KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
MULTIMEDIJOS INŽINERIJOS KATEDRA

MINDAUGAS MATONIS

GEOMETRINIŲ OBJEKTŲ TRIANGULIAVIMO METODAI

MAGISTRO DARBAS

Darbo vadovas doc. A. Riškus

KAUNAS, 2006

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
MULTIMEDIJOS INŽINERIJOS KATEDRA

MINDAUGAS MATONIS

GEOMETRINIŲ OBJEKTŲ TRIANGULIAVIMO METODAI

MAGISTRO DARBAS

Kalbos konsultantė .....
Lietuvių k. katedros lekt.
I. Mickienė
2006 m. gegužės .....d.

Vadovas .....
dr. A. Riškus
2005 m. gegužės .....d.

Recenzentas .....
dr. E. Karčiauskas
2006 m. gegužės .....d.

Atliko .....
IFM 0-1 gr. stud.
Mindaugas Matonis
2006 m. gegužės 25 d.

KAUNAS, 2006

TURINYS

1. PRATARMĖ

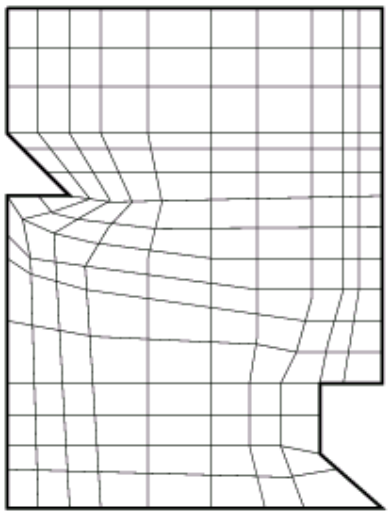
Geometrinio objekto skaidymas į trikampius yra vadinamas trianguliacija. Trianguliacijos rezultatas yra trikampių elementų tinklas, kuris yra labai plačiai naudojamas įvairiose srityse. Kompiuterinėje grafikoje atlikus tokį 3D objekto diskretizavimą, žymiai sumažėja informacijos, reikalingos aprašyti objektą kiekis ir supaprastinami būsimieji skaičiavimai. Geografijoje trikampių tinklas kompaktiškai ir tiksliai atvaizduoja paviršių. Baigtinių elementų metode nagrinėjamas objektas taip pat yra suskaidomas į mažesnius paprastos formos(dažniausiai trikampius) elementus.

Dėl labai plačios trianguliavimo pritaikymo ir naudojimo srities, labai svarbu kurti naujus bei tobulinti senus algoritmus. Šiame darbe aprašyta Delaunay trianguliacija, įvairus jos generavimo algoritmai, bei taikymo būdai. Sukurti ir optimizuoti godaus įterpimo Delaunay ir DataDep algoritmai, atliktas jų tyrimas pateikti rezultatai ir išvados.

2. ĮVADAS

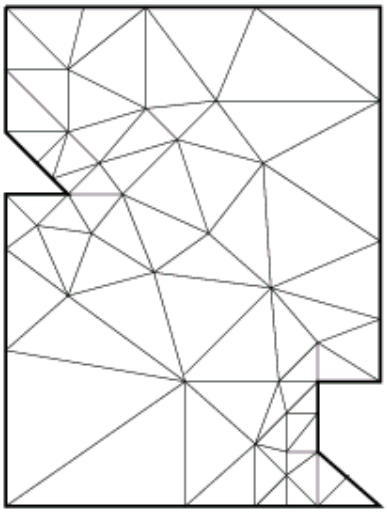
2.1. Geometrinio tinklo tipai

Yra išskiriami trys pagrindiniai geometrinio tinklo tipai: struktūrizuotas, nestruktūrizuotas ir hibridinis (struktūrizuotas blokais) [1].



1 pav. Struktūrizuotas tinklas

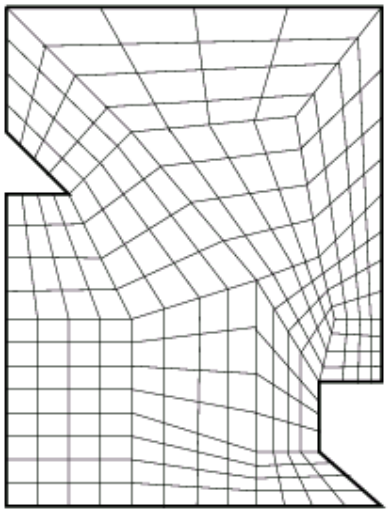
Struktūrizuotas tinklas (*1 paveikslas*) yra toks tinklas, kurio visi elementai yra panašūs topologiškai.



2 pav. Netruktūrizuotas tinklas

Nestruktūrizuoto tinklo (*2 paveikslas*) elementai gali vienas nuo kito skirtis. Atskiras nestruktūrizuoto tinklo atvejis yra anisotropinis tinklas.

Hibridinis tinklas (*3 paveikslas*) yra formuojamas iš mažesnių struktūrizuotų dalių, kurios yra sujungiamos į bendrą tinklą pagal struktūros neturintį šabloną.



3 pav. Hibridinis tinklas

Struktūrizuoti tinklai yra sąlyginai paprasti ir efektyvūs. Jiems aprašyti reikia mažiau atminties, lengva ieškoti kaimyninių elementų – užtenka didinti arba mažinti masyvo, aprašančio tinklą, elementus. Tačiau sukurti struktūrizuotą tinklą sudėtingos geometrinės formos sritims gali būti labai sudėtinga, o kartais net ir neįmanoma. Struktūrizuoti elementai negali keisti dydžio taip greitai kaip reikėtų, todėl kartais generuojant tinklą gali prireikti daug daugiau struktūrizuotų elementų lyginant su nestruktūrizuotais, nes nestruktūrizuoto tinklo elementai tokio trūkumo neturi. Hibridiniai tinklai turi tiek struktūrizuotų, tiek nestruktūrizuotų tinklų pranašumų, tačiau juos labai sudėtinga generuoti.

Struktūrizuotus tinklus paprastai sudaro keturkampiai, o nestruktūrizuotus trikampiai elementai.

Dėl savo universalumo ir lankstumo nestruktūrizuoti tinklai yra daug plačiau naudojami nei struktūrizuoti ar hibridiniai tinklai, todėl ir buvo pasirinkta tokia darbo tema.

Šiame darbe bus aptariami ir nagrinėjami algoritmai ir metodai, kurie naudojami sudaryti nestruktūrizuotam tinklui iš trikampių elementų, dar vadinamam trianguliacija.

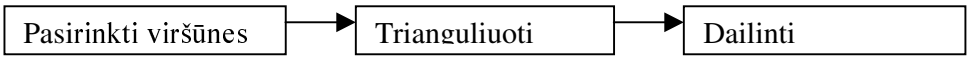
2.2. Trianguliavimo algoritmai

Trianguliavimo algoritmai yra skirstomi į keturias pagrindines grupes [3]:

- Trianguliacija pasirenkant viršūnes,
- Inkrementinė trianguliacija,
- Tinklo generavimas trianguliuojant pabaigoje,
- Tinklo generavimas retrianguliuojant.

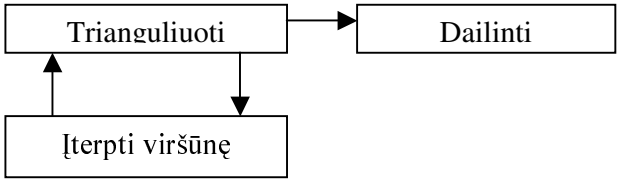
Metodai priklausantys pirmajai grupei parenka viršūnes ir, neįterpdami naujų viršūnių, jas trianguliuoja ir pasirinktinai pakoreguoja viršūnių pozicijas, naudojant kurią nors dailinimo

priemonę (*4 paveikslas*) [16, 20]. Tokio metodo trūkumas yra toks, kad netinkamai pasirinkus pradines viršūnes gaunamos prastos kokybės tinklas.



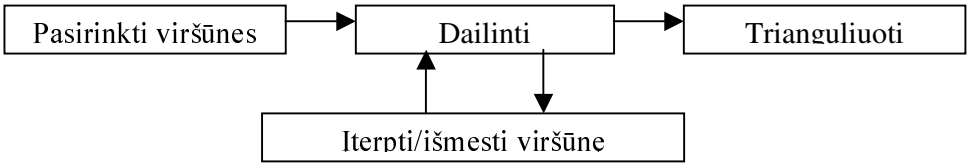
4 pav. Trianguliacijos pasirenkant viršūnes struktūra

Inkrementinės trianguliacijos metodai (*5 paveikslas*) įterpia naują viršūnę ir atnaujina trianguliaciją [4, 15, 26]. Progresuojančio priekio (advancing front) metodas yra tokio trianguliavimo pavyzdys [8, 21]. Netoli nuo srities ribos yra generuojami aukštos kokybės trikampiai, tačiau tose vietose, kur frontai susikerta, elementai gali būti deformuojami. Po to atliktas dailinimas gali išspręsti šią problemą.



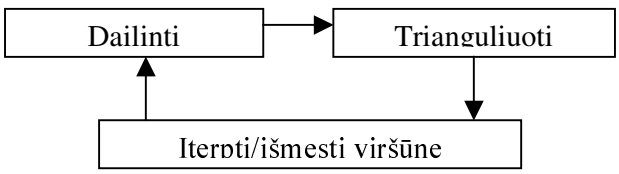
5 pav. Inkrementinės trianguliacijos struktūra

Tinklo generavimo trianguliuojant pabaigoje metodas (*6 paveikslas*) įterpia/išmeta viršūnę ir atlieka dailinimą vienu metu. Tai leidžia keisti tinklo elementų kiekį, kuomet viršūnių kiekis yra per didelis arba per mažas. Shimada ir Gossard sukūrė algoritmą, kuris parenka pradines viršūnes, įterpia/įšmeta viršūnes ir dailina, o pabaigoje atlieka Delaunay trianguliaciją [27].



6 pav. Tinklo generavimo trianguliuojant pabaigoje metodo struktūra

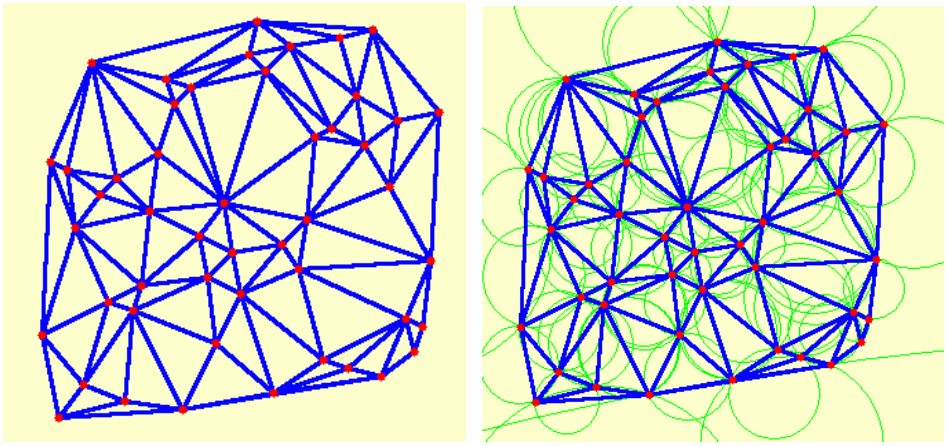
Retrianguliavimo metodas (*7 paveikslas*) įterpia/išmeta viršūnę, dailina ir trianguliuoja vienu metu. Welch sukūrė algoritmą, kuris naudoja Laplaso dailinimą ir Delaunay trianguliaciją [29].



7 pav. Retrianguliavimo metodo struktūra

Darbe daugiausia dėmesio skirta Delaunay trianguliacijai. 3 skyriuje aprašyti Delaunay trianguliacijos generavimo algoritmai, jos taikymas anisotropinio tinklo kūrimui. 4 skyriuje optimizuoti ir išnagrinėti godaus įterpimo Delaunay ir godaus įterpimo DataDep algoritmai. 5 skyriuje pateikti rezultatai, o darbas baigiamas išvadomis.

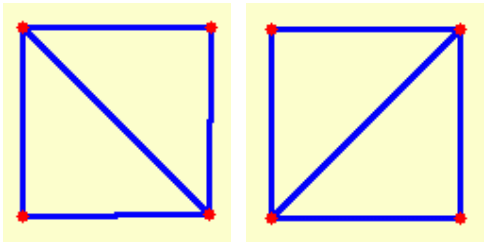
3. DELAUNAY TRIANGULIACIJA



8-9 pav. Plokštumos taškų Delaunay trianguliacija

Delaunay trianguliacija vadinama toks plokštumos taškų M_i trejetų sujungimas trikampiais (8-9 paveikslai), kad nubrėžtame per bet kurio trikampio viršūnes apskritimo viduje nėra nei vieno taško M_i.

Delaunay trianguliacija padalina plokštumos sritį į trikampius. Delaunay trianguliacijos sąvoką galima apibendrinti ir daugiamačiu atveju. Pavyzdžiui, trimačiai M_i taškai yra talpinami į tokių tetraedrų viršūnes, kad nubrėžtos per bet kurio teatraedro viršūnes sferos viduje nėra nei vieno taško M_i.



10-11 pav. Nevienareikšmė Delaunay trianguliacija

Jei tarp M_i taškų atsiras keturi ir daugiau taškų, per kuriuos galima nubrėžti apskritimą, kurio viduje nėra kitų taškų, tai tokių taškų aibei Delaunay trianguliaciją galima atlikti nevienareikšmiškai. Paprasčiausią nevienareikšmės Delaunay trianguliacijos atvejį, kai duoti trianguliacijai keturi taškai yra plokštumos stačiakampio viršūnės, iliustruoja (*10-11 paveikslai*). Nevienareikšmė Delaunay trianguliacija galima tik tokiu "išsigimusių" duomenų atveju ir todėl praktiškai ji pasitaiko retai. "Išsigimusių" duomenų atveju galima nežymiai pakeisti daugiau nei trijų vienam apskritimui priklausančių taškų koordinates taip, kad modifikuotiems duomenims būtų gauta vienareikšmė Delaunay trianguliacija.

Delaunay trianguliacija (DT) yra taikoma:

- Kartografijoje, miestų planavime,
- Planuojant mobiliųjų telefonų retransliacijos stočių išdėstymą,
- Efektyviai artimiausiojo kaimyno paieškai (atpažinimas, kompiuterinė rega),
- Aproksimuojant paviršius (kompiuterinė geometrija, atpažinimas),
- Formuojant gardeles, baigtinių elementų metode (diferencialinės lygtys),
- Vizualizacijoje (kompiuterinė rega, geometrija),
- Daugiamačių duomenų struktūrose (kompiuterija).

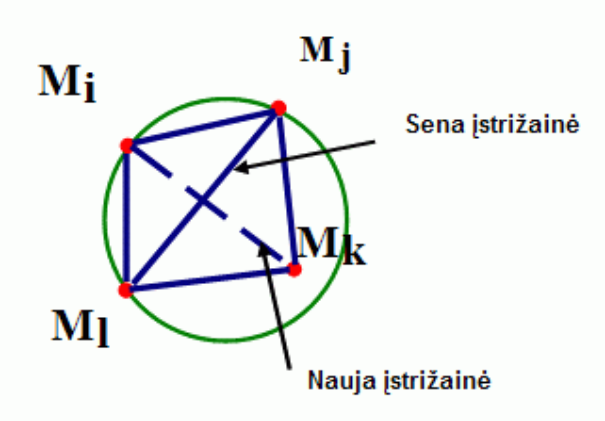
3.1. Algoritmai

Bet kokios plokštumos taškų aibei galima atlikti Delaunay trianguliaciją. Galimas toks tiesioginis M_0, M_1, ..., M_{N-1} taškų Delaunay trianguliacijos algoritmas.

```
for (int i=0; i<N-2; i++)
    for (int j=i+1; j<N-1; j++)
      for (int k=j+1; k<N; k++){
            // Rasti apskritimą kertantį taškus Mi, Mj, Mk
            for (int l=0; l<N; l++){
            // Tikrinti ar Ml taškas nėra apskritimo viduje
            // Jei visi Ml taškai nėra apskritimo viduje
            // papildyti trianguliaciją (Mi,Mj,Mk) trikampiu
            }
      }
```

Šio tiesioginio DT algoritmo sudėtingumas yra $O(N^4)$. Trumpai apibūdinsime keletą efektyvesnių DT algoritmų.

- *Keitimo* algoritmas [28] pradedamas nuo bet kokios trianguliacijos, kuri yra nuosekliai konvertuojama į Delaunay trianguliaciją.



12 pav. Keturkampio istrižainės sukeitimas

Keturkampio $[M_i,M_j,M_k,M_l]$, sudaryto iš dviejų gretimų trianguliacijos trikampių $[M_i,M_j,M_l]$ ir $[M_j,M_k,M_l]$, įstrižainė $[M_j,M_l]$ keičiama į $[M_i,M_k]$, jei viršūnė M_k yra apskritimo, kertančio trikampio $[M_i,M_j,M_l]$ viršūnes, viduje (*12 paveikslas*). Tokiu būdu ($[M_i,M_j,M_l]$, $[M_j,M_k,M_l]$) trikampių pora keičiama į ($[M_i,M_j,M_k]$,$[M_k,M_l,M_i]$) porą.

- Atsitiktinio įterpimo algoritmas [17] trianguliaciją pradeda nuo trijų apjungtų trikampiu taškų pasirenkant atsitiktinai sekantį tašką, kuris prijungiamas prie trianguliacijos. Ši procedūra kartojama kol nelieka taškų. Trianguliacijos papildymas atliekamas panaudojant *ketvirtainį*, *aštuntainį* arba *Delaunay* medį.
- Nuoseklaus trianguliacijos papildymo algoritmas papildo dalinę trianguliaciją nauju trikampiu. Delaunay trikampių sparti paieška atliekama panaudojant reguliarią gardelę [10] ir išretintą matricą [11].
- "Skaldyk ir valdyk" algoritmas [6] rekursyviai atlieka į dvi dalis dalinamas taškų Delaunay trianguliacijas. Yra sugalvoti efektyvūs trianguliuotų dalių apjungimo algoritmai.
- "Šluojančios tiesės" (angl. *sweepline*) yra vienas žinomiausių ir efektyviausių DT algoritmų [12]. Algoritmo pradžioje plokštumos taškai yra sunumeruojami taškų abscisių didėjimo tvarka. Trianguliacija atliekama nuosekliai papildant taškais su didėjančiomis abscisėmis. Vizualiai tokį trianguliacijos taškų papildymą galima vaizduoti iš kairės į dešinę judančia vertikalia tiese, iš kur ir kilo metodo pavadinimas. Algoritmo sudėtingumas yra O(Nlog(N)). Čia N - trianguliuojamų plokštumos taškų skaičius.
- "Projekcinis algoritmas" remiasi tokia geometrijos teorema.

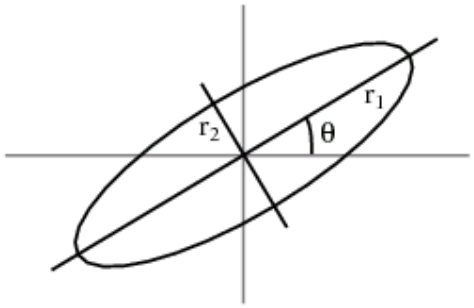
Teorema. Plokštumos taškų Delaunay trianguliacija yra projekcija į x-y plokštumą apatinės iškilos trimatės srities, gaunamos surandant erdvės taškų $(x_i,y_i,(x_i)^2)+(y_i)^2)$ iškilą apvalkalą.

Tiesioginė šio algoritmo realizacija yra $O(N^4)$ sudėtingumo. Tiesioginio algoritmo schema būtų tokia.

```
for ( i=0; i<N; i++) z_i = (x_i)^2 + (y_i)^2;
for ( i=0; i<N-2; i++){
   for ( j=0; j<N-1; j++){
        for ( k=0; k<N; k++){
                Apskaičiuoti trikampio normalę
                Surandame apatinius trikampius
                for ( m=0; m<N; m++){
              jei taškas m yra virš trikampio apibrėžiamo indeksais (i,j,k),
                  tai trikampis lieka Delaunay trianguliacijos kandidato
              priešingu atveju baigti tikrinimą, nes (i,j,k) indeksais
                          apibrėžiamas trikampis nepriklauso Delaunay trianguliacijai.
           }
        }
   }
}
```

3.2. Delaunay trianguliacijos taikymas anisotropinio tinklo generavimui plokštumoje

Norint sukurti anisotropinį (nevienodą visomis kryptimis) tinklą naudojamas 2x2 išmatavimų tenzorius M(x), kuris nustato tinklo elemento dydį naudodamas elemento pozicijos funkciją [3]. Elemento dydis priklauso nuo vietos, kurioje jis yra.



13 pav. Anisotropiją aprašanti elipsė

Anisotropijai aprašyti yra reikalingi trys parametrai: spinduliai r_1, r_2 ir kampas θ (*13 paveikslas*). Šiuos parametrus atitinkantis tenzorius yra:

$$M = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1/r_1^2 & 0 \\ 0 & 1/r_2^2 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

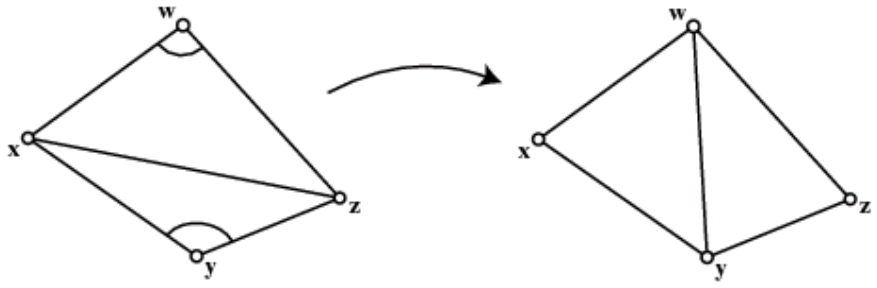
Naudojant šį tenzorių atstumas tarp taškų x ir y yra skaičiuojamas pagal formulę:

$d(x,y) = \sqrt{(x-y)^T M_{avg} (x-y)}$ kur $M_{avg}=(M(x)+M(y))/2$

Tinkle, kuris sukurtas remiantis tokiomis taisyklėmis, viršūnės *i* kaimynai yra arti elipsės $d(p_i, y) = 1$

Normalizuotas trikampio, apibrėžto taškais x, y, z, plotas yra skaičiuojamas pagal formulę:

$$A(x,y,z) = \frac{1}{2}\sqrt{\det(M_{avg})}(y-x)*(z-x)$$ kur $M_{avg}=(M(x)+M(y)+M(z))/3$



14 pav. Briaunos sukeitimas vykdant anisotropinę Delaunay trianguliaciją

Modifikuojame Delaunay kriterijų, kad jis atitiktų anisotropijos reikalavimus. Trikampiai xyz ir zwx yra greta vienas kito (*14 paveikslas*). Delaunay trianguliacija sukeičia bendrą trikampių briauną siekiant maksimizuoti minimalų kampą. Atliekant anisotropinę Delaunay trianguliaciją, taisyklę, kuomet reikia pakeisti briauną xz briauna yw, galima užrašyti:

$$[(z-y)*(x-y)](x-w)^T M_{avg}(z-w)+(z-y)^T M_{avg}(x-y)[(x-w)*(z-w)]<0$$

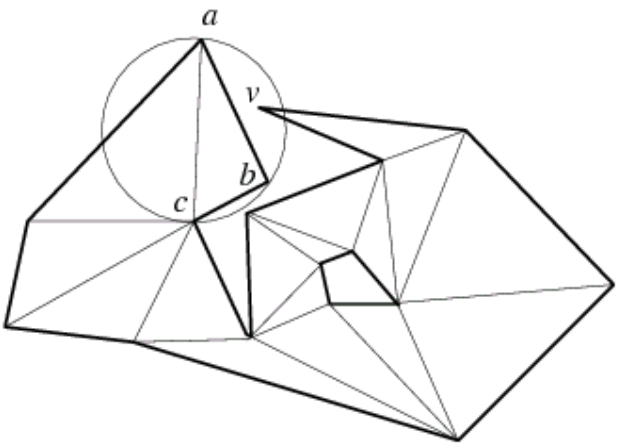
Ši taisyklė yra ekvivalentiška standartiniam apskritimo testui [2]. Įterpiant viršūnę M_{avg}=(M(w)+m(x)+M(y)+M(z))/4 kur w, x, y, z yra tikrinamo keturkampio viršūnės, o naikinant i viršūnę M_{avg}=M(p_i).

Kuomet tenzorius nekinta, vykdant briaunų sukeitimą randamas globalus optimumas [20]. Kuomet tenzorius kinta, atsiranda galimybė, kad briaunų sukeitimas niekada nebus baigiamas. To galima išvengti naudojant tą patį tenzorių visiems briaunų sukeitimo testams.

3.3. Dirbtinė Delaunay trianguliacija

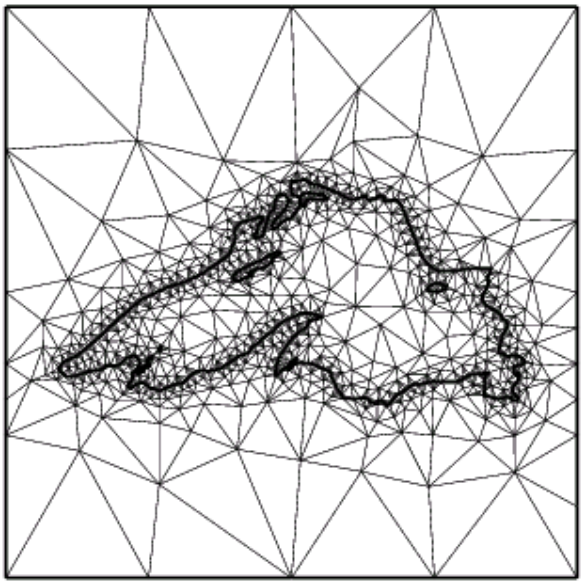
Be įprastos Delaunay trianguliacijos dar yra naudojama dirbtinė (priverstinė) Delaunay trianguliacija. Tokia trianguliacija paprastai naudojama kuomet yra nagrinėjamos sudėtingos geometrinės formos sritys [1].

Taškas *p* yra taško *q* matymo lauke, jei linija *pq* yra srityje W ir nekerta jos ribų. Dirbtinė Delaunay trianguliacija bus toks srities W suskirstymas į trikampius, kad nei vienas trikampis nekerta W ribos ir, atlikus apskritimo testą, nėra nei vienos viršūnės esančios apskritimo lauke. Pateiktas pavyzdys (*15 paveikslas*), kuriame viršūnė *v* nėra nei vieno trikampio abc viršūnės matymo lauke, nors ir atitinka standartinį Delaunay kriterijų (apskritimo testą).



15 pav. Dirbtinei Delaunay trianguliacijai netinkamas taškas

Ruppert [25], remdamasis Chew atliktais darbais [5], pasiūlė dirbtinės Delaunay trianguliacijos algoritmą. Šis algoritmas sugeneruoja pradinę trianguliaciją ir vėliau ją papildo naujomis viršūnėmis, kurios pagerina trianguliacijos kokybę (*16 paveikslas*).



16 pav. Tinklas sugeneruotas Ruppert algoritmu

Algoritmo žingsniai:

- Surasti visas viršūnes, kurių kampas yra mažesnis už 45˚.
- Naudojant surastas viršūnes, sukurti lygiašonius trikampius, kurių pagrindas yra priešais rastą viršūnę, o šonai lygus pusei atstumo nuo viršūnės iki artimiausios kaimynės esančios viršūnės lauke.
- Pakeistai sričiai sugeneruoti dirbtinę Delaunay trianguliaciją.
- Papildyti esamą trianguliaciją viršūnėmis vykdant ciklą:

```
while yra toks trikampis t kurio kampas yra mažesnis už 20˚ do
    apskaičiuojamas trikampio t apskritimo centras c
    if c yra apskritimo, kurio diametras lygus išorinės briaunos e ilgiui, o centras
    yra briaunos e viduryje, viduje then
        įterpti naują viršūnę į briaunos e vidurį
    else įterpti naują viršūnę taške c
    perskaičiuoti trianguliaciją
end
```

Šis algoritmas garantuoja, kad trianguliacijoje nebus trikampių su mažesniais nei 20˚ kampais. Ši taisyklė negalioja tik tiems trikampiams, kurie buvo sugeneruoti iš viršūnių, atrinktų pirmame algoritmo žingsnyje.

4. GODUS ALGORITMAI PAVIRŠIAUS TRIANGULIAVIMUI

Darbo tikslas yra sukurti efektyvius objekto aproksimavimo į tinklą algoritmus, atliekant paviršiaus suskirstymą į trikampius. Šie algoritmai minimizuoja aproksimavimo paklaidą ir bendrą aproksimacijos trikampių skaičių. Išnagrinėti trianguliavimo metodai priklauso „godaus įterpimo“ algoritmų šeimai.

Tobulinimo algoritmas, kuris įterpia bent po vieną viršūnę kiekvienos iteracijos metu, vadinamas godžiu. Egzistuoja daugybė godaus algoritmo modifikacijų [9, 13, 14, 22, 23, 24].

Nagrinėjamu atveju paviršius yra aprašomas dviejų kintamųjų funkcija H(x,y). Duomenys yra pateikiami, kaip taškų plokštumoje ir papildomų duomenų, susietų su šiais taškais rinkinys. Darbo tikslas yra rasti H(x,y) aproksimaciją S(x,y), kuri aproksimuoja H kaip įmanoma tiksliau ir naudoja kaip galima mažiau taškų, o skaičiavimai atliekami kaip galima greičiau. Duomenų taškų skaičių pažymimas n, o aproksimacijos viršūnių skaičių pažymimas m.

Klaida apskaičiuojama iš pradinės taško reikšmės atimant aproksimuoto taško reikšmę H(x,y)-S(x,y).

4.1. Paprastas algoritmas

Paprastas algoritmas yra nuoseklus godaus įterpimo algoritmas [9, 14, 24]. Jis yra paprastas ir neoptimizuotas. Pradinė dviejų trikampių trianguliacija yra sukuriama naudojant kraštinius H taškus, vėliau atliekama taško su didžiausia klaida paieška ir taškas yra įterpiamas į trianguliaciją.

Mesh Insert(Point p): įterpia viršūnę p į Delaunay tinklą

Mesh Locate(Point p): suranda trikampį, kuriame yra taškas p

Insert(Point p):
Viršūnę p padaryti neaktyvia
Mesh Insert(p)

Error(Point p): apskaičiuoja ir gražina klaidos reikšmę taške p

Skaičiavimai yra baigiami, kai pasiekiamas nustatytas tikslas (įterptų viršūnių skaičius, trianguliacijos maksimali paklaida ir pan.). Tikslo ir skaičiavimų pabaigos sąlygos yra saugomos funkcijoje Goal Met.

```
Greedy Insert 1():
Atlikti pradinę dviejų trikampių trianguliaciją
while not Goal Met() do
        best := nil
        maxerr := 0
        visiems aktyviems taškams p do
                err := Error(p)
                if err > maxerr then
                        maxerr := err
                        best := p
        Insert(best)
```

Ciklo vykdymo metu atliekami šie skaičiavimai: taško su didžiausia klaida radimas, viršūnės įterpimas į tinklą, klaidų perskaičiavimas tinklo taškuose.

L pažymimas laikas, kurio reikia norint surasti tašką Delaunay tinkle ir *I* laikas, kurio reikia norint įterpti viršūnę į tinklą, o *i* yra vykdomo ciklo numeris.

Taško su didžiausia paklaida suradimui naudojamas tikrinimo metodas, kuris atlieka $O(n)$ palyginimus. Atliekamas viršūnės įterpimas į tinklą *I*. Kiekvienam taškui, kuris dar nėra įterptas į trianguliaciją, reikia atlikti perskaičiavimą, kuris užtrunka $O(nL)$. Blogiausiu atveju taško suradimas užtruks $O(n)$, įterpimas $O(i)$, perskaičiavimas $O(in)$. Bendra skaičiavimų trukmė $O(m^2n)$. Tai yra labai didelis skaičiavimų kiekis, todėl algoritmą reikia tobulinti.

4.2. Greičiau perskaičiuojantis algoritmas

Tobulinant algoritmą pirmiausia atkreipiamas dėmesys į tai, kad vykdant Delaunay trianguliaciją ir įterpiant viršūnę, pokyčiai įvyksta tik nedidelėje srityje. Po Delaunay įterpimo, briaunos iš įterptos viršūnės eis į jį supančio poligono kampus. Šis poligonas ir apibrėžia sritį, kurioje trianguliacija buvo pakeista. Šį poligoną galima pavadinti atnaujinimo sritimi, nes pakeitimai buvo atlikti tik jame. Tai įgalina atlikti pirmą reikšmingą patobulinimą: neperskaičiuoti nekeistų taškų klaidų vertės ir perskaičiuoti klaidas tik atnaujinimo srityje. Masyvas *Cache* saugo visų pradinių taškų klaidos įverčius, kuriuos apskaičiuoja *Error*. Trianguliacijos papildymo metu yra patikrinami visi įterptą viršūnę supantys trikampiai ir perskaičiuojama jų klaida.

Modifikuotas algoritmas:

Insert(Point p):
Viršūnę p padaryti neaktyvia
Mesh Insert(p)
taškams q esantiems trikampių, supančių viršūnę p, viduje do
Cache[q] := *Error*(q)

Greedy Insert 2():
Atlikti pradinę dviejų trikampių trianguliaciją
aktyviems taškams p do
    Cache[p] := *Error*(p)
while not *Goal Met*() do
    best := nil
    maxerr := 0
    aktyviems taškams p do
        err := Cache[p]
        if err > maxerr then
            maxerr := err
            best := p
Insert(best)

Šis algoritmas, kaip ir ankstesnis algoritmas, atlieka tuos pačius skaičiavimus: taško radimas, įterpimas ir perskaičiavimas. Laikas sugaištamas taško suradimui ir įterpimui liko toks pats, tačiau beveik visada yra atliekama mažiau perskaičiavimo operacijų.

Taško radimą galima atlikti greičiau surikiuojant visus taškus pagal klaidos dydį. Kiekvienas trikampis gali turėti tik vieną tašką, labiausiai tinkantį įterpimui, tačiau jei trikampio

viduje nėra taškų jis visai neturi taškų, kurie galėtų būti įterpti. Visus taškus išrikiavus pagal klaidos dydį, nereikia ieškoti geriausio kandidato įterpimui, nes jis paprasčiausiai paimamas iš rikiuotos eilės (*heap*) pradžios [7].

Klaidos perskaičiavimą galima pagreitinti iš anksto kiekvienam taškui nustačius ir užfiksavus, kuriam trikampiui jis priklauso arba kiekvienam trikampiui paskaičiavus plokštumos, kurioje jis yra, lygtį.

Algoritmas naudoja šias pradines duomenų struktūras: taškų masyvas, plokštumų aibė, trianguliacija ir surikiuota taškų eilė.

Taškų masyvo struktūra susideda iš stačiakampio masyvo, kuriame kiekvienas narys saugo aukštį H(x,y) ir požymį, ar taškas jau buvo naudojamas trianguliacijai. Plokštumų struktūra saugo plokštumos lygties $z = ax + by + c$ koeficientus a, b, c.

4.3. Optimizuotas algoritmas

Algoritmas, atlikus minėtus pakeitimus:

Heap Change(Point h, Float key, Triangle T): priskiria taškui h klaidą key ir trikampį T, papildo tašku h taškų eilę heap

```
if h ≠ nil then
        if key > 0 then
                Heap Update(h, key) papildo esamą tašką naujais duomenimis
        else
                Heap Delete(h) pašalina nebereikalingą tašką
                return nil
else
        if key > 0 then
                return Heap Insert(key, T) įterpia naują tašką
return h
```

```
Scan Triangle(Triangle T):
        plane := Find Triangle Plane(T)
        best := nil
        maxerr := 0
        taškams p esantiems trikampyje T do
                err := Error(p)
                if err > maxerr then
                        maxerr := err
                        best := p
        T.heapptr := Heap Change(T.heapptr, maxerr, T)
        T.candpos := best
```

```
Mesh Insert(Point p, Triangle T):
```

```
Insert(Point p, Triangle T):
        mark p as used
        Mesh Insert(p, T)
        trikampiams U kurie gretimi taškui p do
                Scan Triangle(U)
```

Greedy Insert 3():
Atlikti pradinę dviejų trikampių trianguliaciją
    pradiniams trikampiams T do
        Scan Triangle(T)
    while not Goal Met() do
        T := Heap Delete Max()
        Insert(T.candpos, T)

Vykdant šį algoritmą ieškojimo metu vykdomi trys veiksmai: įterpimas į eilę, suradimas eilėje ir eilės duomenų atnaujinimas. Eilės narių padidėjimas kiekvienos iteracijos metu yra 2. Tačiau šis masyvas ne visada auga taip greitai. Kuomet trikampiai tampa nedideli arba taip gerai aproksimuoja paviršių, kad net neturi potencialių įterpimo taškų, jie yra pašalinami iš masyvo ir todėl masyvas sumažėja. Dažniausiai trianguliacijos, kurias reikia pasiekti, yra daug mažesnės nei pradiniai duomenys, todėl algoritmas beveik niekada neturės naudos iš sumažėjusios taškų eilės, nes įterptų viršūnių kiekis beveik visada yra daug kartų mažesnis už neįterptų taškų kiekį. Galima daryti prielaidą, kad masyvo dydis i iteracijos metu yra O(i) ir atlikti manipuliacijas eilėje reikia O(log i) laiko.

Įterpimas ir perskaičiavimas atliekami greičiau, kadangi nei vienas jų neatlieka vietos paieškos.

1, 2 ir 3 lentelėse yra pateikiamas visų godaus algoritmo modifikacijų sudėtingumas.

1 lentelė

Didžiausias sudėtingumas vykdant i-ąją iteraciją

	Suradimas	Įterpimas	Perskaičiavimas
Greedy insertion 1	O(n)	O(i)	O(in)
Greedy insertion 2	O(n)	O(i)	O(in)
Greedy insertion 3	O(log i)	O(i)	O(n)

2 lentelė

Vidutinis sudėtingumas vykdant i-ąją iteraciją

	Suradimas	Įterpimas	Perskaičiavimas
Greedy insertion 1	O(n)	O(√i)	O(n)
Greedy insertion 2	O(n)	O(√i)	O(n/i)
Greedy insertion 3	O(log i)	O(1)	O(n/i)

3 lentelė

Bendras algoritmų sudėtingumas

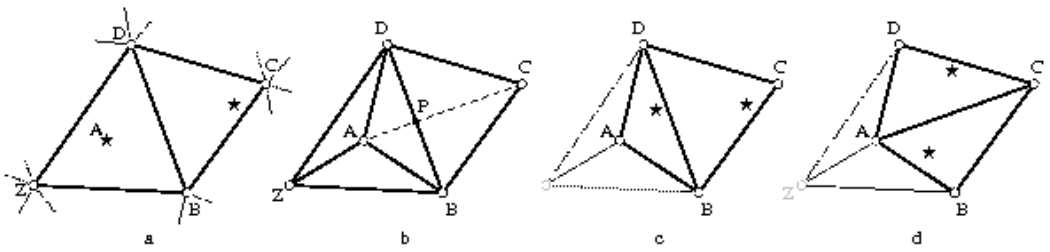
	Blogiausias atvejis	Vidutinis atvejis
Greedy insertion 1	$O(m^2n)$	O(mn)
Greedy insertion 2	$O(m^2n)$	O(mn)
Greedy insertion 3	O(mn)	O((m+n)log m)

4.4. Trianguliacija atsižvelgiant į duomenis

Aptartas Delaunay trianguliacijos metodas, generuodamas tinklą naudoja tik geometrinę informaciją ir dėl to gali nukentėti aproksimacijos kokybė. Tikslesnę aproksimaciją galima sukurti naudojant algoritmą, kuris atsižvelgia į duomenis. Tokiu atveju trikampiai parenkami pagal tai, kaip aproksimuotas paviršius atitinka pradinius duomenis.

Nuo duomenų priklausomos godaus įterpimo algoritmas yra panašus į jau aptartą algoritmą, bet vietoje Delaunay trianguliacijos naudojama duomenų priklausoma trianguliacija [18, 24]. Viršūnė, kurią reikia įterpti kiekvienos iteracijos metu surandama ir pasirenkama taip pat, bet trianguliacija atliekama kitaip.

Delaunay algoritmas testuoja briaunas naudodamas geometrinį apskritimo testą. Duomenų priklausomoje trianguliacijoje vietoj briaunų tinkamumo tikrinimo apskritimo testu, yra taikoma tokia taisyklė: briauna yra sukeičiama, jei sukeitimas sumažina aproksimacijos klaidos dydį.



17 pav. Trianguliacija atsižvelgiant į duomenis

Taškas A turi didžiausią klaidos dydį ir sekanti viršūnė bus įterpta į trianguliaciją. Iš taško A vedamos briaunos į jį supančio poligono viršūnes. Kiekviena gaubiančio poligono briauna yra patikrinama. Kai kuriais atvejais nagrinėjamas keturkampis bus įgaubtas ir todėl galės būti trianguliuojamas tik vienu būdu, bet daugeliu atvejų jis išgaubtas ir todėl visos briaunos turi būti tikrinamos, ar mažina klaidos dydį. Jei briauna BD sumažina klaidą (*17 paveikslas c*), tai naujų briaunų tyrimui nėra ir rekursija yra stabdoma.

Paprasčiausias būdas patikrinti briauną BD yra atlikti rekursinę procedūrą: ištirti abu keturkampio ABCD trianguliacijos variantus. Jei briauna BD sumažina klaidą, tai naujos briaunos netiriamos ir rekursija baigiama. Jei keičiant briauną BD briauna AC padidina klaidos dydį, tada pakeisti briauną AC ir taikyti rekursiją naujoms briaunoms BC, CD, AB ir AD.

Kai visos briaunos yra patikrintos, reikia atnaujinti informaciją apie tinkamiausius taškus trikampiams, esantiems nagrinėtame poligone. Tam reikia patikrinti kaimyninius trikampius du kartus: pirma kartą tikrinant briaunos sukeitimą ir antrą kartą ieškant taškų.

Greitesnė alternatyva yra tikrinti vieną kartą, skaičiuojant klaidą ir tašką tos pačios iteracijos metu.

Šis algoritmas naudoja visas jau aptartas duomenų struktūras ir vieną naują struktūrą *FitPlane*, kuri saugo aproksimacijos plokštumą ir skaičiavimų metu kaupia informaciją apie klaidą ir viršūnę, kurią norima įterpti į trikampį, aproksimuojamą tos plokštumos: plokštumos koeficientus, viršūnės vietą *candpos* ir klaidą *canderr*, trikampio klaidą *err* ir požymį ar trikampis jau buvo tikrintas.

FitPlane yra inicijuojama procedūroje *FitPlane Extract*(*T*). Kuomet yra tikrinamas naujas trikampis, procedūra *FitPlane Init*(*a, b, c*) inicijuos struktūrą *FitPlane* plokštumai, einančiai per taškus a, b, c su klaidos įverčiais, nustatytais 0 ir požymiu done = nil. Toliau kreipiamasi į *Scan Triangle Datadep* ir *FitPlane* kaupia informaciją apie klaidą ir potencialią viršūnę. Informacija apie labiausiai tinkamą viršūnę yra išsaugoma ir vėliau panaudojama procedūroje Set Candidate(T,fit).

Funkcijos Left Triangle ir Right Triangle gražina trikampius, esančius nurodytos briaunos kairėje arba dešinėje pusėje.

```
Set Candidate(var Triangle T, FitPlane fit):
    T.heapptr := Heap Change(T.heapptr, _t.canderr, T)
    T.candpos := fit.candpos
    T.err := fit.err

Scan Point(Point x, Fitplane fit):
    err := Error(x, fit.plane)
    fit.err := Error Accum(fit.err, err)
    if err > fit.err then
        fit.canderr := err
        fit.candpos := x
```

Scan Triangle Datadep(Point p, Point q, Point r, var FitPlane u, var FitPlane v) Tikrinti trikampį pqr, pakeisti klaidos įvertį ir tinkamiausius taškus plokštumoms u ir v

```
    visiems taškams x esantiems trikampio pqr viduje do
    if u ≠ nil and not u.done then
        Scan Point(x, u)
    Scan Point(x, v)
```

First Better(float q1, float q2, float e1, float e2) Grąžinti true, jei 1 briauna tinka labiau nei 2 briauna remiantis formos ir tinkamumo kriterijumi

```
    qratio := Min(q1, q2) / Max(q1, q2)
```

```
if qratio ≤ qthresh then
        return (q1 ≥ q2) taikomas formos kriterijus
else
        return (e1 ≤ e2) taikomas tinkamumo (klaidos) kriterijus
```

Check Swap(DirectedEdge e, FitPlane abd) tikrina briauną e ir ją sukeičia, jei tai sumažina klaidą, atnaujina trianguliaciją ir eilę

```
if abd = nil then
        FitPlane abd := FitPlane Init(a, b, d)
if briauna e yra krašte arba keturšonis abcd yra įgaubtas then
        if not abd.done then
                Scan Triangle Datadep(a, b, d, nil, abd)
        Set Candidate(Left Triangle(e), abd)
else
        FitPlane cdb := FitPlane Extract(Right Triangle(e))
        FitPlane dac := FitPlane Init(d, a, c)
        FitPlane bca := FitPlane Init(b, c, a)
        Scan Triangle Datadep(p, d, a, abd, dac)
        Scan Triangle Datadep(p, a, b, abd, bca)
        Scan Triangle Datadep(p, b, c, cdb, bca)
        Scan Triangle Datadep(p, c, d, cdb, dac)
        ebd Error Combine(abd.err, cdb.err)
        eac Error Combine(dac.err, bca.err)
        if First Better(Shape Quality(a, b, c, d), Shape Quality(b, c, d, a), ebd, eac)
then
                Set Candidate(Left Triangle(e), abd)
                if not cdb.done then
                        Set Candidate(Right Triangle(e), cdb)
        else
                sukeisti briauną e iš bd į ac
                dac.done := true
                bca.done := true
                Check Swap(DirectedEdge cd, dac)
                Check Swap(DirectedEdge bc, bca)
```

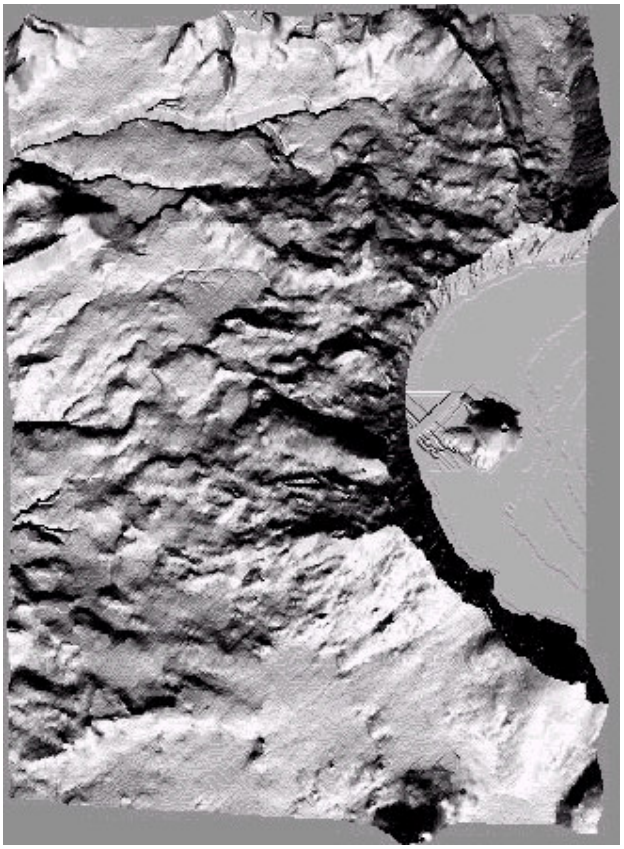
```
Insert Datadep(Point a, Triangle T):
Pažymėti tašką a kaip panaudotą
Nuvesti briaunas iš taško a į jį supančio daugiakampio viršūnes
(T and possibly a neighbor of T)
visoms supančio poligono briaunoms e do
Check Swap(e, nil)

Greedy Insert Datadep():
Atlikti pradinę dviejų trikampių trianguliaciją
e := diagonali briauna
Check Swap(e, nil)
while not Goal Met() do
T := Heap Delete Max()
Insert Datadep(T.candpos, T)
```

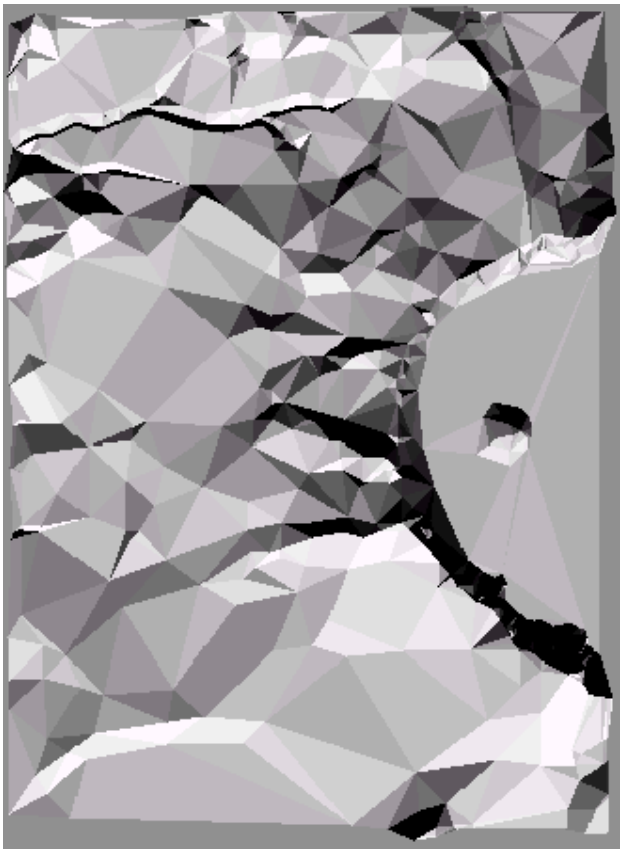
Šis algoritmas yra lėtesnis už Delaunay trianguliacijos metodą, nes atlieka dvigubai daugiau perskaičiavimų. Algoritmo sudėtingumas blogiausiu atveju yra O(mn), vidutinis sudėtingumas O((m+n)log m).

5. REZULTATAI

Pradiniai duomenys pavaizduoti 18 paveiksle. Tai yra stačiakampis duomenų laukas, kurį sudaro 154224 taškų. Pradinio paviršiaus aproksimacija panaudojant 0,5% pradinių taškų (771 viršūnę) parodyta 19 paveiksle. Aproksimacija panaudojant 1% pradinių taškų (1542 viršūnes) parodyta 20 paveiksle. Ši aproksimacija yra daug tikslesnė ir informatyvesnė už pirmąją, bet, lyginant su pradiniais duomenimis, skirtumas yra aiškiai pastebimas.

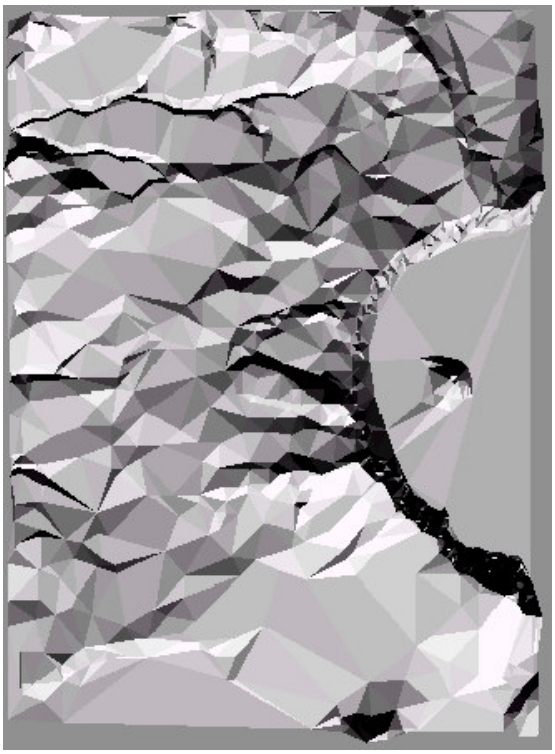


18 pav. Pradiniai duomenys (154224 taškų)

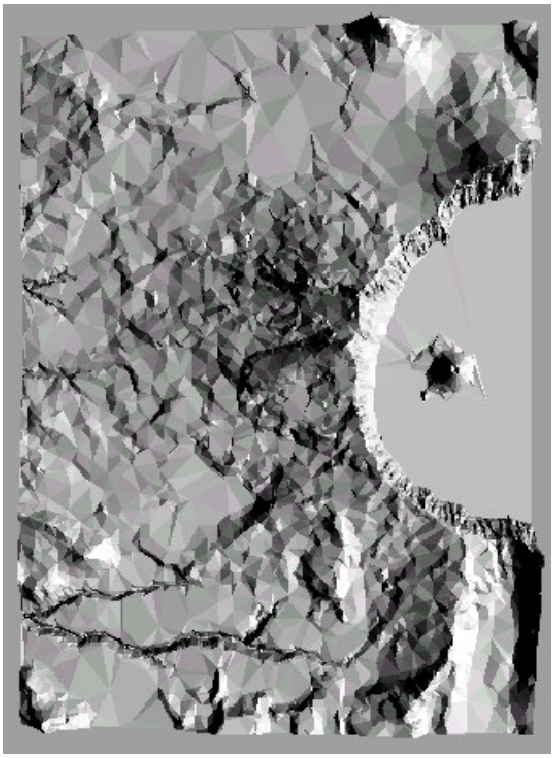


19 pav. Aproksimacija panaudojus 771 viršūnių

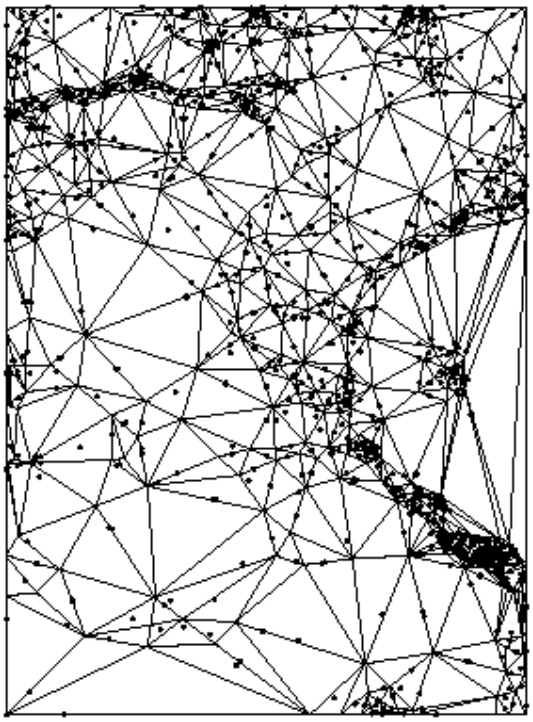
21 paveiksle yra pavaizduota aproksimacija naudojant 5% duomenų (7711 viršūnių). Šis modelis gerai atspindi didžiąją dalį originalo bruožų. Panaudojant nedidelį kiekį pradinių duomenų galima sukurti pakankamai aukšto tikslumo aproksimacijas. Visi šie modeliai yra sukurti naudojant Greedy Insert3 algoritmą.



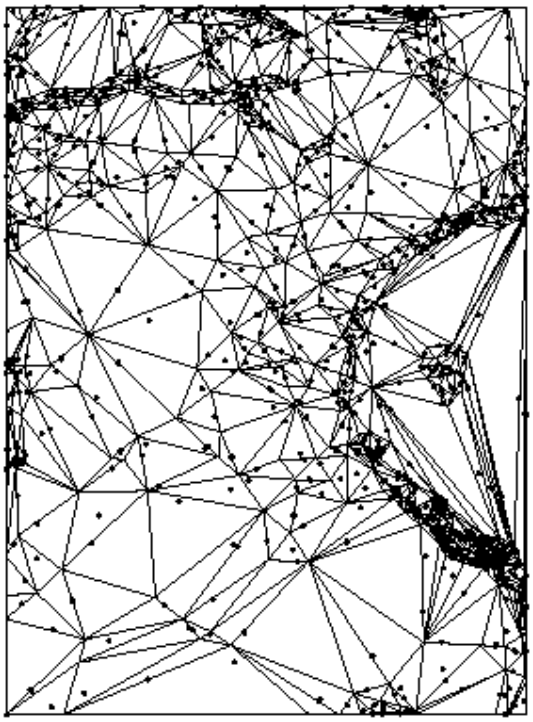
20 pav. Aproksimacija panaudojus 1542 viršūnių



21 pav. Aproksimacija panaudojus 7711 viršūnių



22 pav. Tinklas sugeneruotas Greedy Insert3

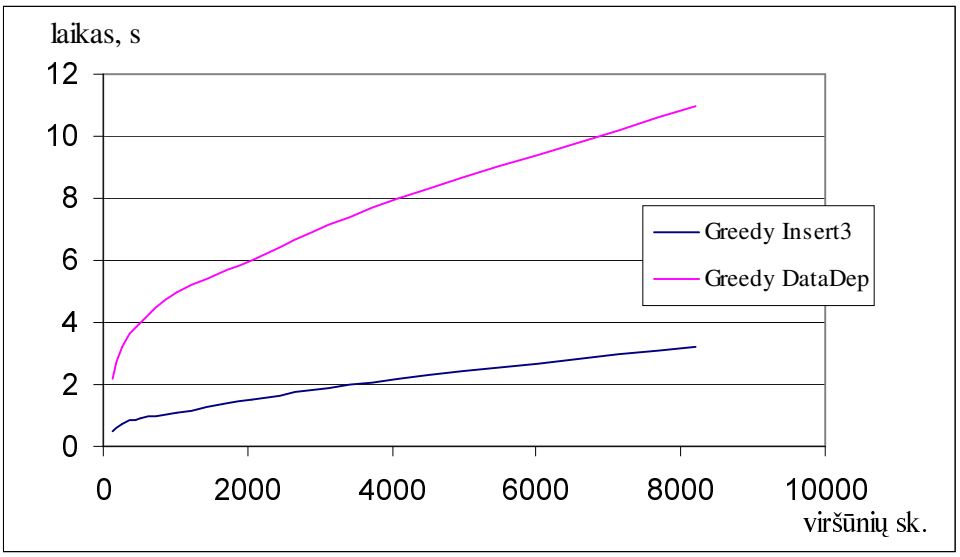


23 pav. Tinklas sugeneruotas Greedy DataDep

22 ir 23 paveiksluose parodytas tinklas, sugeneruotas Greedy Insert3 ir Greedy DataDep algoritmų. Abejais atvejais viršūnių kiekis yra 555. Greedy DataDep algoritmas generuoja daug netinkamos geometrinės formos trikampių – skiedrų (sliver). Taip yra todėl, kad, įterpdamas naujas viršūnes, šis algoritmas nekreipia dėmesio į trikampių formą.

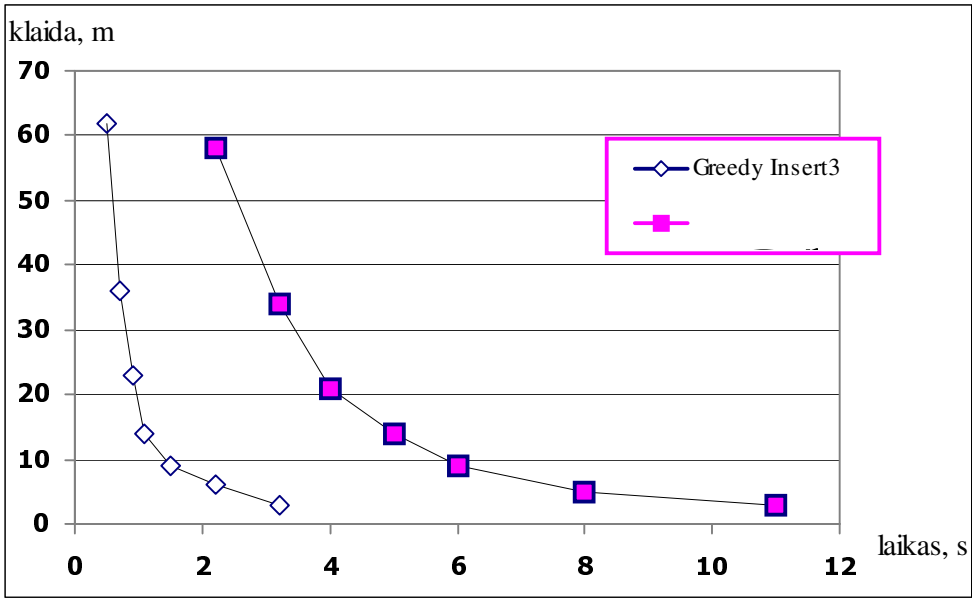
24 pav. Algoritmų generuojamos klaidos palyginimas

Vertinant algoritmus aproksimacijos klaidos kriterijumi (*24 paveikslas*), Greedy Insert3 pralaimi Greedy DataDep algoritmui, bet gerokai lenkia pastarąjį greičiu (*25 paveikslas*).



25 pav. Algoritmų greičio palyginimas

Bendras abiejų algoritmų greičio ir klaidos palyginimas pateiktas grafiku (*26 paveikslas*).



26 pav. Algoritmų greičio ir klaidos palyginimas

6. IŠVADOS

1. Pateiktas Delaunay trianguliacijos apibrėžimas ir taikymo sritys, kai kurių Delaunay trianguliavimo metodų sarašas ir aprašymai. Pristatytas algoritmas anisotropinio trikampio tinklo generavimui naudojant Delaunay trianguliaciją. Aprašytas algoritmas dirbtinei (constrained) Delaunay trianguliacijai sukurti, bei esminiai skirtumai lyginant su standartine Delaunay trianguliacija.
2. Sukurti, išanalizuoti ir ištirti godaus įterpimo trianguliavimo algoritmai. Algoritmai optimizuoti, nustatytas jų sudėtingumo laipsnis, pateikti testų rezultatai ir atliktas palyginimas. Paprastas godus įterpimo algoritmas buvo optimizuotas:
    - atliekant perskaičiavimus tik reikiamose vietose
    - naudojant išrikiuotą masyvą (eilę) surasti taškus su didžiausia klaida
    - eliminuojant taško buvimo vietą.

    Aproksimuojant n taškų tinklą naudojant m viršūnių šie patobulinimai padidino našumą nuo O(mn) iki O((m +n)log m).
3. Palyginti Greedy Insertion ir Greedy DataDep trianguliacijų rezultatai: Greedy DataDep algoritmas generuoja geresnės kokybės aproksimacijas ir turėtų būti taikomas kur yra reikalingos auštos kokybės aproksimacijos. Delaunay algoritmas yra greitesnis vertinant pagal skaičiavimo greitį ir galėtų būti taikomas srityse kur skaičiavimų greitis yra svarbiausias veiksnys. Greedy DataDep algoritmas generuoja daug netinkamos geometrinės formos trikampių – skiedrų, nes vykdo trianguliaciją remdamasis klaidos sumažinimo faktoriumi ir dėl to nukenčia trikampių kokybė.
4. Toliau tobulinant algoritmus reikėtų atkreipti dėmesį, kad algoritmai lėtai veikia skaičiavimų pradžioje, kuomet aproksimacijoje yra nedaug trikampių ir todėl reikia atlikti daug perskaičiavimų. Kaip vieną šios problemos sprendimo būdų, galima pabandyti sukurti sudėtingesnę ir kartu efektyvesnę pradinę trianguliaciją (šiame darbe pradinė trianguliacija sudaryta tik iš dviejų elementų). Taip pat labai svarbu išspresti skiedrų problemą Greedy DataDep algoritme. Galimas sprendimo būdas būtų naudoti tarpinį trianguliavimo kriterijų, kuris turėtų ir Delaunay ir DataDep trianguliavimo savybių.
5. Šiuos metodus taip pat būtų galima pabandyti realizuoti ir kitose srityse: vaizdų suspaudime, grafikoje ir kt.

LITERATŪRA

1. Bern, M.; ir Plassmann P. *Mesh Generation*. Elsevier Science, 1999
2. Bossen, F.J. *Anisotropic mesh generation with particles*. Carnegie Mellon University, 1996.
3. Bossen, F.J.; ir Heckbert, P.S. *A Pliant Method for Anisotropic Mesh Generation.* 5th International Meshing Roundtable, Sandia National Laboratories, 1996, p.63-76.
4. Castro-D'iaz, M.J.; Hecht, F.; ir Mohammadi, B. *New progress in anisotropic grid adaptation for inviscid and viscous flows simulations.* 4th Intl. Meshing Roundtable, 1995.
5. Chew, L.P. *Guaranteed quality triangular meshes*. Cornell University, 1989.
6. Cignoni, P.; Montani, C.; Perego, R; ir Scopigno, R. *Parallel 3D Delaunay Triangulation.* Eurographics 93, 1993.
7. Cormen, T.H.; Leiserson, C.E.; ir Rivest, R.L. *Introduction to Algorithms*. MIT Press, Cambridge, 1990.
8. David, L.M. *Generation of unstructured grids for viscous flow applications*. 33rd AIAA Aerospace Sciences Mtg., Reno, 1995.
9. De Floriani, L.; Falcidieno, B.; ir Pienovi, C. *A Delaunay-based method for surface approximation*. Eurographics 83, Elsevier Science, 1983.
10. Fang, T.; ir Piegl, L. *Algorithm for Delaunay triangulation and convex hull computation using a sparse matrix*. Computer Aided Design Vol.24(8), 1992.
11. Fang, T.; ir Piegl, L. *Delaunay triangulation using a uniform grid*. IEEE Computer Graphics and Applications, 1993.
12. Fortune, S. *Sweepline Algorithms for Voronoi diagrams*. Algorithmica 2, 1987.
13. Fowler, R.J.; ir Little, J.J. *Automatic extraction of irregular network digital terrain models*. Computer Graphics, 13(2), 1979.
14. Franklin, W.R. *C code* 1993, ftp://ftp.cs.rpi.edu/pub/franklin/tin.tar.gz
15. Frey, W.H. *Selective refinement: a new strategy for automatic node placement in graded triangular meshes*. IJNME, 1987.
16. Guibas, L.; ir Stolfi, J. *Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams*. ACM Trans, 1985.
17. Guibas, L.J.; Knuth, D.E.; ir Sharir, M. *Randomized Incremental Construction of Delaunay and Voronoi diagrams*. Algorithmica 7, 1992.
18. Hamann, B.; ir Chen, J.L. *Data point selection for piecewise trilinear approximation.* Computer-Aided Geometric Design, 1994.

19. Heller, M. *Triangulation algorithms for adaptive terrain modeling*. Spatial Data Handling, 1990.
20. Lawson, C.L. *Software for C^1 surface interpolation*. Mathematical Software III, Academic Press, 1977.
21. Peraire, J.; ir Peir'o, J. *Adaptive remeshing for threedimensional compressible flow computations*. Computational Physics, 1992
22. Polis, M.F.; ir McKeown, D.M. Jr. *Issues in iterative TIN generation to support large scale simulations*. Minneapolis, 1993.
23. Puppo, E.; Davis, L.; DeMenthon, D.; ir Teng, Y.A. *Parallel terrain triangulation*. Geographical Information Systems, 1994.
24. Rippa, S. *Adaptive approximation by piecewise linear polynomials on triangulations of subsets of scattered data*. SIAM 13(5), 1992.
25. Ruppert, J. *A Delaunay refinement algorithm for quality 2dimensional mesh generation*. Algorithms, 1995.
26. Ruppert, J. *A new and simple algorithm for quality 2dimensional mesh generation*. ACMSIAM Symp. Discrete Algorithms, 1993.
27. Shimada, K.; ir Gossard, D.C. *Bubble mesh: Automated triangular meshing of nonmanifold geometry by sphere packing*. Solid Modeling and Appls, 1995.
28. Sibson, R. *Locally equiangular triangulations*. The Computer Journal Vol.2 (3), 1973.
29. Welch, W. *Serious Putty: Topological Design for Variational Curves and Surfaces*. PhD tezė, CS Dept, Carnegie Mellon University, 1995.

PAVEIKSLŲ SĄRAŠAS

Triangulation methods of geometry objects

SUMMARY

Subject of this paper is triangulation of given domain also called as mesh generation. Overview of main mesh types (structured, unstructured and hybrid) is given. Groups of triangulation methods are defined and include collective triangulation, incremental triangulation, pliant mesh generation with post-triangulation and plaint mesh generation with retriangulation. Delaunay triangulation is described in greater detail and variuos Delaunay triangulation algorithms are presented including use of Delaunay triangulation for anisotropic mesh generation and method to generate Constrained Delaunay triangulation.

Greedy insertion Delaunay and data dependent allgorithms are developed for hight fields surface aproximation. Significant improvements are made to these algorithms including faster recalculation, node selection and use of supplementary data sets in order to maximise efficiency of calculations. Main criteria to evaluate developed algorithms is overall error of approximation and speed of calculation. Data dependent algorithm generates better quality mesh (less approximation error), however Delaunay triangulation algorithm is significantly faster. Results and conclusions are presented at the end of paper.