tam tikros objektų poros negali išvis susidurti ar, kad objektų pora susidurs visą laiką.

Kita vertus, kolizijos nustatymo moduliui reikia išsaugoti tam tikrus duomenis su kiekviena objektų pora, kurių koliziją reikią patikrinti (pakartojimus, informaciją laikomą atmintyje paskutinio kolizijos patikrinimo metu, laiko ženklus, ir t. t.).

Duomenų struktūra dažniausiai yra diagonalinė matrica, kitos duomenų struktūros gali būti maišos lentelės (angl. *hash table*), reta matrica ar leksikografiškai sutvarkyti sąrašai.

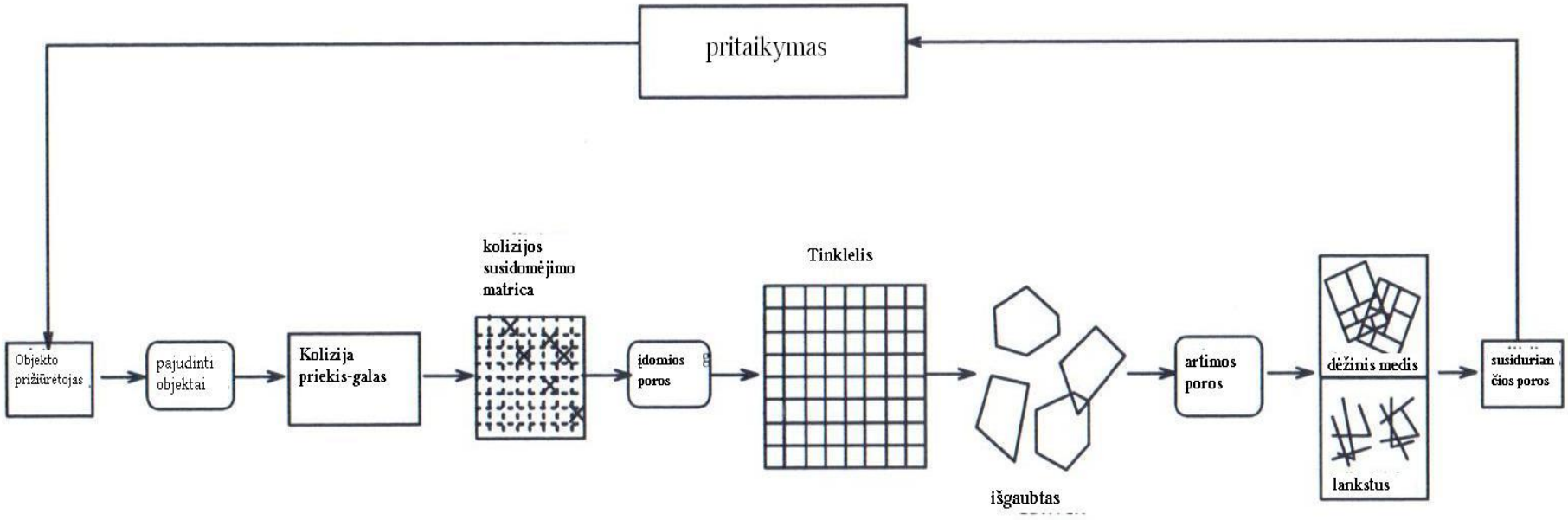
Tinklelio metodas

Pirmojo metodo atveju perkėlimo nustatymui naudojamas paprastas, pagal ašį sulygiuotas tinklelis. Judančio objekto centras perkeliamas į visų elementų centrą. Kiekvienu atveju objektas juda link fiksuoto objekto tol, kol pasiekiamas reikiamas atstumas arba įsiskverbimo gylis. Tada, konfigūracija išsaugoma. Deja, iš anksto neįmanoma žinoti šiuo metodu nustatomų konfigūracijų skaičiaus.

2.4 Eksperimentiniai išvesties duomenys

Be šių pagrindinių metodų, reikalingų pasirinktiems algoritmams, buvo suprojektuoti išvesties duomenys, kurių pagalba, tyrimų eksperimentinėje dalyje, bus analizuojami šių algoritmų veikimo našumai. Išvesties duomenys:

- Laikas – apskaičiuojamas per kiek laiko yra įvykdomas susidūrimų paieškos testas vykdant konkretų algoritmą. Laikas skaičiuojamas milisekundėmis (ms).
- Poligonų skaičius – šio kintamojo pagalba galime keisti objekto poligonų skaičių ir taip stebėti kaip keičiasi laikas, per kurį įvyksta kolizija.
- Atstumas – šio kintamojo pagalba galime nustatyti, kokiu atstumu turi būti objektai vienas nuo kito, kad skaitytumėm jog jie jau susidūrė vienas su kitu.



2.1 pav. Darbo eiga, nustatant kolizijas

3 Tyrimo eksperimentinė dalis

3.1 Atliekamo tyrimo metodologija

Šiame darbe yra atliekami susidūrimų paieškos algoritmų našumo tyrimai. Algoritmų našumas yra matuojamas nustatant jų pilną vykdymo laiką. Visi šiame darbe pateikti laiko matavimai yra išskaičiuojami kaip trijų to paties matavimo paleidimų vidurkis. Visų matavimų metu yra išlaikoma ta pati matavimui naudojamos aparatūrinės bei programinės įrangos konfigūracija. Matavimai yra tiesiogiai priklausomi nuo matavimo įrangos, todėl jų atlikimas kitoje aplinkoje gali pateikti kitokius rezultatus.

3.2 Pasirinktų tyrimų paskirtis

Šiame darbe vykdomų tyrimų sąrašas:

1. Algoritmų laiko priklausomybė nuo poligonų skaičiaus.
2. Laiko priklausomybė nuo poligonų skaičiaus, keičiant parametrus.
3. Laiko priklausomybė, keičiant atstumą kuomet traktuojama, jog objektai susidūrė.

Šiame darbe atliekamų tyrimų aprašymai:

1. Nustatyti susidūrimų paieškos algoritmo *PQP, Solid, V-Collade, Opcode, BoxTree ir Dop-Tree* vykdymo laiko bei poligonų skaičiaus priklausomybę nuo susidūrimo atstumo.

2. Nustatyti visų tiriamų algoritmų veikimo laiko bei poligonų skaičiaus priklausomybę keičiant jų parametrus.

3.3 Tyrimui naudojamos įrangos bei parametrų aprašas

Tyrimui atlikti buvo pasirinkta aparatūrinė bei programinė įranga, aprašyta žemiau pateikiamoje lentelėje (žr. 3.1 pav.).

Procesorius	Intel Core2 6700 @ 2.66 GHz
Atmintis (RAM)	2 GB
Sistemos tipas	64-bitų operacinė sistema
Programa/Biblioteka/Įrankis	Microsoft Visual Studio, OpenSG

3.1 pav. Naudojamo kompiuterio techninių duomenų paveiksliukas

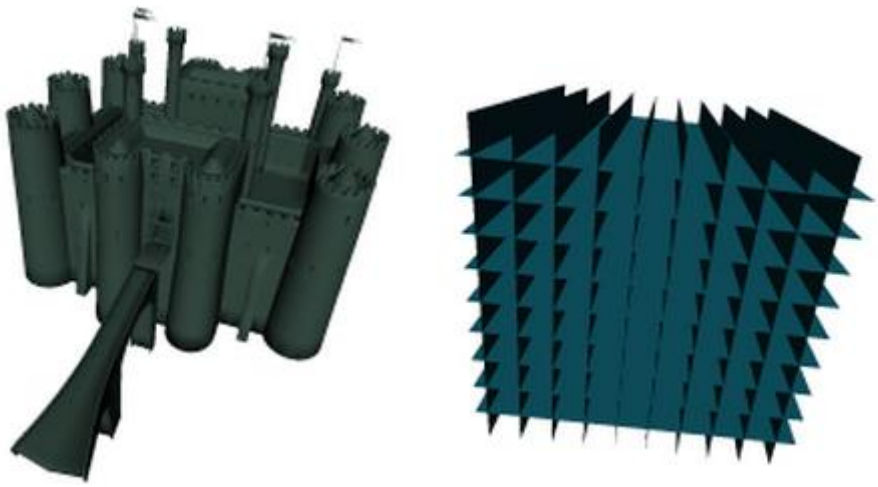
Visų atliktų tyrimų rezultatai yra tiesiogiai priklausomi nuo aukščiau aprašytos įrangos. Tai reiškia, jog kitos konfigūracijos atveju yra galimybė gauti skirtingus rezultatus.

3.4 Tyrimo rezultatai

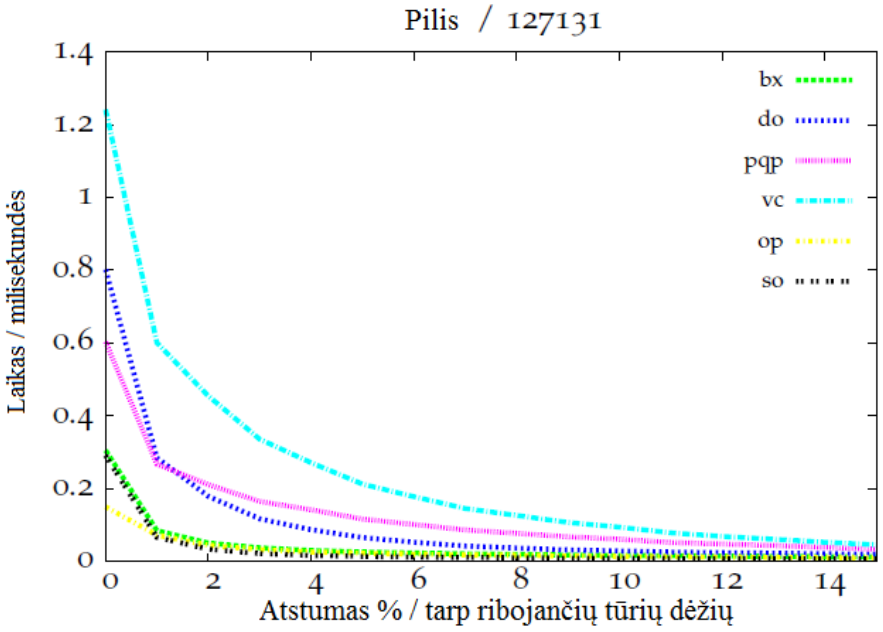
Pagrindinis skiriamasis susidūrimų aptikimo metodų veiksnys yra kontaktinės informacijos, kurias jie pateikia, pobūdis. Kaip buvo teigiama analizės skyriuje, ši informacija gali būti minimalus atstumas tarp objektų poros, įsiskverbimo gylis arba paprasčiausias atsakymas ar objektai susiduria, ar ne. Tiesą sakant, daugelyje laisvai prieinamų bibliotekų pateikiama tik pastaroji informacija. Dažniausiai „Būlio“ algoritmai sustabdo paiešką, kai suranda pirmąją susikertančių daugiakampių porą. Šįkart būtent tai ir bus tiriama.

Scenarijus: dauguma „Būlio“ susidūrimų aptikimo metodų yra paremta susiduriančių tūrių hierarchija. Jeigu dviejų objektų tūriai nesusikerta, nėra susidūrimo, juos galima labai greitai atmesti. Jeigu du objektai iš dalies sutampa, jų rekursyvi paieška patikros metu turi greitai pakrypti link susiduriančių daugiakampių porų. Blogiausias atvejis šiems algoritmams yra konfigūracija, kurios atveju sutampa daug BV, tačiau daugiakampiai nesusikerta.

Šiame scenarijuje norima sukonstruoti konfigūraciją, kurioje objektai yra labai arti vienas kito, bet nesutampa. Buvo panaudotas „Apollo 13“ erdvėlaivio modelis, kadangi jis yra išgaubtas, tačiau tuo pačiu turi daug mažų paviršiaus detalių. Norint patikrinti susidūrimų aptikimo bibliotekų našumą išlenktų objektų atveju, buvo pasirinkta šviestuvo, kėdės ir „ATST“ ėjiko modeliai. Pilies modelį sudaro labai maži ir labai dideli trikampiai. Grotelių modelis buvo naudotas nevienodų geometrijų tikrinimui.

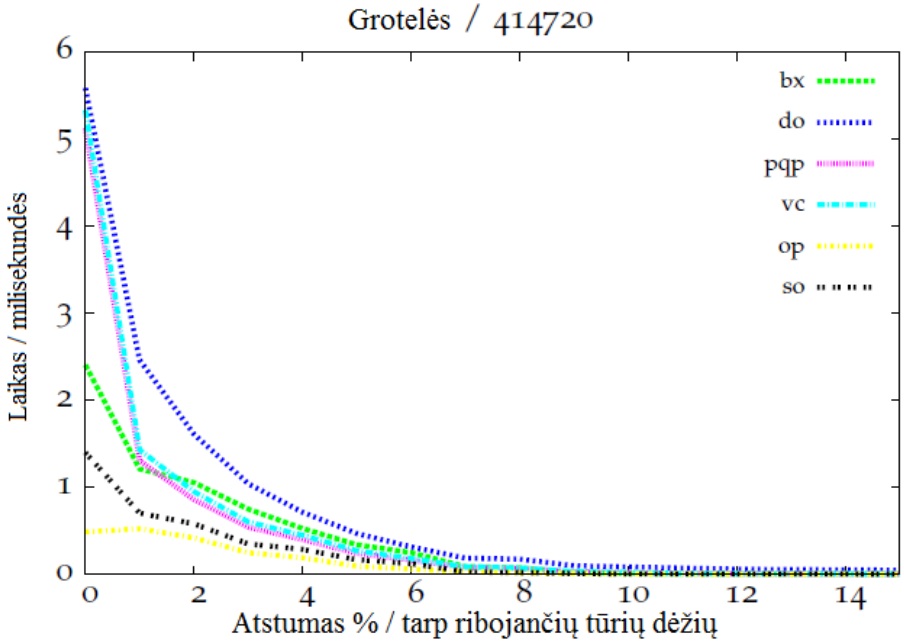


3.2 pav. [14] Pilies ir grotelių paveiksliukai, kurie buvo naudojami tyrime.



3.3 pav. Algoritmų rezultatai, naudojant pilies paveiksliukus.

Gauti algoritmų rezultatai naudojant pilies paveiksliukus su aukšta rezoliucija, kur poligonų skaičius siekė 127 131. X ašis žymi santykinį atstumą tarp objektų, kur 1.0 yra objekto dydis. Atstumas 0.0 reiškia, kad objektai yra labai arti vienas kito (beveik liečiasi), tačiau nesikerta. Grafike matomų santrumpų reikšmės yra šios: bx=BoxTree, do=Dop-Tree, pqp=PQP, vc=V-Collide, op=Opcode, so=SOLID. Šiuo atvėju „AABB“ pagrįsti algoritmai yra pranašesni.

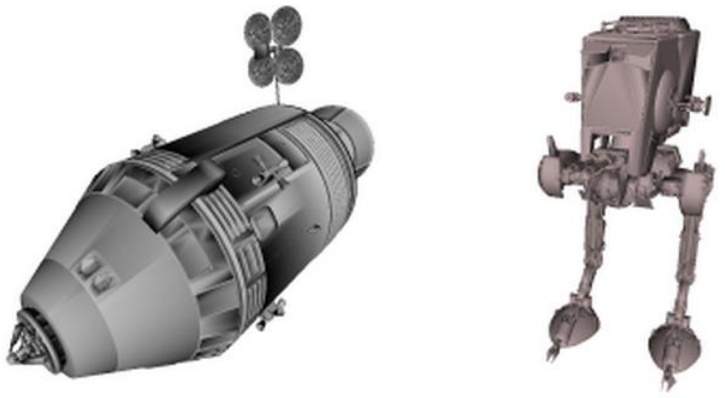


3.4 pav. Algortmų rezultatai, naudojant grotelių paveiksliukus.

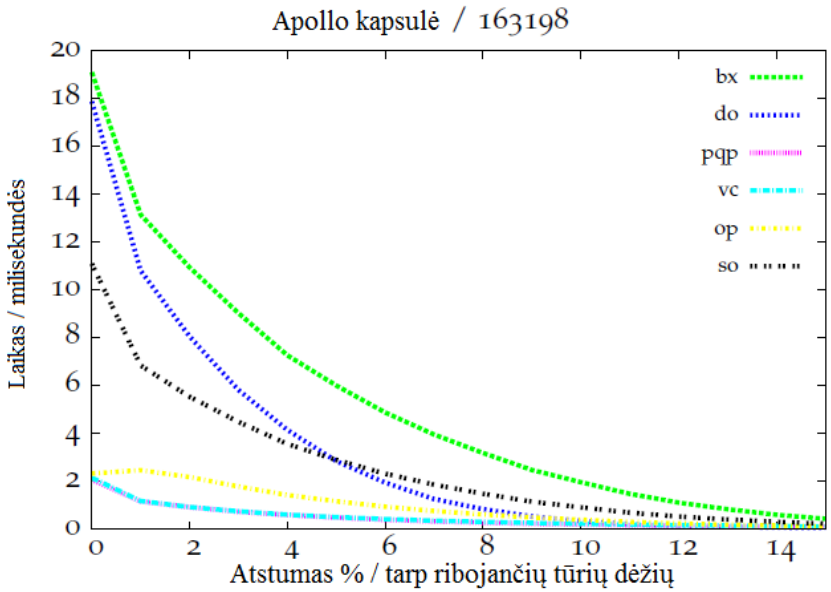
Gauti algoritmų rezultatai, naudojant grotelių paveiksliukus su aukšta rezoliucija, kur poligonų skaičius siekė 414 720. Nors ir algoritmų rezultatai skiriasi nuo pilies scenarijaus, tačiau galime teigti, kad principas yra vienodas. Kaip ir praeitame paveiksliuke, „AABB“ pagrįsti algoritmai yra pranašesni.

Pirmoji pagrįsta matavimų išvada yra faktas, jog algoritmai, naudojantys tos pačios rūšies *BVH*, elgiasi labai panašiai. Antroji išvada teigia, kad visi algoritmai skirtinguose scenarijuose pasižymi savo privalumais ir trūkumais. Pavyzdžiui, „AABB“ paremti algoritmai, kaip „SOLID“, „Opcode“ ir „BoxTree“ puikiai tinka reguliariems tinkleliams, pvz., šviestuvui bei itin skirtingų trikampių dydžių tinkleliams, pvz., pilims (žr. 3.2 ir 3.3 pav.). Priežastis yra faktas, jog šiais testų atvejais „AABB“ puikiai dera su objektais, todėl algoritmai turi naudos iš savo greitesnio susidūrimų patikrinimo algoritmo.

Naudojant įgaubtus ir išsklaidytus objektus, pvz., šviestuvas ar „ATST“ bei objektus su gausybe mažų detalių, pvz., „Apollo“ erdvėlaivis, situacija kardinaliai pasikeitė ir „OBB“ paremti algoritmai pasirodė daug geriau nei „AABB“ paremtos bibliotekos (žr. 3.5 ir 3.6 pav.). Tai lemia faktas, jog šių objektų rūšių atveju *BVH* veikia geriau nei spartus BV testas. Ypatingą vaidmenį atlieka „DOP-Tree“, kurio atveju derinami greiti „AABB“ paremtų algoritmų BV testai ir OBB paremtų bibliotekų BV.

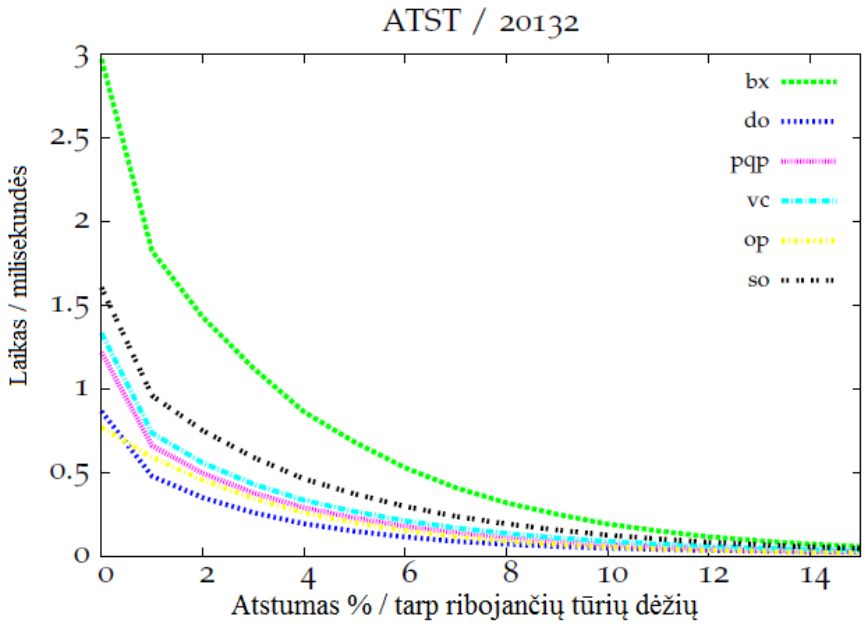


3.5 pav. [14] „Apollo 13“ ir „ATST walker“ paveiksliukai, kurie buvo naudoti tyrime.



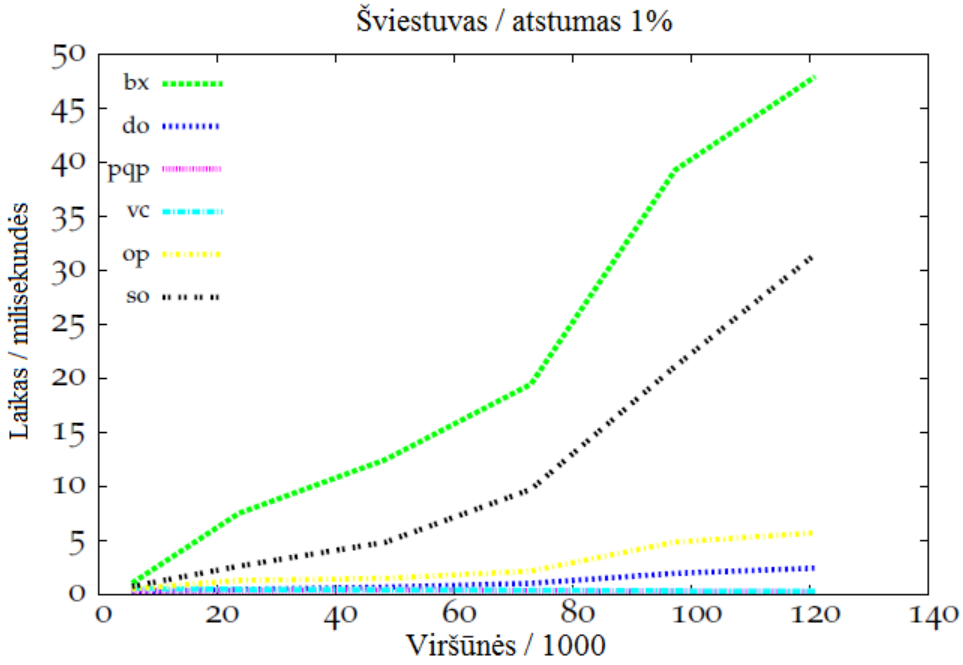
3.6 pav. Algoritmų rezultatai, naudojant „Apollo 13“ kapsulės paveiksliukus.

Gauti algoritmų rezultatai, naudojant „Apollo 13“ kapsulės paveiksliukus su aukšta rezoliucija, kur poligonų skaičius siekė 163 198. Šio atvejo rezultatai skiriasi nuo praeitų bandymų, šiuo atveju „OBB“ pagrįsti algoritmai buvo žymiai greitesni už „AABB“ pagrįstus algoritmus.



3.7 pav. Algoritmų rezultatai, naudojant „ATST“ ėjiko paveiksliukus.

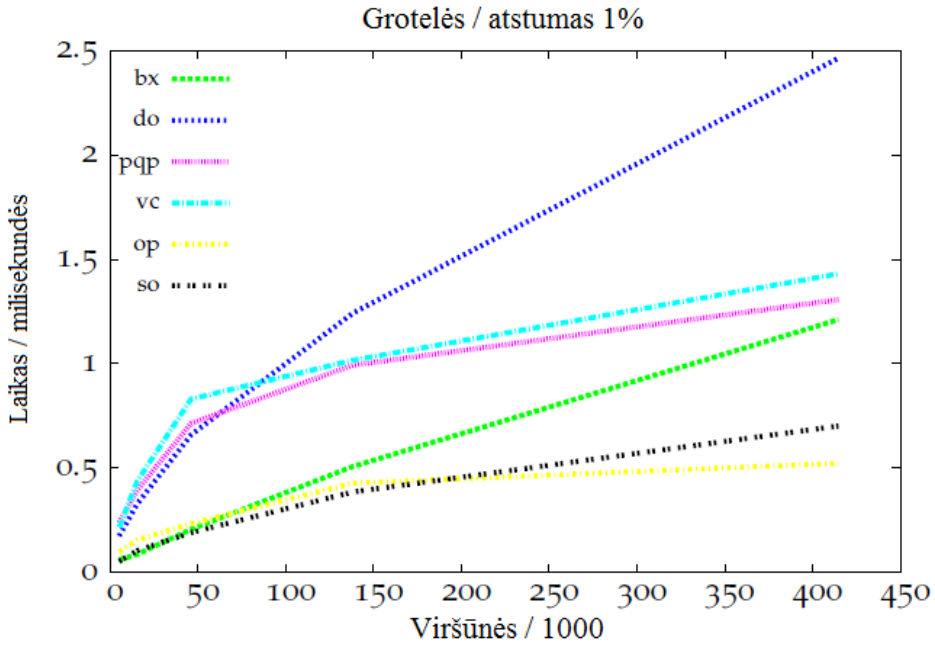
Gauti algoritmų rezultatai naudojant „ATST“ ėjiko paveiksliukus su aukšta rezoliucija, kur poligonų skaičius siekė 20132. Kaip ir „Apollo 13“ kapsulė atveju „OBB“ pagrįsti algoritmai buvo greitesni už „AABB“ pagrįstus algoritmus.



3.8 pav. Kolizijos aptikimo laiko priklausomybė nuo paveikslėlio sudėtingumo su šviestuvo paveikslėliais.

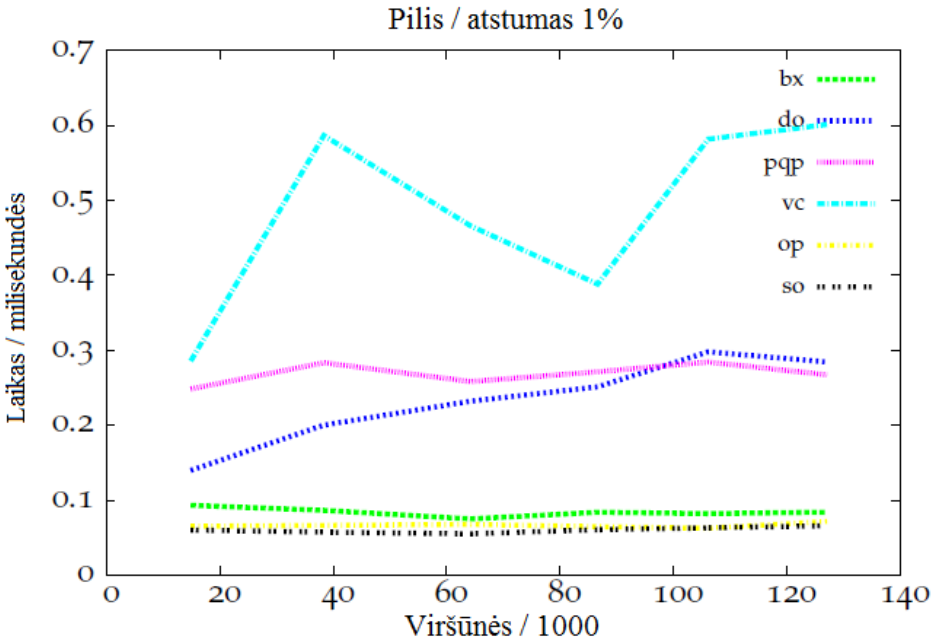
Gauti algoritmų rezultatai, naudojant šviestuvo paveiksliukus, kur atstumas tarp objektų yra 1% lyginant su objekto dydžiu. Kur x ašis žymi poligonų skaičių padalintą iš 1000. Šios scenos susidūrimo aptikimo laikas naudojant „AABB“ pagrįstus algoritmus, galima sakyti, jog didėja tiesiškai, tačiau geresnius rezultatus parodė „OBB“ pagrįsti algoritmai.

3.9 pav. [14] Šviestuvo ir kėdės paveiksliukai, kurie buvo naudoti tyrime.



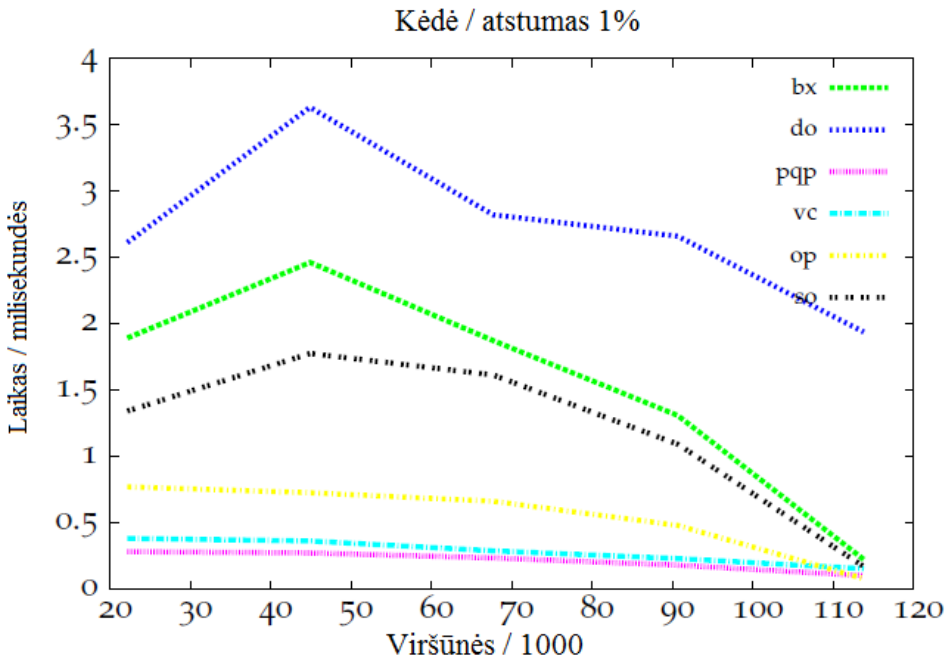
3.10 pav. Kolizijos aptikimo laiko priklausomybė nuo paveikslėlio sudėtingumo su grotelių paveikslėliais.

Gauti algoritmų rezultatai, naudojant grotelių paveiksliukus, kur atstumas tarp objektų yra 1% lyginant su objekto dydžiu. Kur x ašis žymi poligonų skaičių padalintą iš 1000. Priešingai negu prieš tai buvusiame paveikslėlyje (kolizijos su šviestuvu, žr. 5.7 pav.) veikimo laikas padidėjo O (n log n) visų algoritmų požiūriu.



3.11 pav. Kolizijos aptikimo laiko priklausomybė nuo paveikslėlio sudėtingumo su pilies paveikslėliais.

Kolizijos aptikimo laiko priklausomybė nuo paveikslėlio sudėtingumo su pilies paveikslėliais scenoje. Kur atstumas tarp objektų yra 1% lyginant su objekto dydžiu**.** Galime matyti, kad kolizijos aptikimo laikas yra priklausomas nuo paveiksliuko sudėtingumo. Geresnius rezultatus parodė „AABB“ pagrįsti algoritmai.



3.12 pav. Kolizijos aptikimo laiko priklausomybė nuo paveikslėlio sudėtingumo su kėdės paveikslėliais.

Kolizijos aptikimo laiko priklausomybė nuo paveikslėlio sudėtingumo su kėdės paveikslėliais scenoje. Taipogi, kur atstumas tarp objektų yra 1% lyginant su objekto dydžiu. Priešingai negu pilies scenarijuje (žr. 3.10 pav.), kuo labiau didinamas objekto sudėtingumas, tuo kolizijos aptikimas atliekamas greičiau. Galima teigti, kad pranašesni buvo „OBB“ pagrįsti algoritmai.

Kitas įdomus aspektas, kuris buvo patikrintas, yra priklausomybė nuo objektų sudėtingumo. Dėl šios priežasties visi modeliai buvo testuojami skirtingose rezoliucijose. Stebina rezultatas, jog nebuvo jokios priklausomybės nuo testuojamų algoritmų sudėtingumo. Pavyzdžiui, šviestuvo atveju laiko reikšmė didėjo beveik tiesiškai su poligonų skaičiumi „AABB“ paremtų bibliotekų atveju, tuo tarpu „OBB“ paremtų algoritmų atveju ji išliko beveik pastovi (žr. 3.7 pav.). Grotelių scenarijaus atveju didėjimas buvo lygus O (n log n) visiems algoritmams (žr. 3.9 pav.). Pilies atveju susidūrimo aptikimo trukmė nepriklauso nuo sudėtingumo (žr. 3.10 pav.), o kėdės atveju susidūrimo aptikimo trukmė mažėjo, didėjant objektų sudėtingumui, visiems algoritmams (žr. 3.11 pav.).

3.5 Išvados

Ištyrus susidūrimo aptikimo metodus su skirtingų detalių paveiksliukais buvo pastebėta, kad kiekvienu atveju algoritmai elgėsi skirtingai. Buvo atlikti du skirtingi testai, kurių rezultatams turėjo įtakos pasirinktas paveiksliukas. Antrojoje išvadoje aprašoma laiko priklausomybė nuo poligonų skaičius ir atstumo tarp objektų rezultatai. Trečiojoje išvadoje aprašoma laiko priklausomybė nuo objektų sudėtingumo ir atstumo tarp jų.

Naudojant pilies paveiksliuką, kuris susideda iš didelių ir mažų trikampių geresnius rezultatus parodė „AABB“ pagrįsti algoritmai ir geriausiai pasirodė „Opcode“ algoritmas. Naudojant grotelių paveiksliukus, taipogi geriau pasirodė „AABB“ pagrįsti algoritmai ir lygiai taip pat kaip pilies paveiksliukų atveju „Opcode“ algoritmas buvo greičiausias. „Apollo 13” erdvėlaivio kapsulės modelis buvo naudojamas išgaubtų objektų tikrinimui ir šiuo atveju pranašesni buvo „OBB“ pagrįsti algoritmai, „PQP“ ir „V-Collide“ pasirodė vienodai ir buvo geriausiai tinkami šiam scenarijui. „ATST“ ėjiko atveju, taipogi „OBB“ pagrįsti algoritmai pasirodė geriau ir čia dominavo „Dop-Tree“ algoritmas.

Išlenktų objektų tikrinimui buvo naudojami šviestuvo, kėdės ir „ATST“ ėjiko modeliai. Naudojant šviestuvo modelio paveiksliukus nors ir „AABB“ pagrįstų algoritmų didinant poligonų skaičių laikas didėja beveik tiesiškai, tačiau „OBB“ pagrįsti algoritmai pasirodė žymiai geriau, kur geriausius ir vienodus rezultatus parodė „PQP“ ir „V-Collide“ algoritmai. Naudojant grotelių modelio paveiksliukus rezultatai buvo atvirkštiniai lyginant su šviestuvo modelio paveiksliukais, čia geriau pasirodė „AABB“ pagrįsti algoritmai ir greičiausias buvo „Opcode“ algoritmas. Naudojant pilies modelio paveiksliukus, vėlgi, greitesni buvo „AABB“ pagrįsti algoritmai, kur savo sparta išsiskyrė „SOLID“ algoritmas. Naudojant kėdės modelio paveiksliukus, galima teigti, kad pranašesni buvo „OBB“ pagrįsti algoritmai, iš kurių greičiausias buvo „PQP“, tačiau šiuo atveju susidūrimo aptikimo trukmė mažėjo didėjant objektų sudėtingumui, visiems algoritmams.

4 Išvados

1. Kolizijų nustatymo tarp objektų problema buvo ištirta ir dažnai motyvuota patobulinto veikimo poreikiu. Nors, kai kurie mokslininkai daugiausia dėmesio skyrė skaičiavimo efektyvumui, kiti sutelkė dėmesį į rezultatų tikslumą. Kai kuriais atvejais šis tyrimas vedė prie atviro kodo programinės įrangos bibliotekų ir įrankių išleidimo, įgalinančio kitus tyrėjus tobulinti šią sritį.
2. Kolizijos nustatymo algoritmų lyginamosios analizės plėtojimas yra ypatingai reikalaujantis pastangų dėl įvairių faktorių, kurie gali žymiai paveikti algoritmų veikimą. Įtrauktų objektų tipai ir jų santykinės transformacijos, ir dydžiai gali turėti įtakos užklausų skaičiavimo laikui. Todėl, taikymo tipas yra vienas iš lemiančių veiksnių pasirenkant tinkamą kolizijos aptikimo strategiją. Deja, ne tik teorijoje, bet ir praktikoje susidūrimų aptikimo algoritmų palyginimas yra itin sudėtinga užduotis, kadangi jie yra jautrūs konkretiems scenarijams, pvz., objekto formai ir dydžiui, padėčiai, krypčiai vienas kito atžvilgiu ir t.t. Be to, skirtingi susidūrimų aptikimo metodai pateikia skirtingą kontaktinę informaciją, pvz., minimalų atstumą, įsiskverbimo gylį.
3. Realizavus tradicinius susidūrimų paieškos *V-Collide, PQP, SOLID, Opcode, Box-Tree ir Dop-Tree* algoritmus, ir viską apibendrinant, galima teigti, kad nėra jokio vieno įrankio, kuris tiktų kiekvienam tikslui. Kiekvienas algoritmas atitinkamuose scenarijuose pasižymi tam tikrais privalumais. Dėl šios priežasties vartotojas, rinkdamasis susidūrimų aptikimo algoritmą, turėtų kiekvieną scenarijų kruopščiai patikrinti. Manoma, kad puikus kompromisas yra „Dop-Tree“, kadangi jo atveju derinami glaudūs BV ir greiti BV testai. Be to, kai kuriais atvejais naudinga padidinti modelio sudėtingumą, norint sumažinti susidūrimų aptikimo trukmę, tačiau tai ne visuomet tinkama. Vis dėlto, beveik visos bibliotekos veikia pakankamai greitai, atlikdamos realaus laiko susidūrimų patikrinimus net ir itin sudėtingiems objektams.

5 Literatūros sąrašas

[1] Paul S. A. Reitsma and Carol O'Sullivan. Effect of scenario on perceptual sensitivity to errors in animation. New York, NY, USA, 2008. ACM.

[2] Susan Fisher and Ming Lin. Fast penetration depth estimation for elastic bodies using deformed distance fields. 2001. IROS. Puslapiai 330-336.

[3] NVIDIA. Nvidia physx. [Tinkle] Prieiga internete : http://www.nvidia.com/object/nvidia_physx.html [žiūrėta 2015-01-22].

[4] Erwin Coumans. Bullet physics library. [Tinkle] Prieiga internete : http://bulletphysics.com [žiūrėta 2015-01-22].

[5] Russel Smith. Open dynamics engine. [Tinkle] Prieiga internete : http://www.ode.org, 2012, [žiūrėta 2015-01-22].

[6] Klein, J., & Zachmann, [Tinkle] Prieiga internete : http://www.gabrielzachmann.org/ [žiūrėta 2015-03-15].

[7] Lin, M. C., & Canny, J. F. (1991). A fast algorithm for incremental distance calculation. In In IEEE International Conference on Robotics and Automation, puslapiai 1008–1014.

[8] Gabriel Zachmann & Elmar Langetepe Geometric Data Structures for Computer Graphics, University of Bonn, Germany, ACM Transactions of Graphics, 2003

[9] Collision detection pipeline [Tinkle] Prieiga internete : http://cgvr.cs.uni-bremen.de/index.shtml [žiūrėta 2015-01-22].

[10] Gabriel Zachmann, Matthias Teschner, Stefan Kimmerle, Bruno Heidelberger, Laks Raghupathi, and rnulph Fuhrmann. Real-time collision detection for dynamic virtual environments. Virtual Reality, 2005. Proceedings. VR 2005. IEEE

[11] Chazelle Polyhedron. [Tinkle] Prieiga internete : https://www.ics.uci.edu/~eppstein/junkyard/untetra/ [žiūrėta 2015-03-15]

[12] Opcode [Tinkle] Prieiga internete : http://www.codercorner.com/Opcode.htm [žiūrėta 2015-01-22].

[13] Kolizijos aptikimo algoritmai [Tinkle] Prieiga internete : http://gamma.cs.unc.edu [žiūrėta 2015-01-22].

[14] Geometriniai paveiksliukai [Tinkle] Prieiga internete: http://sgi.felk.cvut.cz/BES/scenes/ [žiūrėta 2014-10-22].

[15] Trimačių geometrinių modulių klasifikacija [Tinkle] Prieiga internetą: http://www.elibrary.lt/resursai/Konferencijos/KTU_01/IT_2003/Sekcija02.pdf [žiūrėta 2014-10-22].

[16] Jan Klein and Gabriel Zachmann. Interpolation search for point cloud intersection. Puslapiai 163–170, University of West Bohemia, Plzen, Czech Republic.