VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Spartus kintamo detalumo kraštovaizdžių atvaizdavimas: geomipmap algoritmo optimizacija

Fast Terrain Rendering: Geomipmap Optimization

Magistro baigiamasis darbas

Atliko:	Darius Jankauskas	(parašas)
Darbo vadovas:	Doc. dr. Antanas Lenkevičius	(parašas)
Recenzentas:	Doc. dr. Rimvydas Krasauskas	(parašas)

Vilnius – 2009

Santrauka

Dažnai kompiuterinės grafikos taikymuose labiausiai visos scenos atvaizdavimo greitį įtakoja kraštovaizdžių atvaizdavimas. Darbe nagrinėtas vienas iš dviejų greičiausių šiuolaikinių kintamo detalumo kraštovaizdžių atvaizdavimo algoritmų – geomipmap. Šis algoritmas pasirinktas dėl platesnio pritaikomumo, didesnio algoritmo lankstumo ir šiai dienai dažnai didesnio greičio lyginant su jo varžovu geoclipmap. Algoritmo idėja – panaikinti kas antrą viršūnę iš vaizduojamųjų sąrašo, pereinant prie žemesnio detalumo lygio, taip sudarant galimybę laikyti visą geometriją keliuose nekintančiuose viršūnių ir indeksų buferiuose.

Darbe pasiūlyta algoritmo optimizacija išnaudoja iš aukščių žemėlapio generuojamų kraštovaizdžių apatinę ribą. Pagrindinė idėja – nevaizduoti visų šioje riboje esančių trikampių, vėliau visus atsiradusius tarpus uždengti naudojant vieną plokštumą, ištemptą per visą kraštovaizdį. Nors optimizacijos efektyvumas priklauso nuo kraštovaizdžio formos ir negalima vienareikšmiškai įvertinti atvaizdavimo pagreitėjimo bendru atveju, nagrinėtose scenose užfiksuotas didelis našumo padidėjimas, aiškiai vertingas, net jei pasireikštų tik labai retais atvejais. Darbe taip pat parodoma, kad našumo padidėjimas iš tiesų galimas daugelyje scenų.

Summary

Often the bottleneck in the real-time 3D rendering applications is the rendering of a terrain. Geomipmap algorithm is analyzed here as it is one of the best terrain rendering algorithms known. It is much more versatile and flexible in comparative to geoclipmap and still often outperforms its main rival. The idea is to eliminate every other vertex in order to get a lower level of detail, thus having all the geometry in several immutable vertex and index buffers.

An optimization is proposed that utilizes the lower bound of terrains generated from a heightmap. The idea is not to render the triangles that are exactly on the lower bound of a terrain and to cover all the eliminated triangles with a single plane. Even though the effectiveness of the optimization is highly dependent on the exact form of the terrain, the high performance gain observed in the analyzed scenes is valuable even if it would only occur in very rare cases. It is shown that the performance gain can actually be observed in quite a variety of scenes.

Įvadas 5
1. Kompiuterinės grafikos spartinimo metodų apžvalga 6
1.1. Atkirtimas 6
1.2. Išrinkimas 9
1.3. Detalumo adaptacija 13
1.4. Scenos grafas 15
1.5. Bendravimo su vaizdo plokšte optimizacijos 17
1.6. Metodų apžvalgos išvados 19
2. Principiniai sprendimai 20
2.1. Kintamo detalumo kraštovaizdžiai 20
2.2. Grafinės bibliotekos ir grafiniai varikliukai 22
3. Darbo eiga 24
3.1. Pirmas bandymas 24
3.2. Geometrijos suspaudimo (geomipmap) algoritmas 26
3.3. OGRE geomipmap realizacija 29
3.4. Pirmo bandymo analizė 30
3.5. Apatinės ribos optimizacijos 32
3.5.1. Mažiau trikampių 32
3.5.2. Mažiau sričių 32
3.5.3. Tarpų užtaisymas 33
3.5.4. Pritaikymas 34
3.6. Matavimai 35
Literatūros šaltinių sąrašas 38
Sąvokų apibrėžimai 40
Priedai 41
1. Parašytos programos jų išeities tekstai ir šio dokumento el. variantas 41

Įvadas

Trimatė grafika yra labai imli kompiuteriniams resursams, kad ir kaip sparčiai tobulėja kompiuteriai, jie nespėja su vis naujomis galimybėmis trimatėje grafikoje. Labai dažnai, kuriant taikomąją programą, tik artėjant į pabaigą jos kūrimui, pastebima kad programa nespėja atvaizduoti scenos žmogaus akims (taip pat ir daugumai televizorių) tinkamu dažniu – 30 kadrų per sekundę. Norint pasiekti geros kokybės vaizdą ir interaktyvumą, dažnai reikia pasiekti bent 60 kadrų per sekundę.

Darbo tikslas yra spartus realaus laiko trimatės scenos atvaizdavimas. Darbe apžvelgiami spartinimo metodai:

- Matomumo nustatymas (angl. visibility test):
  - Atkirtimas (angl. clipping),
  - Išrinkimas (angl. culling),
  - Erdvės padalijimas;
- Detalumo adaptacija (angl. Level of Detail);
- Scenos grafas;
- Bendravimo su vaizdo plokšte optimizacijos:
  - Vaizdavimo sąrašai (angl. display lists),
  - Viršūnių ir indeksų buferiai.

Didžiąją dalį trimačių scenų labai dažnai sudaro kraštovaizdžiai (skrydžių stimuliatoriuose, kompiuteriniuose žaidimuose, geografinėse sistemose), tokiose scenose būtent kraštovaizdžio atvaizdavimo greitis labiausiai įtakoja visos scenos atvaizdavimo greitį, todėl toliau darbe analizuojamas ir optimizuojamas kintamo detalumo kraštovaizdis (konkrečiai geomipmap algoritmas), metodas priskiriamas tolydžios detalumo adaptacijos grupei, bet iš tiesų jo supratimui ir realizavimui reikalingi visi aukščiau paminėti metodai.

1. Kompiuterinės grafikos spartinimo metodų apžvalga

1.1. Atkirtimas

Atkirtimas – tai labai svarbi kompiuterinės grafikos vizualizacijos dalis. Atkirtimu vadinamas vaizdo dalies, nepatenkančios į atkirtimo langą, pašalinimas. Jis dažniausiai naudojamas prieš paduodant dvimatę scenos projekciją rasterizavimui, kartais prieš projektuojant trimatę sceną į dvimatę.

Pagrindiniai atkirtimo tikslai:

- Išvengti dalybos iš nulio ir persipildymo.
- Neeikvoti laiko objektams už vaizdo ribų.
- Išvengti neapibrėžtumų piešiant už akies matomumo ribų.

Kiekvienas iš šių tikslų yra esminis realaus laiko kompiuterinėje grafikoje. Dalyba iš nulio ir persipildymas yra visiškai netoleruojamos klaidos.

Kiekvieno objekto projektavimas ir transformavimas į rastrinį vaizdą naudoja kompiuterio resursus nepriklausomai nuo to ar mes tą objektą matysime ekrane. Kompiuterinė grafika yra imli kompiuterio resursams, todėl negalime vizualizuoti pavyzdžiui viso miesto, kai šiuo metu matomas tik vienas kambarys.

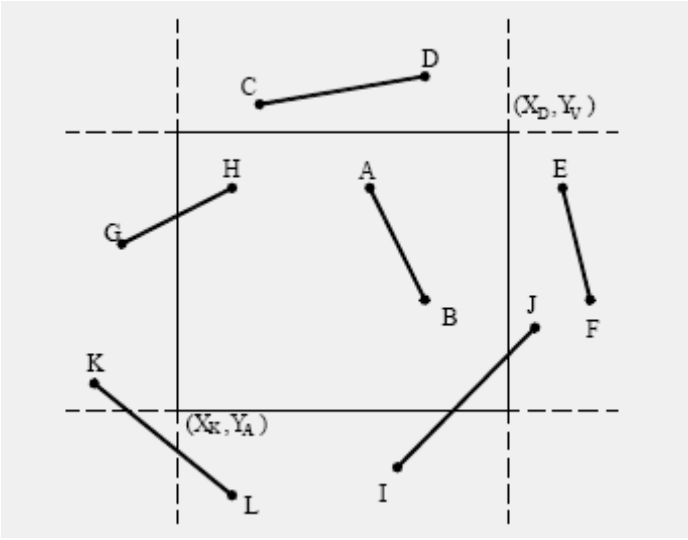
Rastrinio vaizdo kūrimo įrenginiai nepritaikyti dirbti su koordinatėmis nepatenkančiomis į vaizduojamąjį langą (pvz. monitoriaus ekraną). Pavyzdžiui objektas kurio koordinatės šiek tiek užeina už ekrano dešiniosios ribos, daugelyje vaizdo plokščių būtų atvaizduojamas kairėje ekrano dalyje (kai iš tiesų turėtų būti nematomas), o blogiausiu atveju sugadintų visą vaizdą.

Atkirtimo problema sprendžiama atskirai kiekvienam atkirtimo tipui:

- Taško atkirtimas;
- Atkarpos atkirtimas;
- Daugiakampio atkirtimas;
- Kreivės atkirtimas;
- Teksto atkirtimas.

Kiekviena iš šių problemų dar sprendžiama atskirai dvimatėje ir trimatėje erdvėse. Taško atkirtimas yra gana paprastas, taškas visada yra arba atkirtimo lango viduje (vaizduojamas) arba už jo ribų (nevaizduojamas). Atkarpa gali kirsti atkirtimo lango ribą, tokiu atveju reikia atkirsti dalį atkarpos (1 pav.). Kreivės dažniausiai aproksimuojamos atkarpomis, nors atsiranda taikymų, kai būtinas specifinis kreivių atkirtimas, pvz. Bezje splainų atkirtimas [EHS05]. Tekstas priklausomai nuo jo prigimties (vektorinis ar rastrinis) atkertamas daugiakampių arba taškų

atkirtimo metodais, arba paprasčiausiu metodu „viskas arba nieko“ – jei visas simbolis patenka į atkirtimo langą jis piešiamas, kitu atveju – ne.

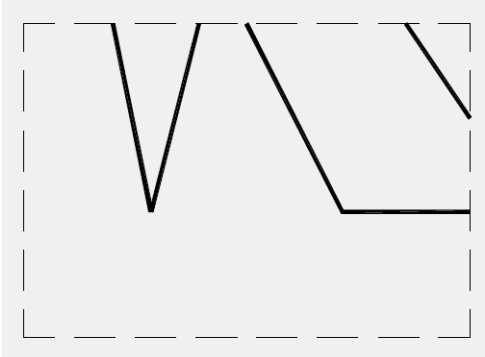


1 pav. Atkarpų išsidėstymas atkirtimo lango atžvilgiu [Len07]

Yra sukurta daugybė atkarpų atkirtimo algoritmų, kurių dauguma dabar benaudojami tik mokymo tikslams.

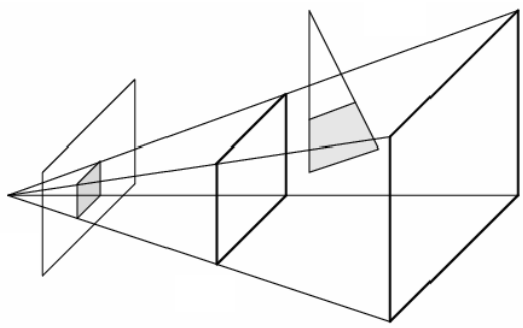
Dabartinėje trimatėje grafikoje, deja, retai naudojamos atkarpos, paprastai viskas vaizduojama daugiakampiais, dažniausiai – trikampiais. Bet kokio detalumo objektą visada galima sudėlioti vien iš trikampių, o jie yra patogiausia daugiakampio forma vaizdo plokštei.

Ilgą laiką daugiakampių atkirtimui buvo stengiamasi pritaikyti jau išnagrinėtus ir išbandytus atkarpų atkirtimo metodus, pvz. [LB83]. Jau 1974 metais buvo pastebėta, paprasčiausiai atkirtus kiekvieną daugiakampio atkarpą nepriklausomai nuo kitų, po to būna sudėtinga išsiaiškinti kur yra daugiakampio vidus, o kur išorė (2 pav.), arba ar prieš atkirtimą daugiakampio viduje buvo atkirtimo lango kampas (tokiu atveju reiktu sugeneruoti dvi papildomas atkarpas einančias į tą kampą), todėl efektyviau yra kurti specifinius daugiakampių atkirtimo algoritmus [SH74]. Naujesni iš tokių algoritmų būtų [Vat92], [Mai92], [GH98], [ZS02].



2 pav. Daugiakampis, atkirstas atkarpų atkirtimo metodu [Len07]

Kaip minėta ankščiau, yra ir trimatei erdvei skirtų atkirtimo algoritmų, pvz. [Puk77], [LB83], [Vat92]. Atkirtimas trimatėje erdvėje yra vienareikšmiškai sudėtingesnis. Dvimatėje erdvėje atkirtimo langas dažniausiai būna stačiakampis, retais atvejais – neiškilusis daugiakampis. Trimatėje erdvėje su perspektyva, atkirtimo langas yra nupjautinė piramidė, kurios smailasis galas sutampa su monitoriaus ekranu arba stebėtojo akimi, o bukasis su matomumo riba (3 pav.), taigi vietoj keturių ribojančių tiesių, kurių visos tarpusavį lygiagrečios arba statmenos, turime šešias plokštumas, iš kurių tik dvi lygiagrečios.



3 pav. Daugiakampių atkirtimas trimatėje erdvėje [Len07]

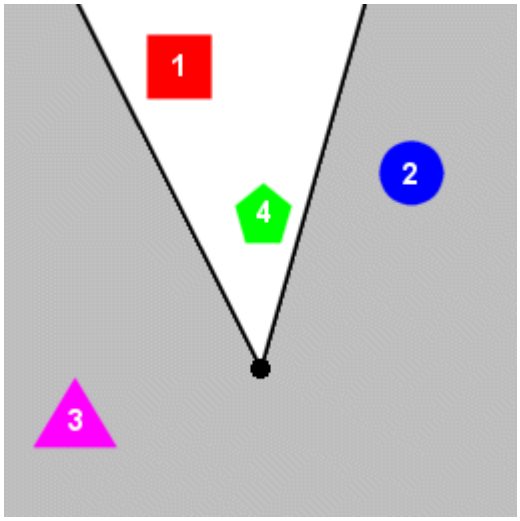
„Bendra ankstesnių algoritmų silpnybė yra būtinybė apskaičiuoti susikirtimo taškus, kurie net nėra rezultato dalis“ [NLN87]. Nors tai buvo pasakyta pristatant naują, geresnį atkirtimo algoritmą, dabar kai atkirtimo algoritmų efektyvumas jau pasiekė viršūnę, ir su daugybe nepublikuojamų optimizacijų jie aparatūriniame lygyje realizuojami visose personalinių kompiuterių vaizdo plokštėse ir apskritai beveik visų kompiuterių vaizdo plokštėse, efektyvaus kompiuterinės grafikos greitinimo verta ieškoti kitur, pvz. atmesti didžiąją dalį daugiakampių dar prieš atkirtimą.

1.2. Išrinkimas

Idėja paprasta – atmesti objektus, kurių tikrai nematysime (4 pav.), dar prieš projektuojant juos į dvimatę erdvę, geriausia – prieš bet kokius skaičiavimus, nereikalingus nematomiems objektams: animacija, apšvietimas, šešėliavimas, ir t.t. Paprastai programose trimatės scenos visada būna suskirstytos į objektus, kad būtų patogiau programuoti. Belieka patikrinti ar jie pakliūna į vaizdavimo erdvę – nupjautinę piramidę.

Taigi visų pirma, reikia turėti vaizdavimo erdvės plokštumų lygtis, yra du būdai jas gauti:

- Virtuali kamera. Bet kuriuo laiko momentu turime stebėtojo poziciją, žiūrėjimo kryptį, bei matomumo kampą (angl. fov). Atvaizdavimo transformacijos vykdomos taip, kad gautas vaizdas atitiktų tai ką „filmuoja“ virtuali kamera. Vaizdavimo erdvės plokštumos gaunamos iš šios minėtų kameros parametrų. Labai patogu naudoti, bet įgyvendinimas reikalauja geros programos architektūros, bei gilių trimatės grafikos programavimo žinių.
- Išgavimas iš pasaulio-vaizdavimo-projekcijų matricos. Ši matrica, bei jos komponentės paprastai saugojamos ir naudojamos grafinėse bibliotekose. Šis būdas aprašytas [GH].

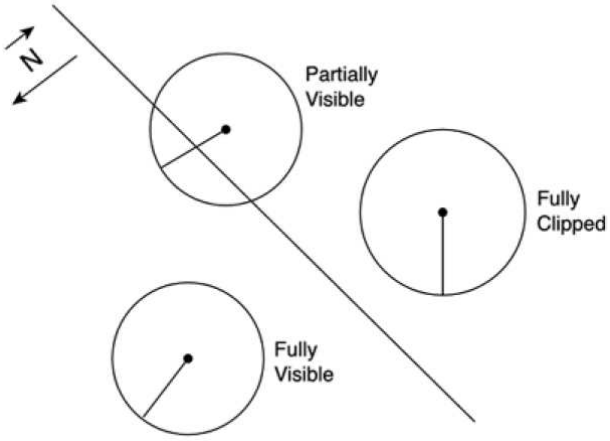


4 pav. Matomi (1,4) ir nematomi (2,3) objektai [internetas: http://developer.valvesoftware.com/wiki/Visibility_determination]

Jei tikrintume kiekvieną objekto trikampio padėtį nupjautinės piramidės atžvilgiu, vėl grįžtume prie brangaus laiko atžvilgiu atkirtimo, todėl kiekvienam objektui naudosime dengiantį apvalkalą. Jei tikrai žinome, kad apvalkalas yra už vaizdavimo erdvės ribų, objektą išmetame iš tolimesnių skaičiavimų šiame kadre, kitu atveju manysime, kad objektas matomas t.y. toliau jį įprastai apdorosime (įskaitant ir atkirtimą).

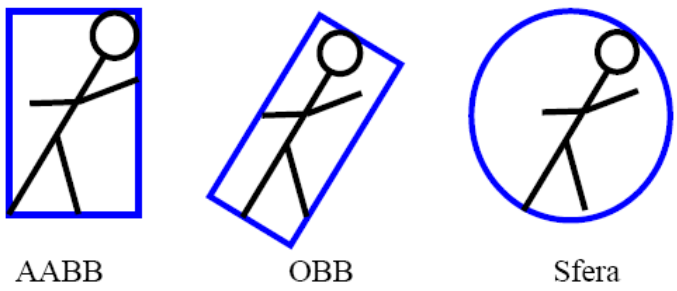
Paprasčiausias dengiantis apvalkalas yra sfera, jis reikalauja mažiausiai skaičiavimų ir atminties, kaip matome 5 pav., įvertinti ar sfera yra vienoje ar kitoje plokštumos pusėje, ar ją

kerta, reikia tik palyginti atstumą nu plokštumos iki sferos centro su sferos spinduliu [San03]. Sferinis apvalkalas aprašomas dviem parametrais: centras (objekto geometrinis centras) ir spindulys (maksimalus atstumas nuo centro iki bet kurios objekto viršūnės). Sferinis apvalkalas nekinta, sukant jo viduje esantį objektą, todėl kartą jį apskaičiavus bei centrą prilyginus objekto pozicijai, sferinis apvalkalas lieka praktiškai statinė objekto savybė.



5 pav. Sferos padėtis plokštumos atžvilgiu [San03]

Sferiniai apvalkalai dažnai apima daugiau nereikalingos erdvės nei pavyzdžiui gabaritiniai (6 pav.). Pagal ašis išlygintas gabaritinis apvalkalas (AABB, angl. axis aligned bounding box) objektus apima tiksliau, bet ir jo palyginimas plokštumos atžvilgiu sudėtingesnis, pagal objektą orientuotas gabaritinis apvalkalas (OBB angl. oriented bounding box), apima objektą dar tiksliau, bet skaičiavimai dar sudėtingesni. Galima sugalvoti ir dar sudėtingesnių apvalkalų, bet praktikoje paprastai apsiribojama sferomis ir AABB.



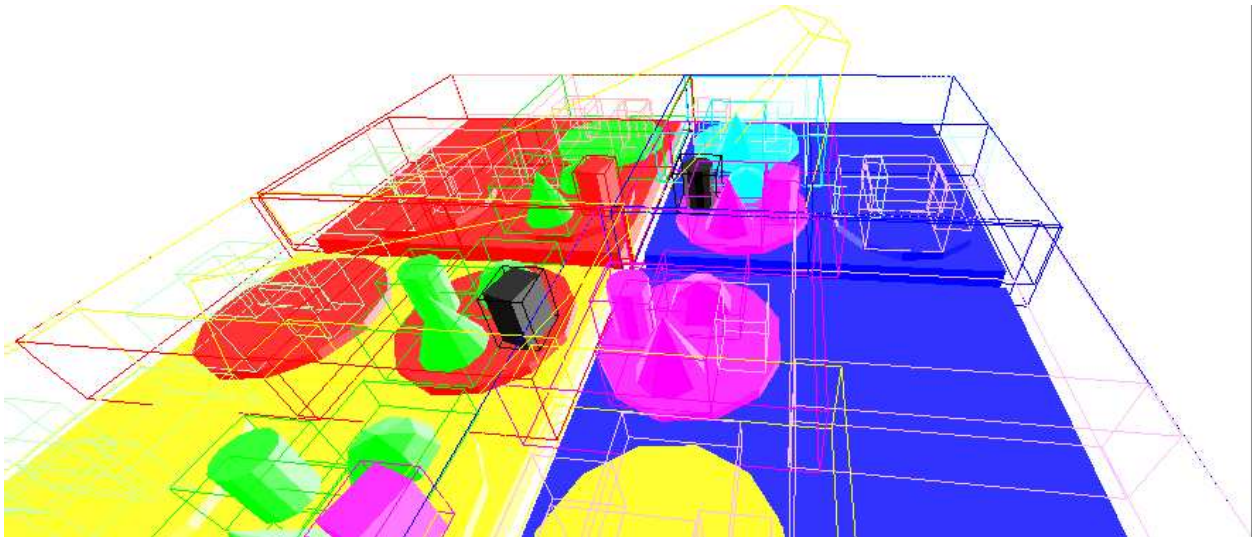
6 pav. Sferos padėtis plokštumos atžvilgiu [Len07]

AABB dažniausiai aprašomas dviem taškais min ir max, kur min=(min_x, min_y, min_z), max=(max_x, max_y, max_z) ir min_x yra mažiausia x koordinatė visose objekto viršūnėse, kitos koordinatės gaunamos analogiškai. Pasukus šio apvalkalo viduje esantį objektą arba pajudinus jo atskirą dalį, AABB reikia perskaičiuoti, tai pakankamai imli skaičiavimams procedūra.

Šiuos skaičiavimus galima sumažinti taip: gaubiamą objektą suskaldyti į statines (nekintamos geometrijos) dalis kurių kiekvienai priskirti po gaubiantį apvalkalą, tada

perskaičiuojant viso objekto gaubiantį apvalkalą, skaičiavimams reiktų naudoti tik vidinių apvalkalų parametrus, o ne visą geometriją. Tai yra vienas iš hierarchinių geometrinių modelių pirmą kartą pasiūlytų [Cla76] pritaikymų.

Hierarchiniai gaubiamieji apvalkalai naudojami ir statinėms geometrijoms, kai jos yra žymiai didesnės už vaizdavimo erdvę (7 pav.), juk mažai naudos iš išrinkimo, jei didžiulis objektas visada piešiamas visas, kad ir kurią jo dalį mes matome. Praktiškai, kad išnaudoti hierarchinių gaubiamųjų apvalkalų pliusus, taip pat tenka skaidyti į tokią pačia hierarchiją ir objekto geometriją, juk jeigu į matomą erdvę patenka tik vienas gaubiantis apvalkalas, tai ir piešti reikia tik atitinkamą geometriją. Taip skaidant objektą galų gale gauname labai daug mažų objektų, kurie visi piešiami jei visas objektas yra matomas. Iš kitos pusės kiekvienas naujos geometrijos piešimas (angl. render call) taip pat reikalauja papildomų resursų. Taigi kiekvieną kartą reikia ieškoti aukso vidurio, iki kokio lygio skaidyti objektą. Klausimas gilesniam tyrimui: gal verta kiekvienam hierarchijos lygiui saugoti po atskirą geometrijų sąrašą.

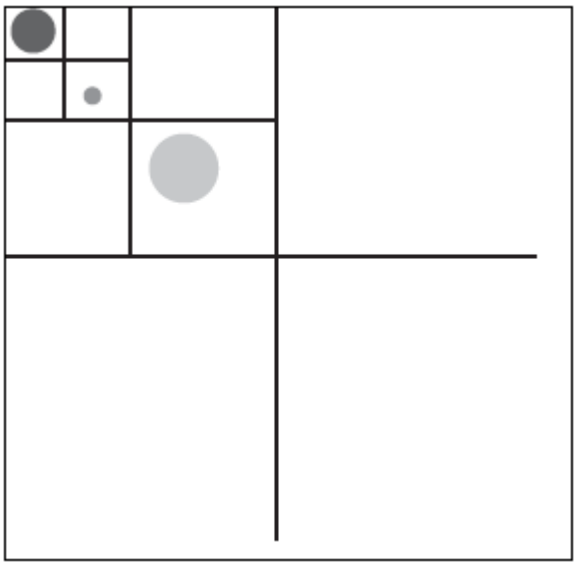


7 pav. Hierarchinis gabaritinis apvalkalas [SJ02]

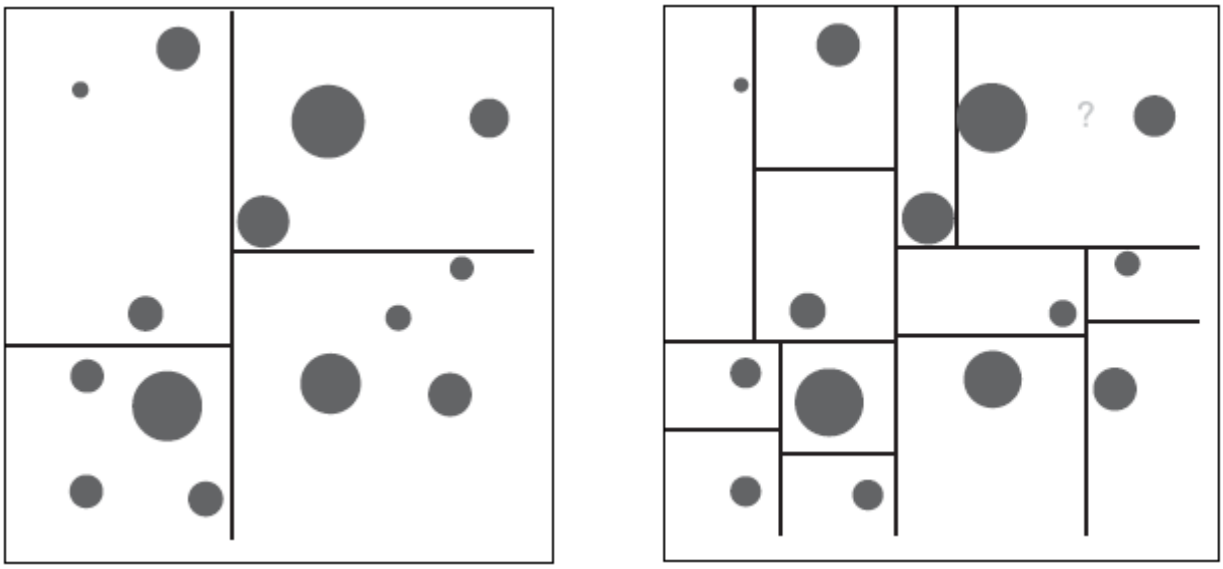
Hierarchinį skaidymą galima naudoti ne tik vienam objektui skaidyti, o ir skaidyti visą erdvę, tada padalinus visus objektus į hierarchinę dengiančių apvalkalų struktūrą, galėtume žymiai sumažinti matomumo nustatymų skaičių, t.y. jei tėvinis apvalkalas nepatenka į matymo erdvę, tai iškart galime atmesti visus jo vaikus.

Yra keletas metodų erdvės skaidymui, vienas iš jų, tai dvejetainiai, ketvirtainiai (8 pav.) ir aštuntainiai medžiai (atitinkamai naudojami vienmatėje, dvimatėje ir trimatėje erdvėse). Tai metodas labai paprastai atvaizduojamas grafiškai, erdvė tiesiog dalinama į lygias dalis, po to, jei reikia, procedūra kartojama. Deja naudojant tokį skaidymą, galime gauti situaciją, kai visi objektai patenka į vieną iš aštuonių erdvės dalių, kurią suskaidžius toliau vėl gauname tą patį.

Tokiu atveju geriau naudoti k-dimensinius medžius (9 pav.). Pagrindinis skirtumas tarp šių metodų yra tas, kad k-dimensiniai medžiai daliną erdvę ne pagal jos tūrį, o taip, kad kiekviename padalijime būtų vienodas objektų kiekis. Padalijimas vykdomas paeiliui kiekvienai ašiai, 9 pav., pirma dalijama pagal x, po to pagal y ašį, dėl to b atveju, galima arba stabdyti padalijimą, arba pridėjus nereikalingą horizontalią liniją, ir sekanti vertikali jau užbaigtų dalijimą [Tre04]..



8 pav. Dvimačiu atveju gaunamas ketvirtainis medis [Tre04]



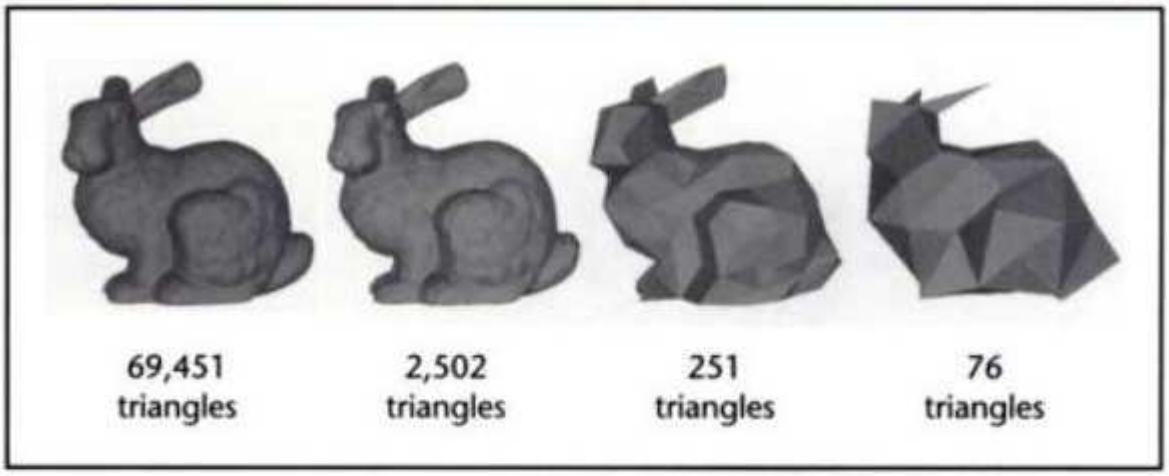
a) pirmas žingsnis **b) antras žingsnis**

9 pav. k-dimensinis medis [Tre04]

Nemažai išrinkimo optimizacijų buvo pasiūlyta [AM99] ir [AM00], vėliau jos dar patobulintos ir detalizuotos [SJ02]. Šios optimizacijos detaliau bus nagrinėjamos darbo eigoje.

1.3. Detalumo adaptacija

Iki šiol buvo nagrinėjamas klausimas piešti objektą ar nepiešti. Įsivaizduokime, kad žiūrime į horizontą: į regėjimo lauką patenka nesuskaičiuojamas kiekis objektų, bet didelė jų dalis yra per toli, kad įžiūrėtume detales, ar netgi išvis atpažintume objektą. Detalumo adaptacijos idėją puikiai galima paaiškinti vienu paveikslėliu (10 pav.).



(a)

(b)

10 pav. Detalumo adaptacijos idėja a) skirtingų detalumų modeliai ir b) jų išdėstymas skirtingu atstumu nuo stebėtojo [LRC+02]

Matome, kad toli esančiam triušiui atvaizduoti užtenka 76 trikampių. Detalumą galima mažinti ir labai greitai judantiems (nebūtinai tolimiems) objektams.

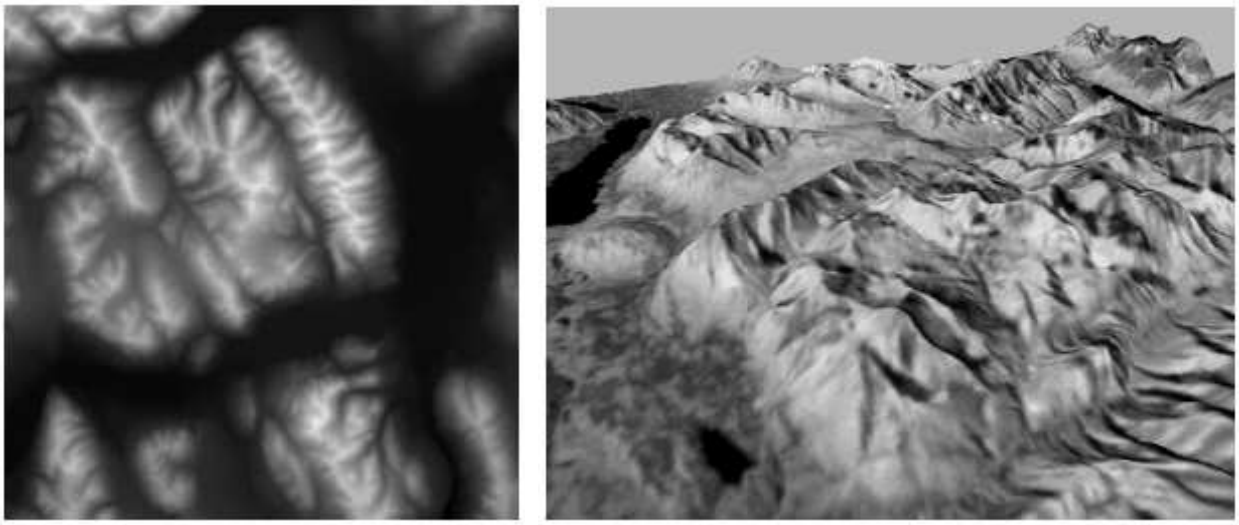
Tradicinis metodas – diskreti detalumo adaptacija. Programos įkrovimo metu arba prieš tai, sugeneruojami kelių detalumo lygių modeliai (paprastai turimas pats detaliausias ir generuojami mažesnio detalumo modeliai) kaip parodyta 10 pav. a. Programos vykdymo metu pagal atstumą

iki stebėtojo (arbą kitokią charakteristiką) vaizduojamas reikiamo detalumo objektas. Tokį metodą paprasta įgyvendinti, paprasta suvesti į vaizdavimo sąrašus ar viršūnių buferius (apie kuriuos bus kalbama vėliau). Didelis diskrečios detalumo adaptacijos minusas yra aiškiai matomas peršokimas iš vieno detalumo lygio į kitą.

Tolydi detalumo adaptacija, naudoja specialias struktūras, iš kurių realiu laiku išgaunamas tiksliai norimo detalumo modelis. Tokiu būdu sumažinamas peršokimo efektas, bei sudaromos galimybės pavyzdžiui to paties modelio skirtingas dalis vaizduoti skirtingais detalumo lygiais (pvz. kuo arčiau stebėtojo tuo detaliau).

Pastaroji savybė labai patogi kraštovaizdžių vaizdavimui. Kraštovaizdžiams dažniausiai taikomi specifiniai detalumo adaptacijos metodai (paprastai tolygūs), nes čia galima naudoti labai patogią pradinę struktūrą (iš kurios gauname reikiamo detalumo modelį) – aukščių žemėlapį, pvz. 11 pav. a. Dvimačiame paveikslėlyje, trečiasis matmuo (aukštis) nurodomas spalva (kuo šviesesnė, tuo aukštis didesnis).

Detaliau apie struktūras tinkančias tolydžiai detalumo adaptacijai aprašyta [Hop96], apie aukščių žemėlapius [RHS+98]. Kintamo detalumo kraštovaizdžiai toliau bus nagrinėjami darbo eigoje.



a) **b)**

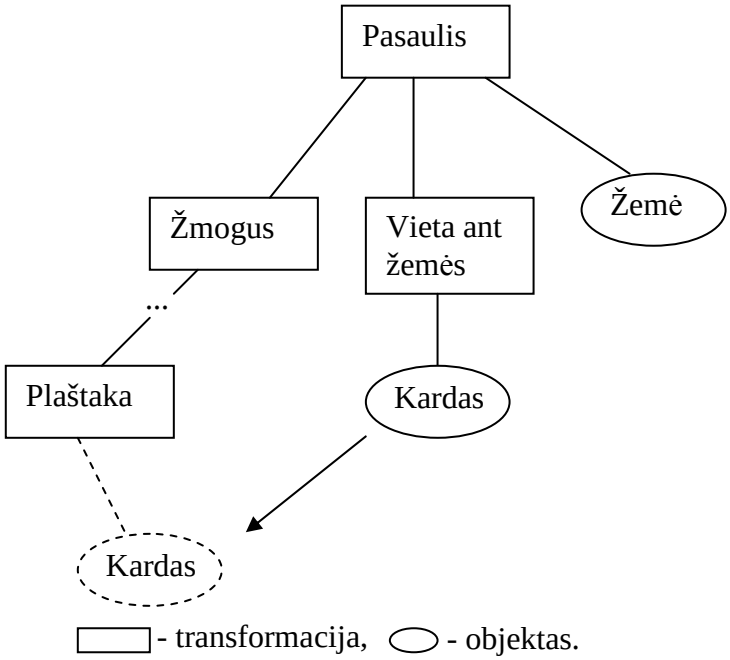
11 pav. a) Aukščių žemėlapis; b) pagal jį sugeneruotas kraštovaizdis [RHS+98]

Beje detalumo adaptaciją galima taikyti ne tik geometrijai (viršūnių, trikampių mažinimui), bet taip pat ir tekstūroms bei šešėliavimui [OKS03]. Detalumo adaptacijos specifinis variantas tekstūros žinomas angliškų terminu mipmaps, ne tik pagreitina bet ir pagerina vaizdo kokybę tolimiems objektams.

1.4. Scenos grafas

Scenos grafas – tai objektų ir jų transformacijų struktūra. Dažniausiai pasitaikanti scenos grafo realizacija yra medis sukonstruotas naudojant kompozicijos šabloną (angl. pattern), kurio mazguose gali būti transformacijos, o lapuose yra pats objektas, pvz. vaizdas (tekstūros, viršūnės), garsas, šviesa ar animacija.

Scenos grafo esminė savybė yra ta, kad visada galima objektą pririšti prie bet kurio mazgo. Vienas iš pliusų yra tai, kad piešiant daug vienodų objektų, patogu naudoti vieną objektą ir daug rodyklių į jį. Naudojant scenos grafą labai patogu animuoti pavyzdžiui tokį scenarijų: „žmogus paima nuo žemės kardą ir puola su juo į kovą“, reiktų tik perkelti rodyklę į kardą iš vienos medžio tipo struktūros vietos į kitą (kardo paėmimo metu) kaip parodyta 12 pav.; jokių papildomų skaičiavimų kardo pozicijai nustatyti nebereiktų.



12 pav. Scenos grafo pavyzdys

Pagrindinė scenos grafo paskirtis – galimybė patogiai valdyti trimatę sceną. Taip pat, naudojant scenos grafą galima sumažinti pasirinktinai transformacijų, tekstūrų keitimų ar kitokių operacijų kiekį (dažniausiai, kaip ir pavyzdyje, sutaupomas transformacijų keikis) ir, jei keliose vietose naudojama ta pati statinė geometrija, sutaupome daugybę atminties ir dar kartą transformacijų.

Reiktų kažkaip susieti scenos grafą ir aukščiau aprašytus objektų ir erdvės padalijimus, kuriuos naudoja išrinkimas. Objekto padalijimas turėtų puikiai tikti vietoj scenos grafe esančio geometrijos lapo. Erdvės padalijimas, visgi, visai nesusijęs su objektais, o tik su erdve, ir paprasčiausiai leidžia vaizduoti visus objektus tam tikroje erdvėje jei ji yra matoma. Taigi,

apjungti scenos grafą ir erdvės padalijimą į vieną hierarchinę struktūrą be papildomų apribojimų, bus sudėtinga.

1.5. Bendravimo su vaizdo plokšte optimizacijos

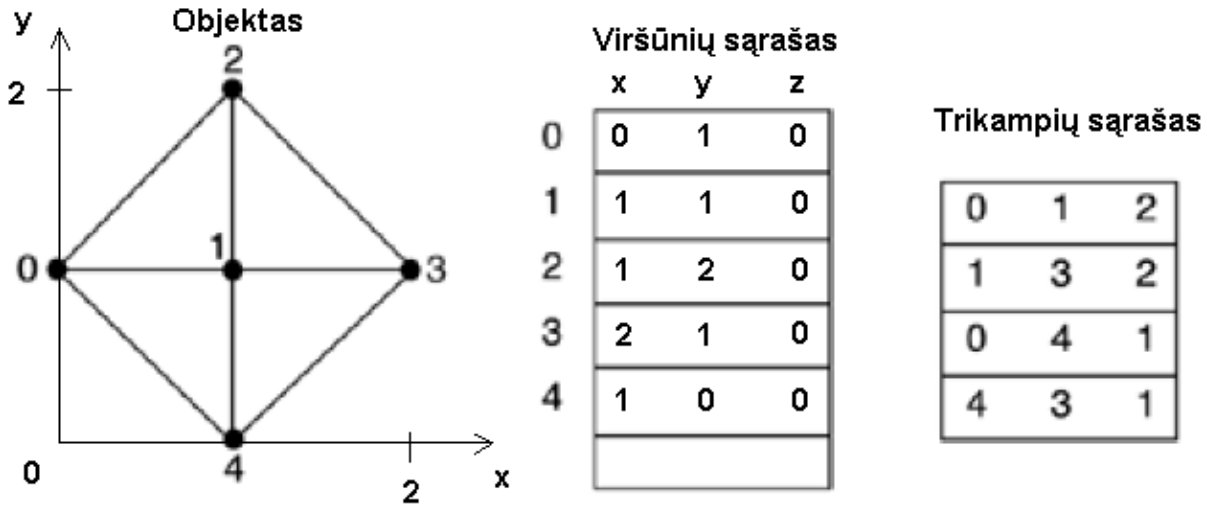
Komandos „kelionė“ iš taikomosios programos į grafinę plokštę užima santykinai daug laiko, todėl jos nesiuntinėjamos po vieną, o kaupiamos specialiame buferyje, o vėliau siunčiamos visos sykiu, kaip viena didelė komanda, taip sutaupoma daug laiko. Komandos šiame buferyje išsiunčiamos kai buferis pilnai užpildomas, kaip prieš sukeičiant vaizdavimo buferius, arba padavus specialią komandą.

Kiekviena grafinės bibliotekos komanda (užrašyta aukšto lygio programavimo kalba), prieš dedant ją į komandų buferį, prieš tai dar yra sukompiliuojama (į grafinei plokštei suprantamą kalbą). Kaip žinia, kiekviena geometrijos viršūnė, jos tekstūravimas, normalės nurodymas ir t.t. yra komandos. Taigi jei turime nekintančią geometriją (arba geometriją, kurios animacija gaunama tiesiog judinant nekintančias jos dalis) ar kitokią nekintančią grafinės plokštės komandų seką, galime tiesiog įsiminti jau sukompiliuotą visų šių komandų variantą iš komandų buferio ir tiesiog įkopijuoti jį ten kiekvieną kartą naudojant. Taip sutaupoma daug komandų kompiliavimo laiko, beje ir funkcijų kvietimo laiko. Šis sukompiliuotas vaizdo plokštės komandų gabalas vadinamas vaizdavimo sąrašu.

Šis metodas sunaudoja nemažai papildomos atminties, o svarbiausia – netinka animuotų objektų atvaizdavimui. Dar vienas būdas optimizuoti bendravimą su vaizdo plokšte yra viršūnių buferiai. Vietoj to, kad kiekvieną viršūnę nurodyti atskira komanda, galima vaizdo plokštei perduoti didelį viršūnių buferį [San03], kur trikampių piešimo atveju, pirmos trys viršūnės sudarys trikampį, taip pat kaip ir sekančios trys ir t.t.

Šį metodą dar galima kaip reikiant optimizuoti. Įprastu būdu vaizduojant objektus, viršūnes dažnai tenka dubliuoti, nes kiekvieną viršūnę dažniausiai naudoja keli trikampiai. Sprendimas – indeksų buferis. Į vieną buferį išrašome visas reikalingas viršūnes jų nedubliuojant, o į kitą – norimus trikampius, nurodome kaip pirmojo buferio eilutės numerius (13 pav.).

Visų pirma taip sutaupome nemažai atminties, o svarbiausia – mažiau viršūnių reikia atvaizduoti ir prieš tai transformuoti. Viršūnių buferis kartu su indeksų buferiu ne tik tinka dinaminiams objektams, bet dažnai greičiu net lenkia vaizdavimo sąrašus.



13 pav. Trikampių sąrašo sudarymas iš viršūnių sąrašo indeksų, modifikuotas [San03]

Šie du metodai realizuoti visose dabartinėse personalinių kompiuterių vaizdo plokštėse. Kaip jais pasinaudoti, aprašyta [WL04].

1.6. Metodų apžvalgos išvados

- Kompiuterinės grafikos spartinimo metodų yra daug ir dauguma jų gana sudėtingi.
- Galimi tolimesnio darbo scenarijai:
    1. Realizuoti vieną metodą ir tyrinėti galimas jo optimizacijas;
    2. Realizuoti 2-3 paprastesnius metodus, nesigilinant į galimas optimizacijas;
    3. Ieškoti programinių paketų, kuriuose būtų realizuota dalis šių metodų, ir išmokti jais naudotis.
- Toliau darbe bus bandoma suderinti pirmą ir trečią scenarijus.

2. Principiniai sprendimai

2.1. Kintamo detalumo kraštovaizdžiai

Iš aukščiau apžvelgtų kompiuterinės grafikos metodų, detaliau bus nagrinėjami kintamo detalumo kraštovaizdžiai. Kraštovaizdžiai reikalingi visuose kompiuterinės grafikos taikymuose, kur vaizduojamos išorinės scenos (skrydžių stimuliatoriai, kompiuteriniai žaidimai, geografiniai įrankiai). Dažniausiai šiuose taikymuose jie užima didžiąją dalį visos grafikos, taigi jų atvaizdavimą labiausiai varta spartinti.

Pilnavertės kraštovaizdžio realizacijos paprastai naudoja daugumą aukščiau aprašytų kompiuterinės grafikos spartinimo metodų: geometrija laikoma viršūnių ir indeksų buferiuose; optimali atvaizdavimo tvarka parenkama scenos grafo; išrinkimas ir erdvės padalijimas neleidžia vaizduoti nematomos geometrijos; speciali detalumo adaptacija yra kraštovaizdžio generavimo ir atvaizdavimo algoritmo esmė; galų gale atkirtimas (kaip ir visose trimatės grafikos programose) sutvarko trikampius prieš pat atvaizdavimą.

Kaip jau minėta 1.3 skyriuje, kraštovaizdžiams saugoti dažniausiai naudojami aukščių žemėlapiai, jie ne tik užima mažai atminties, bet ir yra labai patogi struktūra generuoti norimo detalumo trikampių tinklelį. Pagrindinis šios struktūros minusas yra tai, kad joje negalima atvaizduoti iškyšulių ir olų.

Kaip rašo [Bre05], kraštovaizdžių algoritmai skirstomi į tris kategorijas:

1. **hierarchiniai algoritmai**, rekursyviai skaidantys kraštovaizdį naudojant įprastas duomenų struktūras (dažniausiai kvadratus);
2. **nereguliarūs trikampių tinklai,** kur trikampiai gali būti bet kokios formos ir dydžio, kad kuo tiksliau atitiktų detaliausią geometrijos lygį;
3. **vaizdo plokšte paremti metodai,** kurie naudoja retai kintančius viršūnių buferius, efektyviam atvaizdavimui.

Pirmosios dvi kategorijos iš esmės yra jau pasenusios, nes jų tikslas yra kiek įmanoma sumažinti trikampių skaičių, kurie bus siunčiami vaizdo plokštei, o tai reikalauja daug procesoriaus (ir dažnai atminties) resursų. Kadangi dabartinės vaizdo plokštės specifines užduotis atlieka net žymiai greičiau už pagrindinį procesorių, reikia ir naujų vaizdavimo algoritmų. Trečiosios algoritmų kategorijos pagrindinis tikslas yra parinkti tokias duomenų struktūras, kad būtų galima atvaizduoti kuo daugiau trikampių (viršūnių ir indeksų buferių pagalba, žr. 1.5 skyrių), ir tik po to stengiamasi sumažinti tų trikampių skaičių, bet svarbiausia, kuo mažiau apkraunant procesorių.

Yra ir kombinuotų algoritmų, bandančių suderinti senuosius algoritmus su naujomis grafinių plokščių galimybėmis, žymiausias jų yra „Chunked LOD“ [Ulr02]. Vėliau atsirado ir du

nauji algoritmai: geomipmap [Boe00] ir geoclipmap [LH04]. Abu šie metodai idėjas pasiskolino iš atitinkamų metodų (mipmap ir clipmap) plačiai naudojamų tekstūroms.

Nors geoclipmap algoritmas yra žymiai naujesnis, [Bre05] tyrimas, darytas norint parodyti jo pranašumą, parodė kad:

1. geoclipmap laimi prieš geomipmap tik labai dideliuose kraštovaizdžiuose (aukščių žemėlapio formatas 2049x2049)
2. geoclipmap laimi prieš geomipmap tik naujose aukštos kokybės video plokštėse (NVIDIA 6800)
3. jų geomipmap algoritmo realizacija nesaugojo aukščio reikšmių vaizdo plokštėje (dėl to geomipmap įvertinimas gautas prastesnis už galimą)
4. geoclipmap sunkiai pasiduoda geometrijos pakitimams, t.y. pasikeitus vaizduojamai geometrijai, reikėtų perskaičiuoti visą algoritmo pasiruošimo dalį

Išvados 1, 2, 4 rodo, kad yra daugybė taikymų kur geomipmap algoritmas šiuo metu yra geriausias. Išvados 2, 3 ir tai, kad geoclipmap algoritmas labiau apribotas, sunkiau pasiduoda modifikacijoms nei geomipmap, rodo kad dar nėra aiškiai žinoma kuris iš jų geresnis likusiuose taikymuose. Taigi toliau **šiame darbe bus analizuojamas geomipmap algoritmas**.

2.2. Grafinės bibliotekos ir grafiniai varikliukai

Pirmiausias dalykas dirbant su kompiuterine grafika yra grafinė biblioteka. Praktiškai visos kompiuterinės grafikos programos naudoja grafinę biblioteką. Grafinė biblioteka suteikia galimybę programuoti kompiuterinę grafiką aukštesniame lygyje ir tuo pačiu garantuoja efektyvią realizaciją, kurios didžioji dalis, beje, yra aparatūrinė. Kitaip sakant, grafinės bibliotekos pagrindinė dalis yra sąsaja su vaizdo plokšte.

Yra tik dvi viešam naudojimui skirtos personalinių kompiuterių aparatūrą efektyviai išnaudojančios grafinės bibliotekos: OpenGL ir DirectX. OpenGL yra multiplatforminė biblioteka (veikia visose populiariose personalinių kompiuterių operacinėse sistemose), užtat DirectX, kuri veikia tik Windows aplinkoje, yra patogi kompleksinė biblioteka su visu kompiuterinei grafikai reikalingu papildomu funkcionalumu (pvz. garsas, video rodymas, klaviatūros, pelės ir kitų įvedimo įrenginių valdymas).

Turint didelį kiekį norimų išanalizuoti kompiuterinės grafikos technologijų (išdėstyta aukščiau), norėtųsi dar aukštesnio lygio programavimo galimybių. Jei grafinė biblioteka jau leidžia operuoti kompiuterinės grafikos elementais (pvz. taškas, trikampis, tekstūra), tai gal yra ir priemonių leidžiančių operuoti net kompiuterinės grafikos technologijomis (pvz. scenos grafas, erdvės padalijimas, detalumo adaptacija).

Grafinis varikliukas – tai kompiuterinės grafikos technologijų, optimizacijų ir pagalbinių priemonių paketas. Tai, iš tikrųjų, yra pagrindinė kompiuterinės grafikos taikomosios programos dalis, jos branduolys, labai dažnai kuriamas kartu su ta programa kaip neatskiriama dalis. Vis tik kai kurios įmonės išleidžia varikliuką ir kaip atskirą produktą (kai kurios netgi kuria jį kaip atskirą produktą), labai brangų produktą, nes pirkėjai būna vienetiniai. Yra ir keletas atviro kodo grafinių varikliukų, trumpai apibūdinsiu du populiariausius.

- Irrlicht – tai labai paprastas naudoti grafinis varikliukas, papildomai turintis ir garso valdymo funkcionalumą, vis tik paprastumas naudoti neigiamai atsiliepia jo lankstumui. Gana paprasta naudotis realizuotu funkcionalumu, bet sudėtinga jį papildyti nesugadinant architektūros.
- OGRE – atvirkščiai, labai universalus, bet sudėtingas naudoti, išimtinai grafinis varikliukas. Norint pasinaudoti kokia nors technologiją, dažniausiai būtina suprasti jos veikimo principus. OGRE puikiai išnaudoja objektinio projektavimo ir dinaminių bibliotekų privalumus, yra galimybė pačiam realizuoti norimą technologiją, ir integruoti ją į OGRE architektūrą visiškai nekeičiant paties varikliuko.

Grafinis varikliukas yra nepakeičiamas įrankis, norint per kuo trumpesnį laiką sukurti pilnavertę kompiuterinės grafikos taikomąją programą ar išbandyti palyginimui kuo daugiau

kompiuterinės grafikos technologijų. OGRE atveju – tai netgi puiki priemonė detaliai išanalizuoti, suprasti ar optimizuoti šias technologijas. Taigi, toliau **magistrinis darbas bus vykdomas naudojant OGRE grafinį varikliuką**.

3. Darbo eiga

3.1. Pirmas bandymas

Prieš gilinantis į algoritmą, buvo nuspręsta palyginti jo efektyvumą su statiniu (tinkleliniu) kraštovaizdžiu. Tam tikslui turimas tinkleliais kraštovaizdis buvo konvertuojamas į aukščių žemėlapį.

Didelė problema su aukščių žemėlapiu yra jo formatas – norint, kad galutinis rezultatas nebūtų laiptuotas, reikia 16-os bitų pilkos spalvos atspalvių. Deja toks formatas yra nėra palaikomas daugumos grafinių redaktorių, bent jau .dds (DirectX texture) failams, kurie yra labai populiarūs trimatės grafikos realaus laiko atvaizdavime, ir .bmp (bitmap) failams, kurie yra patys paprasčiausi, todėl dažniausiai norisi pradėti nuo jų.

Kraštovaizdžio konvertavimui buvo naudojamas 3ds Max programos įskiepis OgreMax, turintis funkciją iš tinklelinio modelio sugeneruoti aukščių žemėlapį 16-os bitų .dds formatu, todėl po to tiesiogiai redaguoti aukščių žemėlapį galimybės nebuvo. Tiesioginiam kraštovaizdžių aukščių žemėlapių kūrimui buvo naudojama .png ir .tga formatai, kurių 16-os bitų pilkos spalvos atspalviai dažnai palaikomi.

Lyginant statinį ir dinaminį kraštovaizdžius, vizualiniai (kokybiniai) skirtumai labiausiai priklauso nuo saugojimo būdo, t.y. tiesioginis atvaizdavimas arba generavimas iš aukščių žemėlapio (nesvarbu koks algoritmas naudojamas greitesniam jo atvaizdavimui). Pirmas vaizdas atrodė geresnis nei tikėtasi – iš naujo sugeneruotame trimatyje vaizde buvo aiškiai matomi net labai maži įdubimai.

Žinoma pastebėta ir trūkumų – pirmiausia tai konvertavimo klaidos (14 pav.). Taigi pirmas trūkumas iš statinių kraštovaizdžių sugeneruotiems aukščių žemėlapiams yra reikalingas papildomas rankinis modifikavimas. Kita vertus yra daugybė priemonių tiesiogiai kurti aukščių žemėlapius ir šis procesas yra žymiai spartesnis už standartinį tinklelinių objektų kūrimą.

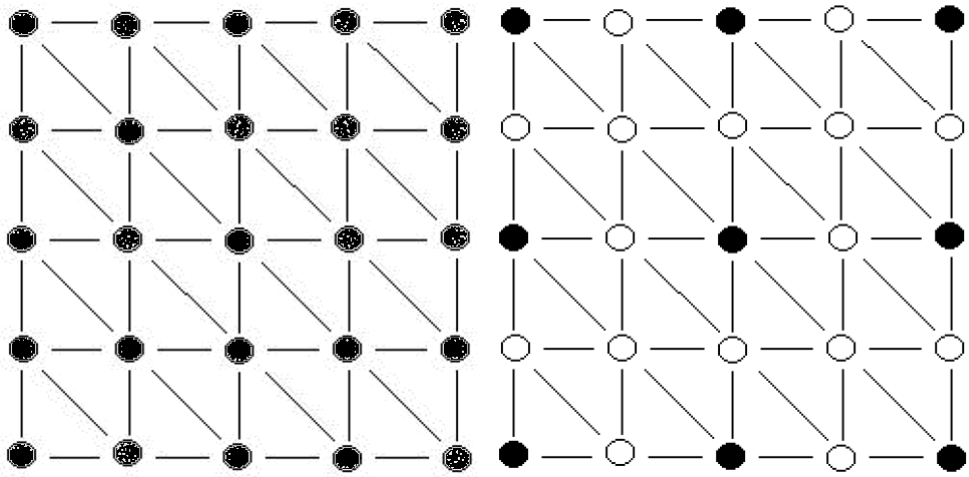
a) padidintas aukščių žemėlapio fragmentas **b) sugeneruoto kraštovaizdžio fragmentas**

14 pav. konvertavimo klaidos (pažymėtos elipsėmis)

Labiausiai nuvylė atvaizdavimo greičio testas – jis buvo tinklelinio kraštovaizdžio naudai. Prieš nagrinėjant priežastis, geriau pereiti prie nuoseklaus algoritmo tyrimo.

3.2. Geometrijos suspaudimo (geomipmap) algoritmas

Čia pateikiamas algoritmas aprašytas [Boe00]. Kiekvienam aukščių žemėlapio elementui (pixel) sukuriama atitinkama viršūnė, taip gaunamas viršūnių tinklas (15 pav. a). Viršūnių x ir z reikšmės (matomos paveiksle) niekada nekinta, o jų y reikšmė nuskaitoma iš aukščių žemėlapio. Viršūnių (taigi ir žemėlapio elementų) ir horizontaliai ir vertikaliai turi būti 2^n+1, kad gautume 2^n kvadratų. Tokios struktūros privalumas yra galimybė sudėti visas vienos srities viršūnes į viršūnių buferį greitam atvaizdavimui, be to, norint sumažinti detalumą, užtenka nurodyti kitą indeksų buferį (15 pav. b). Nedidelis trūkumas yra tai, kad kraštinės viršūnės yra dubliuojamos kaimyninėse srityse.

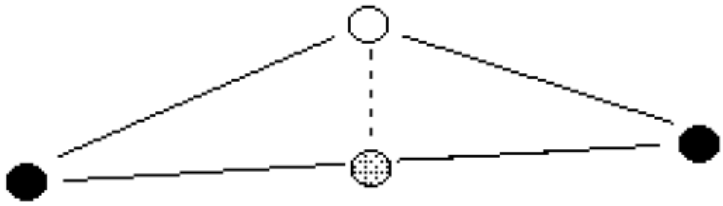


a) **b)**

15 pav. 5x5 viršūnių sritis a) nulinis (aukščiausias) detalumo lygis b) pirmas detalumo lygis [Boe00]

Kraštovaizdžio generavimo metu, sukuriamas ketvirtainis medis (dvimatis erdvės padalijimas, žr. 1.2 skyrių). Kiekvienai sričiai suformuojamas gabaritinis apvalkalas ir prijungiamas prie medžio kaip lapas. Keturi lapai gali būti apjungiami į mazgą su savo gabaritiniu apvalkalu, kol šaknyje gauname viso kraštovaizdžio gabaritinį apvalkalą. Jei lapas (ar aukštesnis mazgas) visai nepakliūna į vaizdavimo erdvę, su šia kraštovaizdžio sritimi neatliekami jokie skaičiavimai.

Kaip parinkti teisingą atstumą *d* kuriuo nutolus nuo kraštovaizdžio srities jau būtų tą sritį galima perjungti į žemesnį lygį? Jei naudosime tiesiog iš anksto nustatytą atstumą, gali matytis prastai atrodantys peršokimai nuo vieno lygio į kitą. Mažindami viršūnių skaičių mes šiek tiek keičiame patį kraštovaizdį (16 pav.), šis kiekvienos viršūnės aukščio pakitimas δ turi būti pakankamai mažas, kad jo nepastebėtume.



16 pav. Naikindami baltą viršūnę, pakeičiame jos aukštį punktyrais pažymėtu dydžiu, dabartinis to taško aukštis lygus įsivaizduojamos pilkos viršūnės aukščiui [Boe00]

Kiekvienos viršūnės δ nėra svarbus, kalbėsime tik apie vienos srities vieno detalumo lygio maksimalų δ. Šį dydį galima suprojektuoti į ekrano erdvę, taip žinosime jo tikrą matomą dydį ε, matuojama ekrano elementais (pixel). Kai ε mažesnis už iš anksto nustatytą leistiną matomą paklaidą τ, galima pereiti į žemesnį detalumo lygį, gali būti kad naujai gauto detalumo lygio ε<τ, vadinasi iškart pereiname į dar žemesnį detalumo lygį ir t.t.

Tikslus ε radimas reikalauja nemažai skaičiavimų, o iš anksto apskaičiuoti ε kiekvienam *d* nepavyks, nes jis priklauso nuo δ, *d*, ir nuo vektoriaus nuvesto iš stebėjimo kameros į nagrinėjamos srities centrą (nuo santykinės kameros pozicijos, srities atžvilgiu). Kai šis vektorius yra horizontalus, ε yra didžiausias. Tarę, kad šis vektorius visada horizontalus, galime iš anksto paskaičiuoti maksimalų ε nagrinėjamam atstumui. Taip gaunamas didesnis nei būtinas detalumas, bet sutaupomas procesoriaus darbas.

Taigi, kiekvienam detalumo lygiui *n*, paskaičiuojam atstumą D_n, kuriuo nutolus nuo srities jau galima naudoti šį detalumo lygį:

$$D_n = |\delta| \cdot C$$

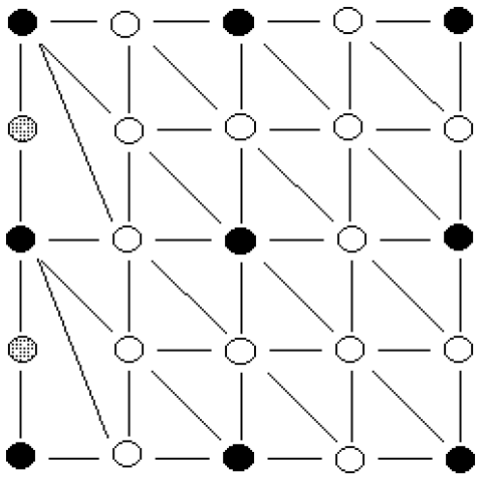
Konstanta C paskaičiuojama taip:

$$C = \frac{A}{T}, \quad \text{kur} \quad A = \frac{n}{|t|} \quad \text{ir} \quad T = \frac{2 \cdot \tau}{v_{res}}$$

Čia *n* yra artimosios atkirtimo plokštumos (near clipping plane) atstumas nuo kameros, o *t*, tos plokštumos viršutinė koordinatė, t.y. pusė aukščio, konstanta *A*, parodo kiek arti įmanoma prieiti prie geometrijos (vadinasi ir kokį jos detalumą įmanoma įžiūrėti). Dydis v_{res} yra vertikali vaizdavimo lango raiška, vadinasi *T* parodo santykinį leistinos klaidos dydį vaizdavimo lango atžvilgiu.

Kai visoms sritims jau priskirti detalumo lygiai, pastebime, kad skirtingų lygių sričių susidūrimuose matomi nedideli tarpai. Beatodairiškas šių tarpų užtaisymo metodas (pvz. toks kuris reikalautų pakeisti viršūnių buferio turinį) gali sugadinti visą algoritmą. Greičiausias žinomas būdas užtaisyti šias skyles yra keičiant tik indeksų buferį, t.y. keičiant tik viršūnių jungimo tvarką, o ne pačių viršūnių pozicijas. Siūloma mažinti detalesniosios srities kraštų

detalumą, kad šis sutaptų su kaimyninės srities detalumu, formuojant trikampių vėduoklę (triangle fan) nuo vienos (viršutinės kairės) bendros viršūnės iki kitos (vidurinės kairės) (17 pav.).



17 pav. Srities besiribojančios su žemesnio detalumo sritimi iš kairės (nepavaizduota) krašto užtaisymas [Boe00]

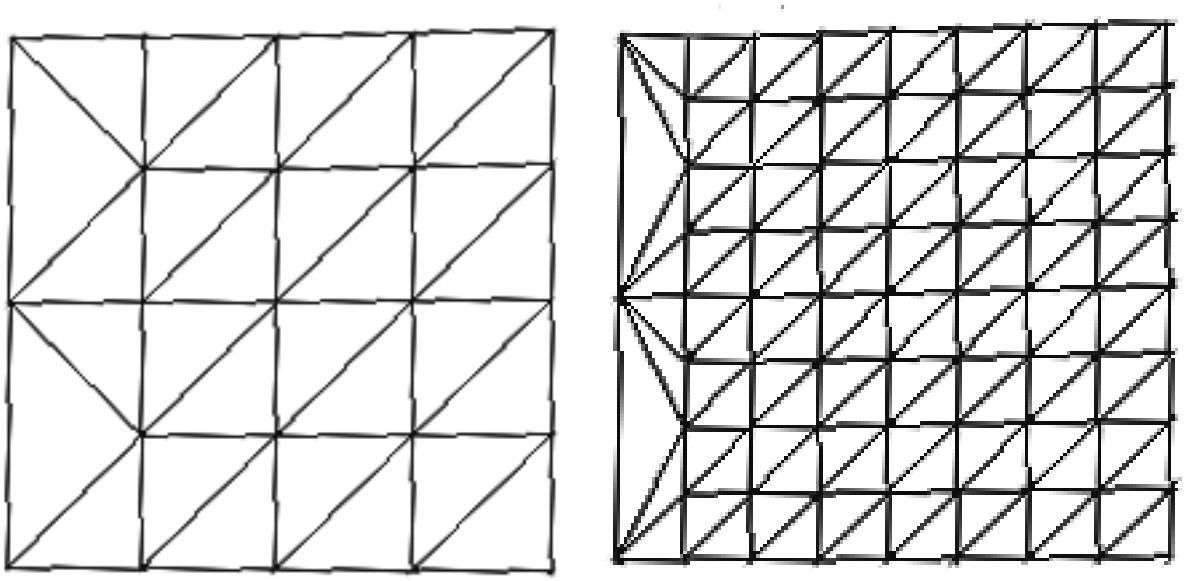
Matome, kad jei kaimyninė sritis bus dar žemesnio lygio, gautume labai ilgus (viename gale smailus, kitame plačius) užtaisančius trikampius. Tokie trikampiai dažnai atvaizduojami prastai, ypač naudojant gouraud apšvietimą. Be tokie trikampiai kenkia ir geometrijos teisingumui (tapatumui su aukščiausiu detalumo lygiu). Įvedus apribojimą, kad dviejų gretimų sričių detalumas gali skirtis ne daugiau kaip vienu lygiu, šis sprendimas yra pakankamai geras, žinoma, toks apribojimas gali smarkiai padidinti detalumą lyginant su iš tiesų reikalingu detalumu. Originaliame algoritme siūloma šiuos užtaisančius trikampius atvaizduoti naudojant specialų trikampių vėduoklių atvaizdavimo būdą, kad sumažinti siunčiamų viršūnių kiekį, bet kažin ar tai verta daryti dabartinėse vaizdo plokštėse, kadangi kitoks trikampių siuntimo būdas reikalauja papildomo kreipimosi į vaizdo plokštę.

3.3. OGRE geomipmap realizacija

Čia bus aptariami pastebėti geomipmap algoritmo realizacijos OGRE varikliuke skirtumai nuo bazinio algoritmo, ir išdėstoma mano nuomonė, kodėl buvo daromi tie skirtumai.

Aukščiau minėta konstanta *A* čia visada laikoma lygi vienetui, kadangi dauguma taikomųjų programų pačios apriboja minimalų atstumą nuo kameros iki geometrijos, tokiu atveju nėra reikalo detalumą laikyti priklausomą nuo artimosios atkirtimo plokštumos (kuri gali būti reguliuojama nepriklausomai nuo leistino atstumo iki geometrijos).

Kraštų užtaisymui čia vietoj vienos trikampių vėduoklės, generuojamos dvi iš skirtingų bendrų viršūnių, ir sujungiamos papildomu viduriniu trikampiu (18 pav.). Taip gauti trikampiai žymiai mažiau įtakoja ir geometrijos detalumą ir gouraud apšvietimą („teisingą“ trikampį galima apibūdinti kaip tokį, kurio bet kurios dvi viršūnės yra nutolusios viena nuo kitos panašiu atstumu)



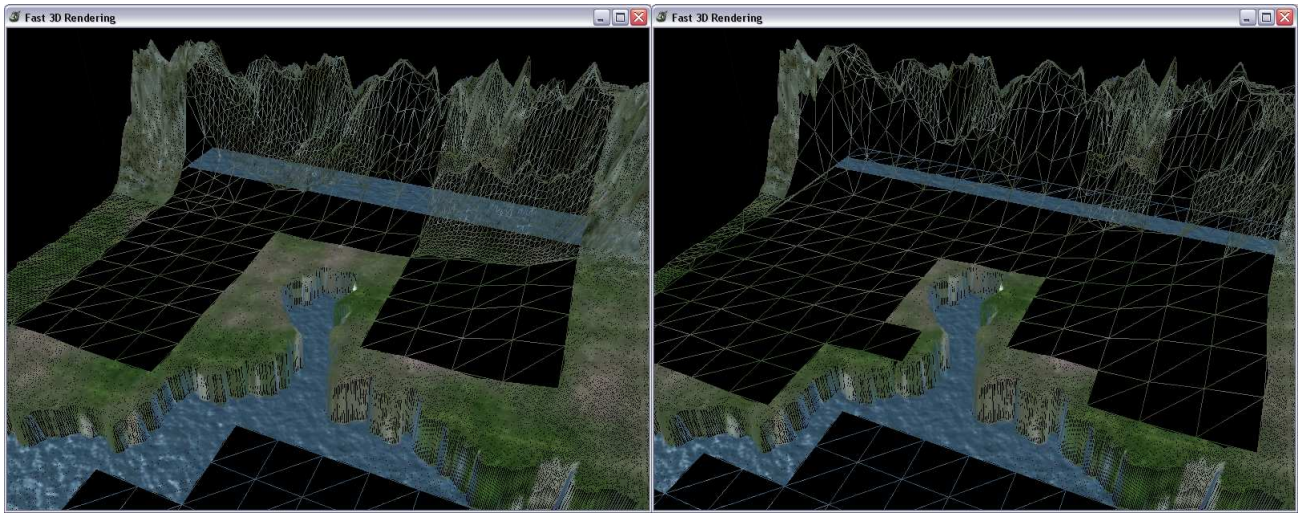
a) vieno detalumo lygio skirtumas **b) dviejų detalumo lygių skirtumas**

18 pav. Kraštų užtaisymas OGRE geomipmap realizacijoje

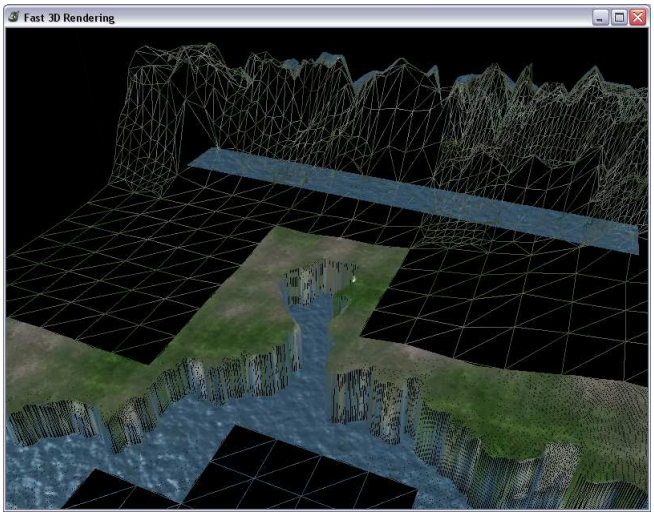
Visi indeksų buferiai, tiek kiekvienam detalumo lygiui, tiek kiekvienai kraštų užtaisymo kombinacijai tam lygiui, kartą sugeneravus yra įsimenami (jie yra vienodi visoms sritims). Prireikus tokį indeksų buferį vėl panaudoti, jis randamas pagal 32 bitų raktą, formuojamą iš srities detalumo lygio, kraštų kurie bus taisomi ir detalumo lygių skirtumų kiekvienam kraštui. Tada vaizdo plokštei nusiunčiamas tik rasto buferio numeris pagal vaizdo plokštės naudojamą numeraciją. Tai labai svarbus momentas, nes indeksų buferio siuntimas į vaizdo plokštę yra labai brangi operacija (nors ir ne tokia brangi kaip viršūnių buferio siuntimas).

3.4. Pirmo bandymo analizė

Buvo pastebėta, kad kraštovaizdžio kraštai yra ypač detalūs (19 pav.), vienas trikampis ne ką didesnis už vaizdo elementą (pixel). Taip yra dėl stačių šlaitų. Kaip buvo minėta 4.2 skyriuje, algoritmas naudoja maksimalios leistinos klaidos įvertinimą, spręsdamas ar galima pereiti prie mažesnio detalumo. Jei detalumas arti šlaitų būtų smarkiai sumažintas, gautume nepageidaujamą vaizdą (20 pav.), t.y. tikrai nebeliktų stačių šlaitų.

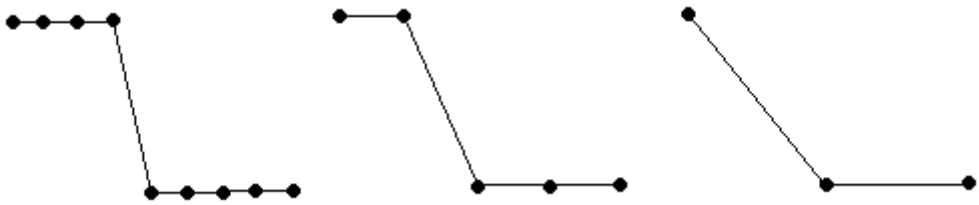


a) srities dydis 65x65 **b) srities dydis 33x33**



c) ištrinti statūs šlaitai kraštovaizdžio kraštuose (srities dydis 65x65)

19 pav. Sugeneruotas kraštovaizdis (matomi trikampiai)



20 pav. Detalumo mažinimas prie stataus šlaito

Sprendimo būdai būtų panaikinti stačius kraštus kur jie nereikalingi, t.y. aukščių žemėlapio kraštuose (16 pav. c) ir mažinti vienos kraštovaizdžio srities dydį (16 pav. b). Su antruoju būdu reikia elgtis atsargiai, nes mažinant srities dydį, daugėja sričių skaičius. Kiekvienos srities atvaizdavimui reikia nurodyti viršūnių buferį ir indeksų buferius, o tai gana brangios operacijos. Šiuo atveju parinkus 33x33 srities dydį (16 pav. b), vaizdavimo greitis šiek tiek padidėjo, bet dažniausiai srities dydis neturėtų būti mažesnis nei 65x65.

Taip pat pirmame bandyme buvo naudojamas per didelės raiškos (513x513) aukščių žemėlapis – iš tokio žemėlapio sugeneruoto kraštovaizdžio detalumas aukščiausiame lygyje yra žymiai didesnis nei statinio kraštovaizdžio iš kurio buvo konvertuojama.

3.5. Apatinės ribos optimizacijos

3.5.1. Mažiau trikampių

Kas būtų jei nevaizduotume trikampių, kurių visų trijų viršūnių aukštis lygus nuliui? Aukštis, kuriame atsidurs viršūnė jei ją atitinkantis aukščių žemėlapio elementas yra visiškai juodas, visada yra nulis, nesvarbu, kur vėliau scenoje bus vaizduojamas kraštovaizdis, nes jo postūmis bus realizuojamas transformacijomis, nuo pradinės pozicijos. Nulinis aukštis dažniausiai pasiekiamas stačiu šlaitu, o statūs šlaitai visada būna aukšto detalumo. Teisingai sukomponuotose scenose atvaizdavimo greitis turi būti maždaug tolygus dydis per visą sceną (svarbiausia, kad niekur nenukristų žemiau leistinos ribos). Taigi, optimizacija šalia stačių šlaitų būtų labai vertinga.

Jei pašalinsime viršūnę iš viršūnių buferio, vėliau bus sugeneruoti neteisingi indeksai. Taigi, kaip ir keičiant detalumo lygį, viršūnes reikia šalinti iš indeksų buferio. Kaip jau buvo minėta, šioje realizacijoje indeksų buferiai yra hešuojami pagal detalumo lygį ir kraštų užtaisymo žymes. Taigi kartą sugeneruotas indeksų buferis, gali būti naudojamas bet kuriai sričiai atvaizduoti. Jei pakeisime bent vieną indeksų buferį, sugadinsime visą kraštovaizdį.

Vadinasi, sritims turinčioms nulinio aukščio viršūnių, reikalingi atskiri indeksų buferiai. Jei indeksus šioms sritims generuotume kiekvieną kadrą, prarastume daugiau laiko negu laimėtume, todėl ir čia kartą sugeneruotus indeksų buferius reikia pakartotinai naudoti. Galima pritaikyti tą patį hešavimo metodą, kaip ir likusiame algoritme. Įprastinės sritys toliau naudos bendrą indeksų buferių rinkinį, o sritys turinčios nulinio aukščio viršūnių, kiekviena turės po atskirą tokį indeksų buferių rinkinį.

Indeksai iš buferio bus šalinami standartinio indeksų generavimo kiekvienam lygiui metu, t.y. prieš dedant naujus tris indeksus į buferį, patikrinama ar juos atitinkančios viršūnės nėra nulinio aukščio, jei yra, šie indeksai į buferį nededami. Tokiu pačiu principu nuliniai trikampiai turi būti pašalinti ir iš kraštų užtaisymo algoritmo.

3.5.2. Mažiau sričių

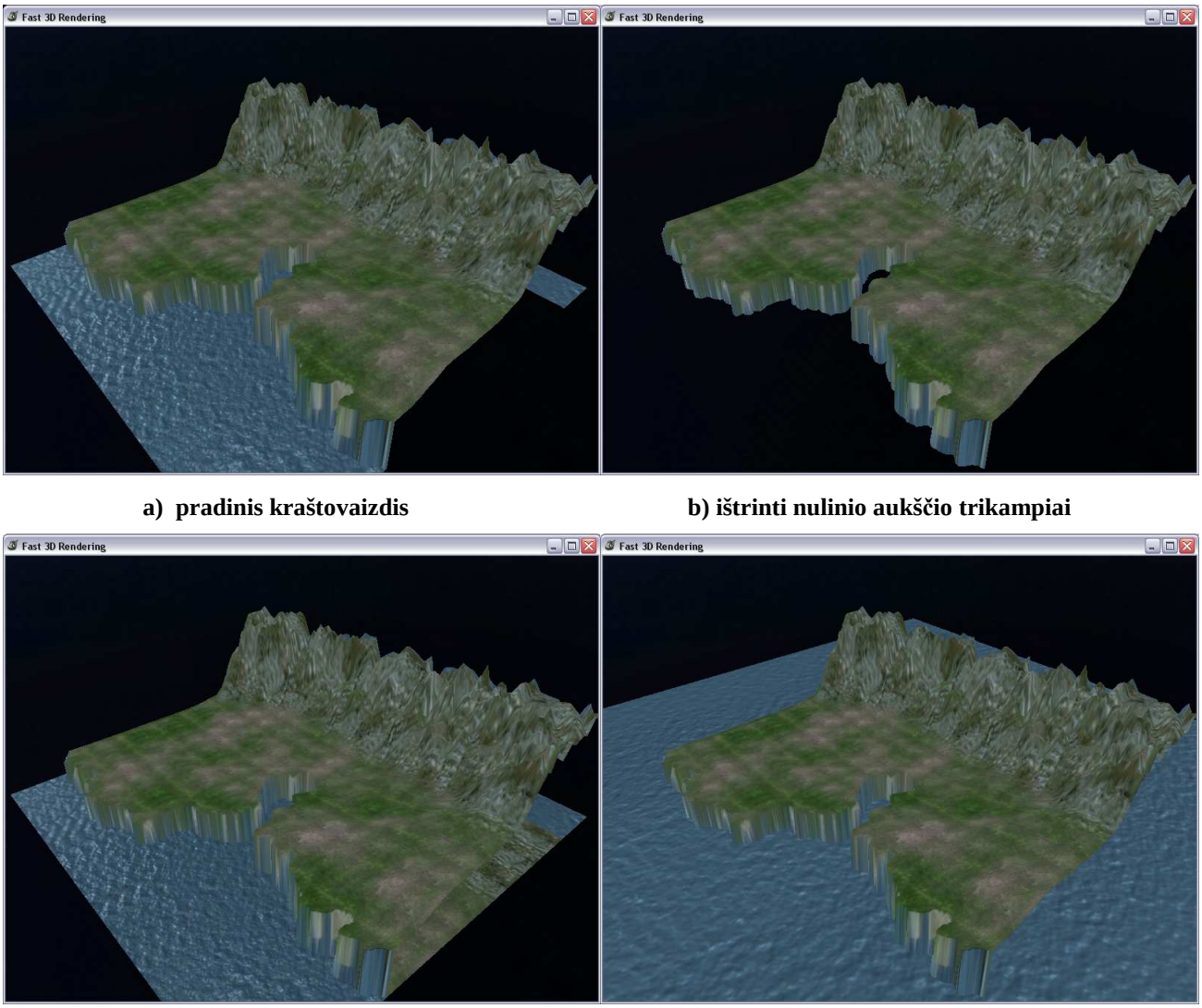
Pagrindinė priežastis dėl kurios daroma ši optimizacija, yra daug nereikalingų trikampių srityje į kurią patenka ir status šlaitas ir plokščia kraštovaizdžio apačia (dažniausiai vanduo arba žemė giliai po vandeniu), pvz. vanduo 16 pav. c. Todėl, iš pirmo žvilgsnio visai nėra būtina naikinti trikampius toliau į vandenį, kur detalumas ir taip labai mažas.

Kita vertus, jei panaikintume visą sritį, sutaupytume ne tik trikampių bet ir kreipinį į vaizdo plokštę (angl. render call), kas yra gana brangi operacija. Taigi sritis, kuri dabartiniame

jos detalumo lygyje neturi nė vienos nenulinio aukščio viršūnės, nebus siunčiama į vaizdo plokštę.

3.5.3. Tarpų užtaisymas

Vietose, kur nebevaizduojami trikampiai, žinoma, liko tarpai (21 pav. b). Kadangi visi panaikinti trikampiai yra nuliniame aukštyje, o kraštovaizdžiai visada vaizduojami taip, kad nebūtų galima pamatyti kas yra po jais, tarpus galima užtaisyti paprasčiausiai pridedant vieną plokštumą (du trikampius) per visą kraštovaizdį. Ši plokštuma bus tekstūruojama lygiai taip pat kaip ir likęs kraštovaizdis ir rezultatas (žiūrint taikomosios programos leistinu kampu) nesiskirs nuo pradinio (21 pav. a ir c). Iš tiesų, automatiškai pridėti šią plokštumą visai nebūtina, netgi geriau palikti galimybę kraštovaizdžio vartotojams šioj vietoj įsidėti tokią plokštumą kokios tik nori (21 pav. d).



a) pradinis kraštovaizdis **b) ištrinti nulinio aukščio trikampiai**

c) pridėta dviejų trikampių plokštuma **d) pridėta didesnė ir kitaip tekstūruota plokštuma**

21 pav. Tarpų užtaisymas

3.5.4. Pritaikymas

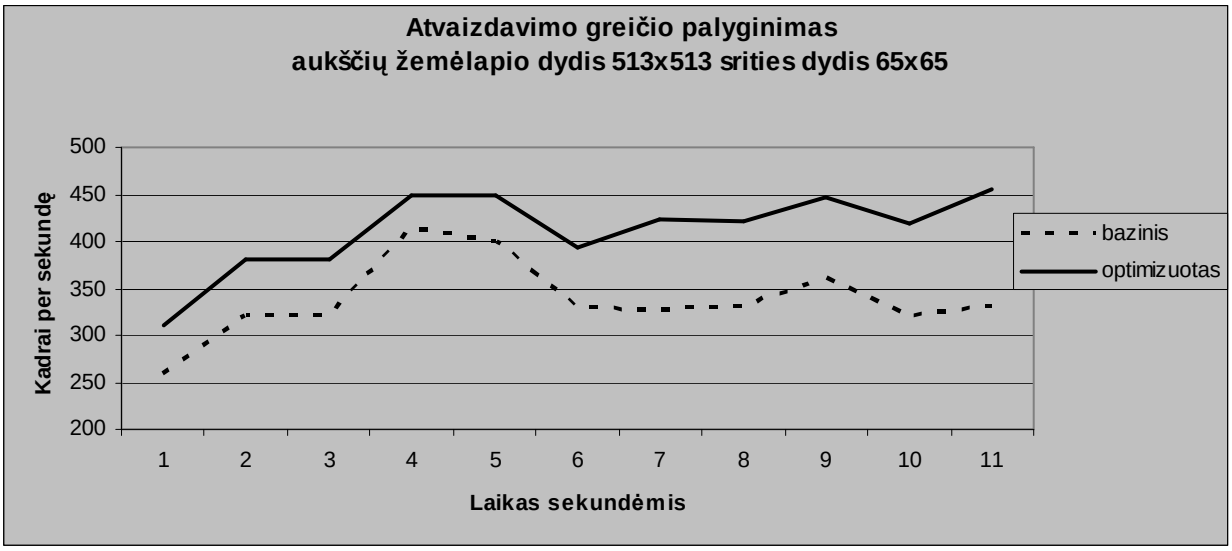
Ši algoritmo optimizacija buvo kuriama pagreitinti (kalbant pavyzdžiais) arti skardžių esančios jūros dalies atvaizdavimą. Vėliau praplėsta ir didelių jūros plotų atvaizdavimo greitinimui. Iš tiesų gana daug scenų galima sureguliuoti (neįtakojant jų galutinio vaizdo) taip, kad būtų išnaudojama ši optimizacija. Pavyzdžiui, gilaus ežero dugnas (į kurį negalima nunerti) gali būti plokščias ir nulinio aukščio, lygumos plotelis kalnuotoje vietovėje taip pat dažnai gali būti kaip tik žemiausioje kraštovaizdžio vietoje ir t.t.

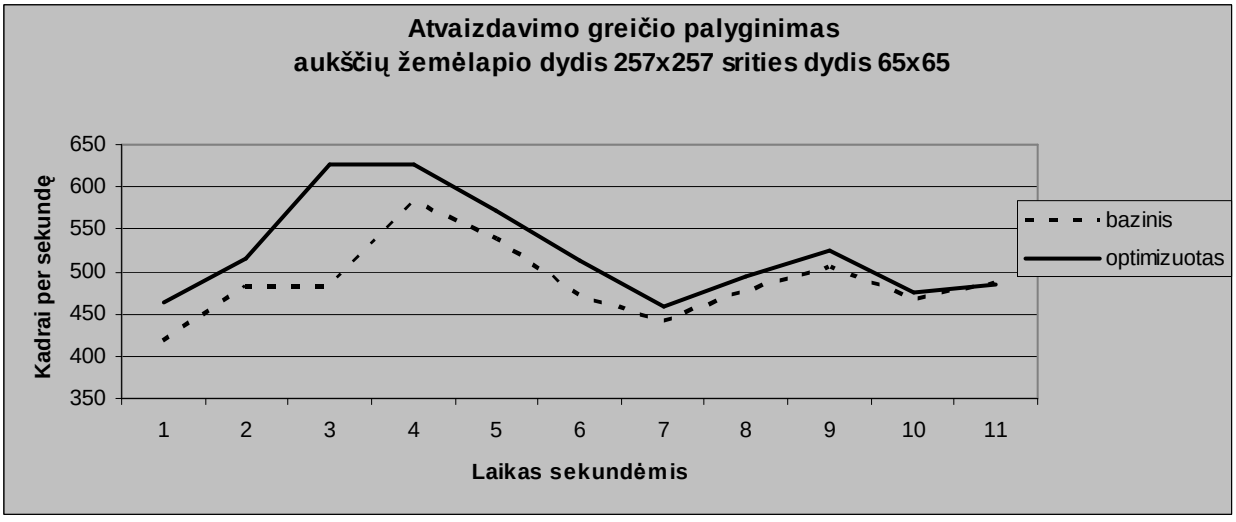
Šią optimizaciją taip pat galima pritaikyti ir ne nulinio aukščio h plokštumoms, t.y. jei nagrinėjamoje srityje nėra nė vienos viršūnės kurios aukštis mažesnis už h, tai trikampiai, kurių visų viršūnių aukštis patenka į intervalą $[h; h + \varepsilon]$, gali būti naikinami, o vėliau uždedama papildoma plokštuma tai sričiai aukštyje h. Kad neatsirastų tarpų, programos paleidimo metu reiktų visoms viršūnėms kurių aukštis patenka į intervalą $[h; h + \varepsilon]$, priskirti aukštį h. Visgi tai papildomas kreipimasis į vaizdo plokštę kiekvienai optimizuojamai sričiai, todėl toks pritaikymas duotų naudos tik jei tenkinama bent viena iš sąlygų:

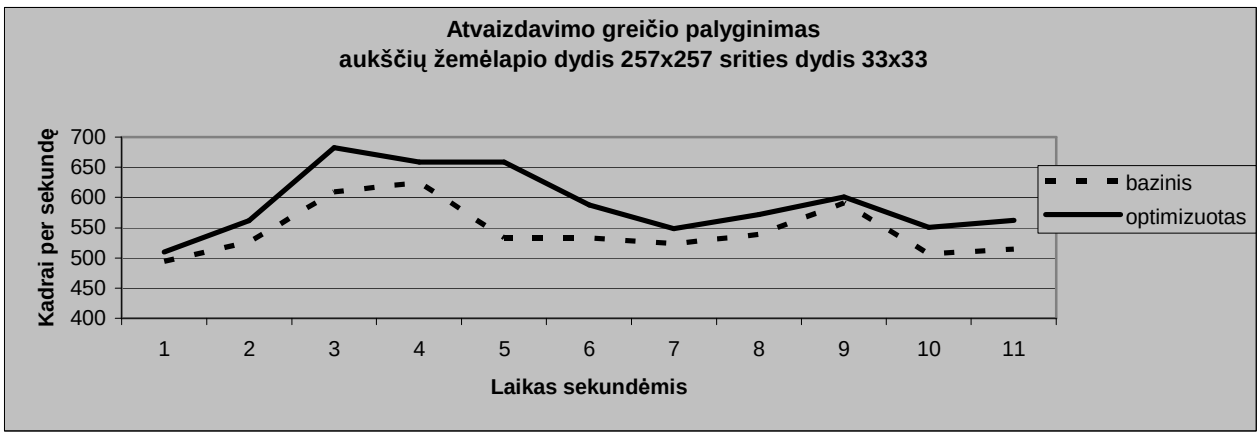
- naudojamos pakankamai didelės sritys (šiai dienai 65x65 arba didesnės);
- randamos kelios tokios sritys ir joms visoms naudojama viena papildoma plokštuma;
- optimizacija taikoma tik pakankamai aukšto detalumo sritims (pvz. tik pačioms detaliausioms).

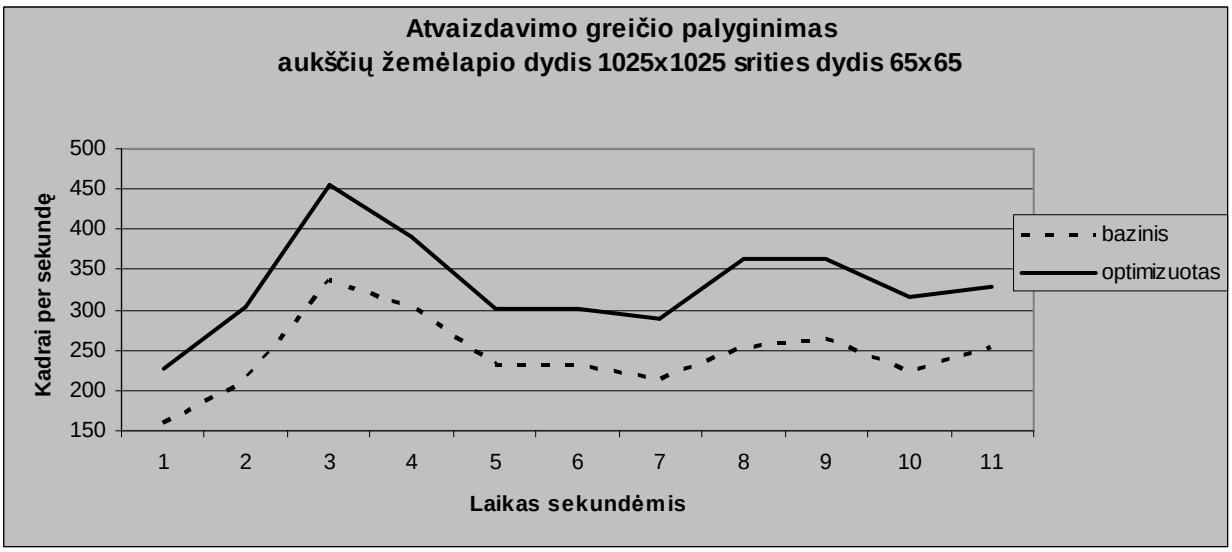
3.6. Matavimai

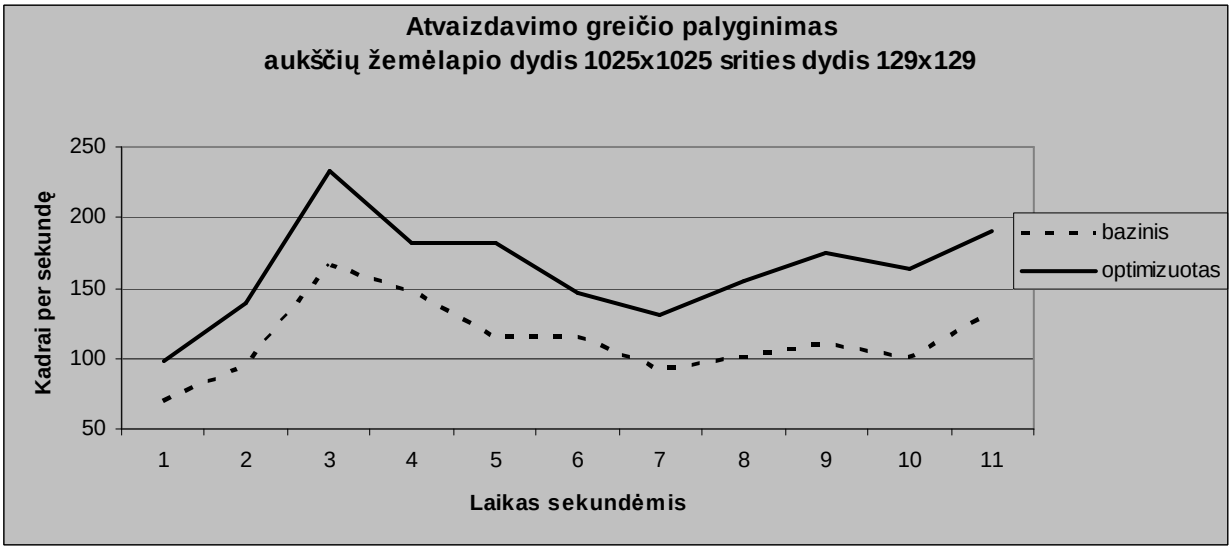
Turimoms scenoms buvo suformuotas automatinis kameros maršrutas, parinktas taip kad bet kuriuo metu apimtų nemažą kraštovaizdžio dalį, ir kas sekundę matuojamas atvaizdavimo greitis. Atlikta keletas matavimų skirtingo detalumo kraštovaizdžiams ir parenkant skirtingo dydžio sritis. Naudota leistina paklaida $\tau = 3$. Toliau pateikiama keletas grafikų iliustruojančių atvaizdavimo greičio kitimą bėgant laikui (t.y. judant stebėjimo kamerai)











Grafikuose aiškiai matosi bendra tendencija – optimizuotas algoritmas visada lenkia bazinį. Remiantis atliktais matavimais, paskaičiuota kad optimizuotas algoritmas lenkia bazinį vidutiniškai 22%. Žinoma, jokie matavimai negali objektyviai pasakyti kiek ši optimizacija pagerina algoritmą, kadangi ji nėra skirta bendram atvejui, t.y. viskas priklauso nuo scenos. Vis dėl to, užfiksuotas vidutinis 22% pagreitėjimas realioje scenoje yra įspūdingas skaičius.

Išvados

- Apžvelgti pagrindiniai kompiuterinės grafikos spartinimo metodai ir išmokta naudotis jų realizacijomis OGRE grafiniame varikliuke.
- OGRE pagalba buvo išanalizuotas ir modifikuotas sudėtingas kintamo detalumo kraštovaizdžio atvaizdavimo algoritmas, reikalaujantis plataus kompiuterinės grafikos konteksto supratimo ir realizavimo.
- OGRE geomipmap realizacijoje rasta ir paminėta keletas svarbių algoritmo patobulinimų, kurie nebuvo pastebėti nagrinėtoje literatūroje.
- Optimizuotas atskiras kintamo detalumo kraštovaizdžio atvejis, t.y. nevaizduojami trikampiai kurių visos trys viršūnės patenka į apatinę kraštovaizdžio ribą, taip pat nevaizduojamos sritys sudarytos vien iš tokių trikampių.
- Parodyta, kad gana daug kraštovaizdžių galima suvesti į tokią formą, kad minėta optimizacija aiškiai paspartintų jų atvaizdavimą.
- Nagrinėtose scenose, naudojant realizuotą patobulinimą užfiksuotas vidutiniškai 22% greitesnis atvaizdavimas lyginant su bazine OGRE geomipmap algoritmo realizacija.

Literatūros šaltinių sąrašas

[AM00] U. Assarsson, T. Moller. Optimized View Frustum Culling Algorithms for Bounding Boxes, 2000. [žiūrėta 2008-06-02]. Prieiga per Internetą: <http://www.ce.chalmers.se/~uffe/vfc_bbox.pdf>

[AM99] U. Assarsson, T. Moller. Optimized View Frustum Culling Algorithms, 1999. [žiūrėta 2008-06-02]. Prieiga per Internetą: <http://www.ce.chalmers.se/~uffe/vfc.pdf>

[Boe00] W. H. de Boer. Fast Terrain Rendering Using Geometrical MipMapping. Prieiga per Internetą: <http://www.flipcode.com/archives/article_geomipmaps.pdf>

[Bre05] N. Brettell. Terrain Rendering Using Geometry Clipmaps, 2005. [žiūrėta 2009-03-02]. Prieiga per Internetą: <http://www.cosc.canterbury.ac.nz/research/reports/HonsReps/2005/hons_0502.pdf>

[Cla76] J. H. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. Communications of the ACM, 19(10), 1976 p.547-554.

[EHS05] A. Efremov, V. Havran, H. P. Seidel. Robust and numerically stable Bézier clipping method for ray tracing NURBS surfaces. Spring Conference on Computer Graphics, 2005, p.127-135.

[GH] G. Gribb, K. Hartmann. Fast Extraction of Viewing Frustum Planes from the World-View-Projection Matrix. [žiūrėta 2008-06-02]. Prieiga per Internetą: <http://www2.ravensoft.com/users/ggribb/plane%20extraction.pdf>

[GH98] G. Greiner, K. Hormann. Efficient clipping of arbitrary polygons. ACM Transactions on Graphics, 17(2), 1998, p.71-83.

[Hop96] H. Hoppe. Progressive Meshes, 1996, SIGGRAPH. [žiūrėta 2008-06-02]. Prieiga per Internetą: <ftp://ftp.research.microsoft.com/pub/tech-reports/Winter95-96/TR-96-02.body.ps>

[LB83] Y. D. Liang, B. A. Barsky. An analysis and algorithm for polygon clipping. Communications of the ACM, 26(11), 1983, p.868-877.

[Len07] A. Lenkevičius. Grafika ir vizualizacija, 2007.

[LH04] F. Losasso, H. Hoppe. Geometry clipmaps: Terrain rendering using nested regular grids. ACM Transactions on Graphics, SIGGRAPH 2004, 23(3), 769-776. Prieiga per Internetą: <http://research.microsoft.com/en-us/um/people/hoppe/geomclipmap.pdf>

[LRC+02] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. Level of Detail for 3D Graphics, 2002.

[Mai92] P. G. Maillot. A new, fast method for 2D polygon clipping: analysis and software implementation. ACM Transactions on Graphics, 11(3), 1992, p.276-290.

[NLN87] T. M. Nicholl, D. T. Lee, R. A. Nicholl. An efficient new algorithm for 2-D line clipping: Its development and anglysis. International Conference on Computer Graphics and Interactive Techniques, 1987, p.253-262.

[OKS03] M. Olano, B. Kuehne, M. Simmons. Automatic Shader Level of Detail, 2003. [žiūrėta 2008-06-02]. Prieiga per Internetą: <http://www.cs.unc.edu/~olano/papers/aslod.pdf>

[Puk77] R. F. Puk. General clipping on an oblique viewing frustrum. International Conference on Computer Graphics and Interactive Techniques, 1977, p.229-235.

[RHS+98] S. Rottger, W. Heidrich, P. Slusallek, H. P. Seidel. Real-Time Generation of Continuous Levels of Detail for Height Fields, 1998. [žiūrėta 2008-06-02]. Prieiga per Internetą: <http://www.cs.ubc.ca/~heidrich/Papers/WSCG.98.pdf>

[San03] D. Sánchez-Crespo Dalmau. 3D Pipeline Overview. Core Techniques and Algorithms in Game Programming, 2003, p.282-307.

[SH74] I. E. Sutherland, G. W. Hodgman. Reentrant polygon clipping. Communications of the ACM, 17(1), 1974, p.32-42.

[SJ02] D. Sýkora, J. Jelínek. Efficient View Frustum Culling, 2002. [žiūrėta 2008-06-02]. Prieiga per Internetą: <http://www.cescg.org/CESCG-2002/DSykoraJJelinek/>

[Tre04] C. Tremblay. Mathematics for Game Developers, 2004, p.230-240.

[Ulr02] T. Ulrich. Rendering Massive Terrains using Chunked Level of Detail Control, 2002. [žiūrėta 2009-03-02]. Prieiga per Internetą: <http://tulrich.com/geekstuff/sig-notes.pdf>

[Vat92] B. R. Vatti. A generic solution to polygon clipping. Communications of the ACM, 35(7), 1982, p.56-63.

[WL04] R. S. Wright, B. Lipchak. It's All About the Pipeline: Faster Geometry Throughput. OpenGL® SuperBible, Third Editon, 2004.

[ZS02] M. Zhang, C. L. Sabharwal. An efficient implementation of parametric line and polygon clipping algorithm. Symposium on Applied Computing, 2002, p.796-800.

Sąvokų apibrėžimai

Aukščių žemėlapis	Dvimatis paveikslėlis, kuriame trečias matmuo (aukštis) atvaizduojamas paveikslėlio elemento (pixel) šviesumu, tokiu būdu jame saugoma trimatė geometrija. Tokiu būdu neįmanoma atvaizduoti olų, iškyšulių, persidengimų.
Detalumo adaptacija	Tam pačiam objektui (ar objekto daliai) naudojama skirtingo detalumo geometrija, priklausomai nuo jos matomumo.
Erdvės padalijimas	Objektų ar objekto dalių saugojimas specialiose struktūrose, kad pagreitinti išrinkimą.
Geoclipmap	Vienas iš greičiausių algoritmų kraštovaizdžių atvaizdavimui. Beveik visi skaičiavimai atliekami vaizdo plokštėje.
Geomipmap	Vienas iš greičiausių algoritmų kraštovaizdžių atvaizdavimui. Dalina kraštovaizdį į sritis ir kiekvienai parenka reikalingą detalumo lygį
Indeksų buferis	Struktūra viršūnių saugomų viršūnių buferyje vaizdavimo tvarkai nurodyti. Originaliai skirta sumažinti viršūnių buferį, t.y. kad nereiktų dubliuoti viršūnių.
Išrinkimas	Objektų, nepatenkančių į vaizdavimo erdvę, atmetimas iš tolimesnių skaičiavimų.
Kintamas detalumas	Žr. detalumo adaptacija.
Kraštovaizdis	Vietovė, teritorija (angl. terrain); šiame darbe turima omeny bet kokia geometrija, kurią galima atvaizduoti aukščių žemėlapiu
Viršūnių buferis	Struktūra geometrijai laikyti, priimtina vaizdo plokštei, skirta pasiekti greitesnį atvaizdavimą.

Priedai

1. Parašytos programos jų išeities tekstai ir šio dokumento el. variantas