KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
Multimedijos inžinerijos katedra

Benas Šilkaitis

Plokščios figūros vidurio linijos paieška

Magistro baigiamasis projektas

Darbo vadovas:
doc. dr. Armantas Ostreika

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
Multimedijos inžinerijos katedra

Benas Šilkaitis

Plokščios figūros vidurio linijos paieška

Magistro baigiamasis projektas

Recenzentas:

Darbo vadovas:
doc. dr. Armantas Ostreika

Atliko
IFM-5/1 gr. studentas
Benas Šilkaitis

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos fakultetas

(Fakultetas)

Benas Šilkaitis

(Studento vardas, pavardė)

Informatika, M4016N21

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Plokščios figūros vidurio linijos paieška“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 ___ m. _____ ____ d.

Kaunas

Patvirtinu, kad mano **Beno Šilkaičio** baigiamasis projektas tema „Plokščios figūros vidurio linijos paieška“ yra parašytas visiškai savarankiškai, o visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka)	*(parašas)*

TURINYS

TERMINŲ IR SANTRUMPŲ ŽODYNĖLIS

Pikselis (angl. *pixel*) – smulkiausia monitoriuje rodomo vaizdo šviečiančioji sritis.

Vidurio linija (angl. *centerline*) – linija nubrėžta per figūros centrą, sutampanti su figūros simetrijos ašimi, jei figūra simetriška.

Topologinis ploninimas (angl. *topological thinning*) – figūros transformavimo metodas, kai figūros geometrinės formos ypatybės yra išsaugojamos, tačiau prarandama dalis tai figūrai priklausiusių pikselių.

Daugiatrafaretis greitasis žingsniavimas (angl. *multistencil fast marching*) – matematinio metodo pavadinimas, kurio esmė – kelio rinkimasis atsižvelgiant į trumpiausią laiko funkcijos vertę.

Delaunay trianguliacija (angl. *Delaunay triangulation*) – figūros skaidymas į trikampius, naudojant Delaunay metodą.

Ribojantysis stačiakampis (angl. *bounding box*) – figūrą apgaubiantis stačiakampis, nurodantis figūros maksimalias ir minimalias reikšmes x ir y ašyse, dažniausiai tai dvi poros koordinačių.

„Matlab“ celė – (angl. *cell*) – specifinė „Matlab“ duomenų struktūra, panaši į „Matlab“ struktūras („Matlab“ struct), iš šios duomenų struktūros duomenys gaunami pagal indeksą, o ne pagal vardą, kaip kad „Matlab“ struktūrose;

Monochrominis – (angl. *monochrome*) – vienspalvis;

Ribojančių talpų hierarchija (angl. *bounding volume hierarchy*) – tam tikras geometrinių figūrų skirstymas į grupes, kai kiekviena grupei priklausanti figūra yra apribota ribojančiuoju stačiakampiu, o pati grupė patalpinta į gaubiantį stačiakampį, kuris apima visas grupėje esančias geometrines figūras;

Matomumo grafas (angl. *visibility graph*) – grafas, kuris saugo informaciją apie tai, kuri grafo viršūnė gali būti pasiekta iš kurios viršūnės, brėžiant tiesią linija nuo vienos viršūnės iki kitos ir kelyje nesutinkant kliūčių;

Išretintoji matrica (angl. *sparse matrix*) – skaičių matrica, kurios didelė dalis elementų yra lygūs „0“.

Minimalus dengiantis medis (angl. trumpinys - *MST* arba *minimal spanning tree*) – neorientuotas jungus grafas be ciklų su svoriais, kurio viršūnės sujungtos taip, jog medis turėtų mažiausią svorio vertę.

LENTELIŲ SĄRAŠAS

PAVEIKSLIUKŲ SĄRAŠAS

SANTRAUKA

Pastaruoju metu virtualios realybės taikymo, medicininių duomenų apdorojimo, trumpiausių maršrutų sudarymo, naudojant kompiuterines sistemas, mastas smarkiai išaugo. Visi minėti procesai, dažniausiai, vienu ar kitu būdu naudoja figūros vidurio liniją tolimesniems savo skaičiavimams, todėl sparčiau veikiančio algoritmo sukūrimas labai praverstų.

Pagrindinis šio darbo tikslas – sukurti sparčiau už egzistuojančius sprendimus veikiantį algoritmą, skirtą rasti figūros vidurio linijai. Šis darbas plėtoja „Paprastojo dvimatės figūros vidurio linijos radimo algoritmo“ (angl. *A simple centerline extraction approach for 2D polygons*) (šaltinis [1]) idėją, suformuluotą doc. dr. Alekso Riškaus bei doc. dr. Armanto Ostreikos. Išplėtotas algoritmas lyginamas su trimis algoritmais, atstovaujančiais skirtingas figūros vidurio linijos radimo klases (topologinis ploninimas, Voronoi diagramos, atstumo transformacijos), lyginant greitį bei tikslumą.

Benas Šilkaitis. Figūros vidurio linijos paieška. Magistro baigiamasis projektas, vadovas doc. dr. Armantas Ostreika; Kauno technologijos universitetas, Informatikos fakultetas.

Mokslo kryptis ir sritis: Informatika

Reikšminiai žodžiai: *vidurio linija, algoritmo sparta*

Kaunas, 2017. 61 p.

SUMMARY

Nowadays virtual reality applications, computer aided medical records analysis, computer aided travel path generation for robots or planes are becoming more and more prevalent. These fields most of the time use centerline as the basis for the further calculations, that is why the need of the algorithm which could determine the centerline fast has arisen.

The aim of this research was to develop algorithm which could determine the centerline of the object faster than the existing algorithms. The chosen approach extends some basic principles presented in [1] paper, which was written by dr. Aleksas Riškus and dr. Armantas Ostreika, and develops these ideas further. Three existing algorithms, representing different approaches (topological thinning, Voronoi diagrams, distance transformation) on centerline extraction were chosen and compared with the algorithm, which has been developed, measuring both accuracy and speed.

Šilkaitis, Benas. EXTRACTION OF THE CENTERLINE FOR 2D POLYGONS: Master‘s thesis in Informatics, supervisor dr. Armantas Ostreika. The Faculty of Informatics, Kaunas University of Technology.

Research area and field: Informatics

Key words: *centerline, algorithm performance*

Kaunas, 2017. 61 p.

ĮVADAS

Tyrimo objektas ir problematika

Tiriamojo darbo objektas yra plokščios figūros vidurio linijos radimas. Plokščios figūros vidurio linijos nustatymas yra ypač svarbus medicininių ir mokslinių duomenų analizėje, chirurginių operacijų modeliavime, maršrutų planavime, sprendžiant virtualios realybės problemas.

Dokumento struktūra

Skyriuje „Plokščios figūros vidurio linijos radimo algoritmų analizė“ aptariami 3 jau egzistuojantys algoritmai, skirti rasti figūros vidurio linijai, taip pat analizuojamas ir metodas (žr. šaltinį [1]), kuriuo remiantis buvo sukurtas naujasis algoritmas, skyriuje „Projektavimas“ pateikiama išsami informacija apie įgyvendinto algoritmo detales, taip pat nurodomos algoritmo vietos, kurias dar galima tobulinti. Eksperimento aplinka, kiti įrankiai, bei rezultatai, gauti lyginant sukurtą algoritmą su jau egzistuojančiais algoritmais, aptariami skyriuje „Eksperimentinė dalis“.

Tikslas ir uždaviniai

Tiriamojo darbo tikslas – sukurti spartų algoritmą, skirtą rasti figūros vidurio linijai.

Uždaviniai:

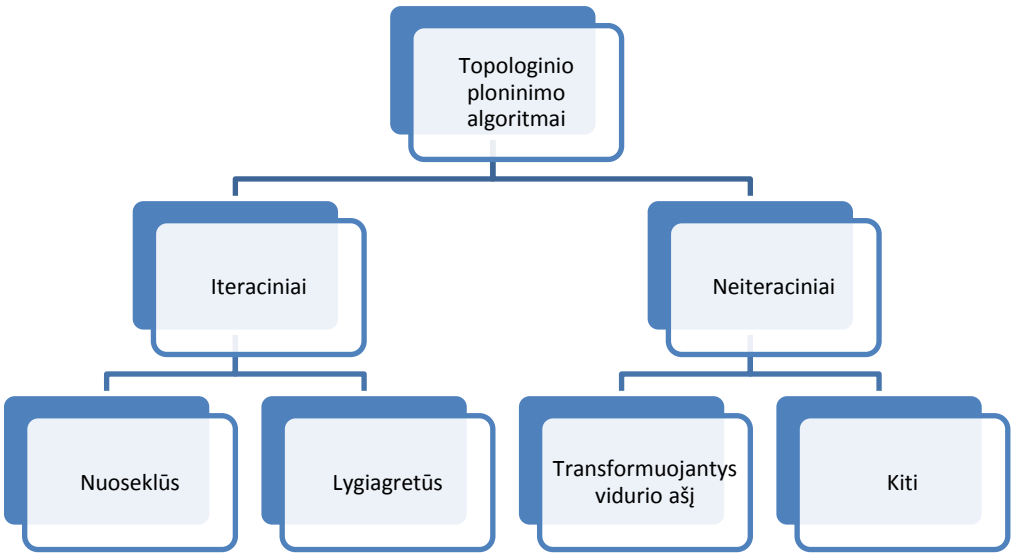
- Atlikti sukurto algoritmo palyginimą su jau egzistuojančiais algoritmais, vertinant veikimo greitį bei tikslumą ir nustatyti su kokio tipo figūromis algoritmas veikia geriausiai;

1. PLOKŠČIOS FIGŪROS VIDURIO LINIJOS RADIMO ALGORITMŲ ANALIZĖ

Šiame skyriuje analizuojami pasirinkti trys algoritmai, skirti rasti figūros vidurio linijai, taip pat analizuojama šaltinyje [1] pateikiamo metodo idėja. Kiekvienas iš pasirinktų algoritmų priklauso skirtingai algoritmų klasei. Pavyzdžiui, Zhang-Suen algoritmas priklauso topologinio ploninimo algoritmų (angl. *topological thinning*) klasei, antrasis nagrinėtas algoritmas priklauso Voronojaus diagramų algoritmų klasei, trečiasis, naudojantis daugiatrafaretį greitąjį žingsniavimą (angl. *multistencil fast marching*)) – atstumų transformacijos metodų klasei. Šie algoritmai skyriuje „Eksperimentinė dalis“ yra lyginami su naujai sukurtu algoritmu, vertinant veikimo spartą bei tikslumą.

1.1. Zhang-Suen algoritmo analizė

Zhang-Suen algoritmas yra vienas iš daugelio topologinio ploninimo algoritmų. Šaltinyje [2] topologinio ploninimo algoritmai buvo suskirstyti į grupes (žr. Pav. 1.1).



Pav. 1.1 Topologinių ploninimo algoritmų klasifikacija

Šaltinyje [2] Zhang-Suen algoritmas yra įvardinamas kaip iteracinis lygiagretus algoritmas. Kaip teigiama [2] iteraciniai algoritmai veikia kiekvienos iteracijos metu šalindami ant figūros sienos esančius pikselius tol, kol bent viena apdorojamo paveikslėlio sritis yra storesnė nei 1 pikselis. Vadinasi, topologinio ploninimo algoritmų spartai didelę įtaką daro figūros storis pikseliais. Taip pat šis algoritmas priskiriamas lygiagrečiųjų algoritmų poaibiui, o tai reiškia, jog neatsižvelgiama į prieš tai buvusių ploninimo iteracijų rezultatus ir kiekvienas pikselis tos pačios

iteracijos metu analizuojamas nepriklausomai – lygiagrečiai. Būtent dėl šios priežasties kai kurioms figūroms vidurio linija gali būti trūki. Algoritmo veikimo metu naudojamas 3x3 langas (žr Pav. 1.2), ir atsižvelgiant į kaimyninių elementų reikšmes, pikselis yra ištrinamas arba ne.

P9 (i-1,j-1)	**P2 (i-1,j)**	**P3 (i-1,j+1)**
P8 (i,j-1)	P1 (i,j)	**P4 (i,j+1)**
P7 (i+1,j-1)	**P6 (i+1,j)**	**P5 (i+1,j+1)**

Pav. 1.2 Pikselio „P1“ 8-nių kaimyninių elementų langas

1.1.1. Algoritmo pseudokodas

Algoritmo pseudo kode (pagal šaltinius [2] ir [3]) (žr. lentelė 1.1), „if“ sąlygoje naudojamas kaimyninių elementų langas (žr. Pav. 1.2). Funkcija `sumuotiAplinkiniųNariųReikšmes(p, i, j)` susumuoja paveikslėlyje „p“ aplink pikselį, kurio koordinatės i ir j, esančių kaimyninių pikselių reikšmes.

lentelė 1.1 Zhang Suen algoritmo pseudokodas

```
Procedure ZhangSuenThinning
// in: p – paveikslėlis (0-lių ir 1-tų matrica)
// out: p – paveikslėlis kuriame likusi tik figūros vidurio linija

// local: paveiksliukoIlgis – apdorojamo paveikslėlio ilgis pikseliais
// local: paveiksliukoAukštis – apdorojamo paveiksliuko aukštis pikseliais
// local: aplinkiniųNariųSuma – aplink p(i,j) pikselį esančių narių suma

begin
  repeat
    for i = 1 : paveiksliukoAukštis
      for j = 1 : paveiksliukoIlgis
        aplinkiniųNariųSuma = sumuotiAplinkiniųNariųReikšmes(p, i, j);
        if aplinkiniųNariųSuma >= 2 && aplinkiniųNariųSuma <= 6 &&
        p(i-1,j)*p(i,j+1)*p(i+1,j) == 0 && p(i-1,j)*p(i,j+1)*p(i,j-1) == 0
        && p(i,j) == 1
          taška p(i,j) pažymėti ištrynimui
        end if
      end for
    end for

    if nėra pažymėtų taškų trynimui
      break;
    else
      visiems pažymėtiems taškams priskirti reikšmę 0
    end if
```

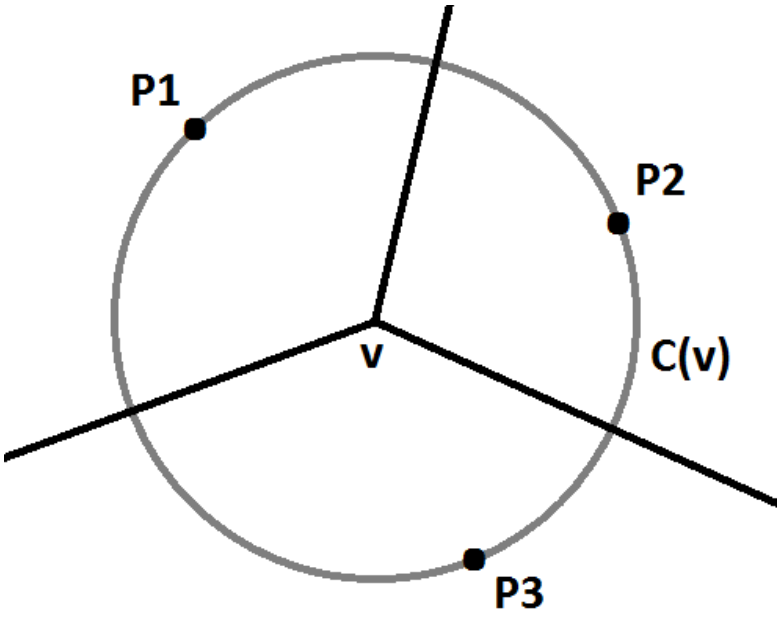
```
    for i = 1 : paveiksliukoAukštis
      for j = 1 : paveiksliukoIlgis
        aplinkiniųNariųSuma = sumuotiAplinkiniųNariųReikšmes(p, i, j);
        if aplinkiniųNariųSuma >= 2 && aplinkiniųNariųSuma <= 6 &&
        p(i-1,j)*p(i,j+1)*p(i,j-1) == 0 && p(i-1,j)*p(i+1,j)*p(i,j-1) ==
        0)    && p(i,j) == 1
          taška p(i,j) pažymėti ištrynimui
        end if
      end for
    end for

    visiems pažymėtiems taškams priskirti reikšmę 0

  until nėra pažymėtų pikselių
end
```

1.1.2. Algoritmo apibendrinimas

Algoritmo įgyvendinimas paprastas, tačiau vidurio linija gali turėti trūkių. Veikimo sparta labai ($O(n \cdot m \cdot s)$, kur n – paveikslėlio plotis, m – aukštis, s – storiausia figūros vieta) priklauso nuo storiausios figūros vietos.

1.2. Algoritmo, naudojančio Voronoi diagramas, analizė

Antrasis algoritmas priklauso algoritmų, naudojančių Voronoi diagramas, klasei. Voronoi diagrama, pagal šaltinį [4] - tam tikras figūros padalinimas į iškilius daugiakampius (žr. Pav. 1.3).



Pav. 1.3 Voronoi diagramos segmentas

Kur Pav. 1.3 „v“ – Voronoi diagramos viršūnė; „P1“, „P2“ ir „P3“ – generuojantys taškai.

Kaip teigiama [4] šaltinyje Voronoi diagrama turi keletą savybių:

- Kiekvienoje Voronoi diagramos viršūnėje kertasi trys briaunos (žr. Pav. 1.3);
- Kiekviena Voronoi diagramos briauna yra bendra tik dviem daugiakampiams (žr. Pav. 1.3);
- Briauna yra statmena linijai, jungiančiai du gretimus generuojančius taškus;
- Kiekviena Voronoi diagramos viršūnė „v“ yra lygiai per tris generuojančius taškus nubrėžto apskritimo centras;
- Voronoi diagramas galima naudoti atliekant figūros „Delaunay“ trianguliaciją (angl. *Delaunay triangulation).*

Remiantis šiomis savybėmis Voronoi diagrama pritaikoma figūros vidurio linijos paieškai.

1.2.1. Algoritmo pseudokodas

Pasirinktas algoritmas iš pradžių išskiria figūros sieną, tuomet apskaičiuoja Voronoi diagramą, atsižvelgiant į figūros sienos taškus. Figūros vidurio linija yra Voronoi viršūnių subgrafas [4], vidurio linijai nepriklausančios grafo viršūnės šalinamos atsižvelgiant į atstumo iki figūros kontūro sąlygą.

lentelė 1.2 Algoritmo, naudojančio Voronoi diagramas, pseudokodas

```
Procedure vidurioLinijaNaudojantVoronoiDiagramas(p)
// in: p – paveikslėlis (0-lių ir 1-tų matrica)
// out: v – vidurio linijos viršūnių matrica
// out: e – vidurio linijos briaunų  matrica
// local: b – figūros kontūras
// local: Dist – atstumų lentelė

begin
  b = bwboundaries(p); // matlab f-ja randanti figūros kontūrą
  [v, e] = konstruotiVoronoiDiagrama(b); // funkcija grąžina Voronoi
                                         //diagramos briaunas bei kraštines
  e = isfiltruotiBlogasBriaunas(e, v); // išfiltruojamos briaunos
                                       // nepriklausančios  paveikslėliui
  Dist = skaičiuotiAtstumusTarpGretimųElementų(b);
  e = išfiltruotiBriaunasPagalAtstumąIkiKontūro(Dist, e); // paliekamos
                        // viršūnės, kurios tenkina atstumo iki
                        // kontūro sąlygą
end
```

lentelė 1.2 pateikiamas pseudokodas atitinka šaltinio [5] algoritmo realizaciją.

1.2.2. Algoritmo apibendrinimas

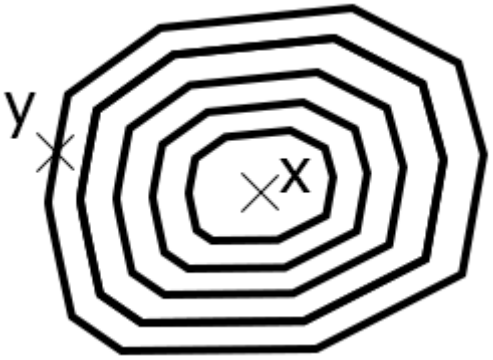
Algoritmo veikimo spartai didesnę įtaką daro figūros sudėtingumas, bei figūros perimetras, o ne figūros storis, kadangi sudėtingesnės figūros Voronoi diagrama - sudėtingesnė. Voronoi diagramos konstravimo sudėtingumas - $O(n \log n + n)$, kur n – figūros kontūrą sudarančių taškų kiekis.

1.3. Atstumų transformacijų klasei priklausančio algoritmo analizė

Šis metodas randa figūros vidurio liniją, skaičiuodamas trumpiausius atstumus nuo apdorojamo pikselių sąrašo iki visų likusių pikselių, naudojant figūros viduje skleidžiamus linijų frontus. Šaltinyje [6] nurodoma, jog šių frontų sekimui naudojamas daugiatrafarečio greitojo žingsniavimo metodas**,** kuris yra praplėsta paprasto greitojo žingsniavimo metodo versija. Šaltinyje [6] bei [7] nurodoma, jog šis metodas apskaičiuoja Eikonal lygtį (žr. formulė 1.1) monotoniškai sklindantiems frontams paveikslėlio viduje (žr. Pav. 1.4).

$$|\nabla u(x)| = \frac{1}{f(x)}, x \in \Omega \quad \textbf{formulė 1.1 Eikonal lygtis}$$

Čia Ω - $\mathbb{R}^n$ (n-matė realių skaičių koordinačių erdvė), ∇ - gradientas, $|.|$ – normalė, $f(x)$ – greičio funkcijos reikšmė taške x, kurios reikšmių sritis yra teigiamų skaičių aibė, $u(x)$ – trumpiausias laikas, norint nukeliauti nuo figūros krašto iki x.



Pav. 1.4 Monotoniškai figūros viduje sklindantys frontai

Čia „x“ – taškas, labiausiai nutolęs nuo visų figūros sienų, „y“ – labiausiai nuo taško „x“ nutolęs taškas esantis ant figūros krašto linijos. Naudojant, daugiatrafaretį greitojo žingsniavimo metodą, bei Rungės-Kutos algoritmą galima apskaičiuoti vidurio liniją, jungiančią šiuos du taškus.

1.3.1. Algoritmo pseudokodas

„lentelė 1.3“ pateikiamo algoritmo pseudokode funkcijos „apskaičiuotiAtstumųIkiSienosMatricą“, bei „pritaikytiGreitojoŽingsMetodą“ skaičiuodamos pikselių atstumus iki sienos naudoja daugiatrafaretį greitojo žingsniavimo metodą, funkcija „apskaičiuotiTrumpiausiąLiniją“ naudoja Rungės-Kutos „RK4“ metodą trumpiausio kelio iki tolimiausio taško ant sienos radimui.

lentelė 1.3 Daugiatrafarečio greitojo žingsniavimo algoritmo pseudokodas

```
procedure  greitojoŽingsniavimoMetodas(p)
// in: p – paveikslėlis (0-lių ir 1-tų matrica)
// out: midLn – vidurio linijai priklausančių taškų matricų sąrašas
(kiekvienai vidurio linijos šakai – po taškų matricą)
// local: distMt – double tipo matrica, kurioje saugojamas kiekvienam figūros
pikseliui surastas artimiausias atstumas iki figūros sienos
// local: srcPt – šaltinio taškų matrica (pradžioje pridedamas labiausiai
nutolęs figūros taškas)
// local: maxDst – labiausiai nutolusio nuo sienos taško atstumas
// local: speedMt – gauta greičių matrica kiekvienam pikseliui
// local: T – atstumų matrica nuo šaltinio taško iki visų likusių piselių
// local: Y – atstumų matrica greitojo žingsniavimo metodui, kurioje saugomos
atstumų nuo šaltinio taškų iki likusių pikselių reikšmės Euklido erdvėje.
// local: strtPt – taškas ant figūros sienos
// local: shortestLn – trumpiausia linija tarp strtPt ir šaltinio taško srcPt
// local: lnLength – trumpiausias linijos ilgis
begin
   distMt = apskaičiuotiAtstumųIkiSienosMatricą(p);
   [srcPt, maxDst] = rastiTolimiausiąTaškąIkiFigūrosSienos(distMt);
   speedMt = distMt/maxDst; // padalinamas kiekvienas matricos elementas
   while(true)
      [T,Y] = pritaikytiGreitojoŽingsMetodą(speedMt, srcPt);
      strtPt = rastiTolimiausiąTaškąIkiFigūrosSienos (Y);
      shortestLn = apskaičiuotiTrumpiausiąLiniją(T, strtPt, srcPt);
      lnLength = gautiLinijosIlgį(shortestLn);
      // jei trumpiausios linijos ilgis trumpesnis už maksimalų vieno taško
atstumą iki sienos – stabdyti algoritmą
      if lnLength < maxDst * 2
        break
      end if

      srcPt = srcPt + shortestLn;

      // Pridedama nauja figūros vidurio linijos šaka
      midLn(size(midLn)+1) = shortestLn;
   end while
end
```

lentelė 1.3 pateikiamas pseudokodas atitinka šaltinio [8] algoritmo realizaciją.

1.3.2. Algoritmo apibendrinimas

Algoritmas veikia atsižvelgdamas į visus figūros sienos taškus, todėl vidurio liniją sudėtingoms figūroms gali apskaičiuoti tiksliai. Sukonstruotos figūros vidurio linijos pradžios ir pabaigos taškai liečia figūros sieną, todėl kai kuriems taikymams šis algoritmas gali būti netinkamas.

Sudėtingumas - $O(n \log n)$, kur n – daugiatrafarečio greitojo žingsniavimo algoritmo tinklelyje esančių taškų kiekis [žr. šaltinį [6]].

1.4. „Paprastojo dvimatės figūros vidurio linijos radimo algoritmo“ analizė

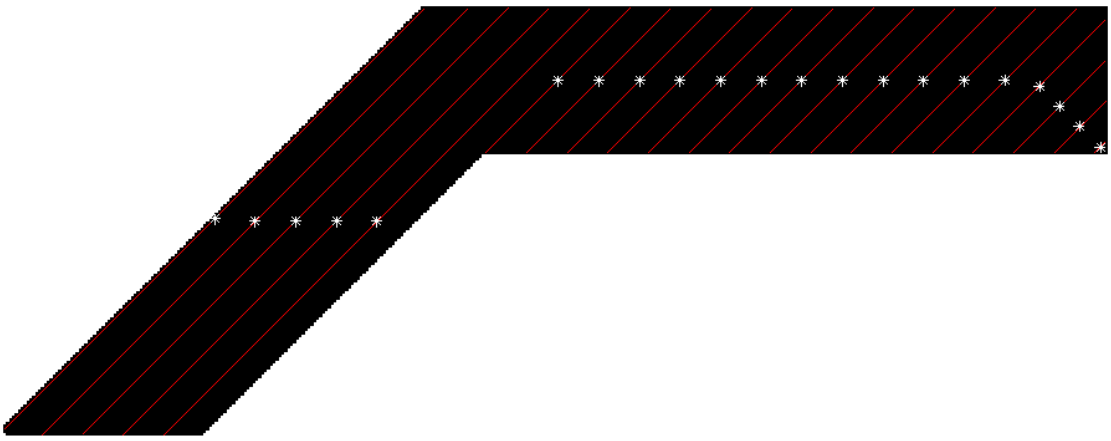
Šiame skyriuje analizuojama šaltinyje [1] pateikta idėja.

Šaltinyje [1] išskiriami 3 pagrindiniai žingsniai:

1) figūros subraižymas linijomis ir vidurio taškų išskyrimas;
2) vidurio taškų sujungimas į linijų grupes;
3) linijų grupių susikirtimo taškų aptikimas.

1) Figūros subraižymas linijomis ir vidurio taškų išskyrimas

Pirmasis algoritmo žingsnis yra paprastas, sudėtingumas – linijinis, ir jis tiesiškai priklauso nuo paveiksliuko aukščio arba pločio pokyčio. Figūra subraižoma lygiagrečiomis, vienodai viena nuo kitos nutolusiomis linijomis (toliau. Braižančiosios linijos). Galiausiai, iš subraižytos figūros gaunamas vidurio taškų sąrašas (žr. Pav. 1.5) (čia pažymėti baltomis žvaigždutėmis).

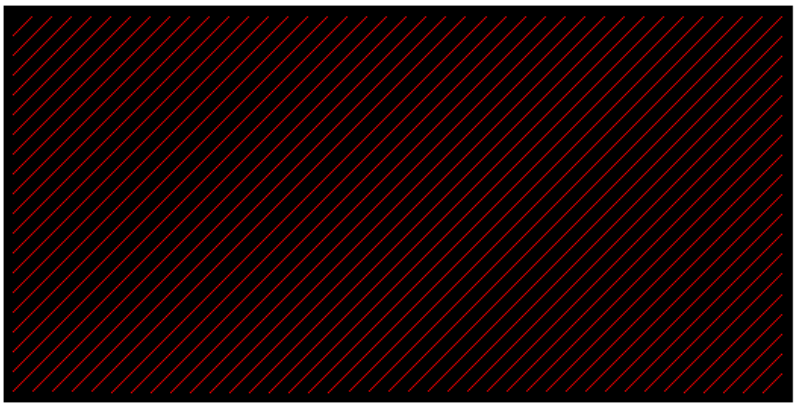


Pav. 1.5 45 laipsnių linijomis subraižyta figūra ir linijų vidurio taškai

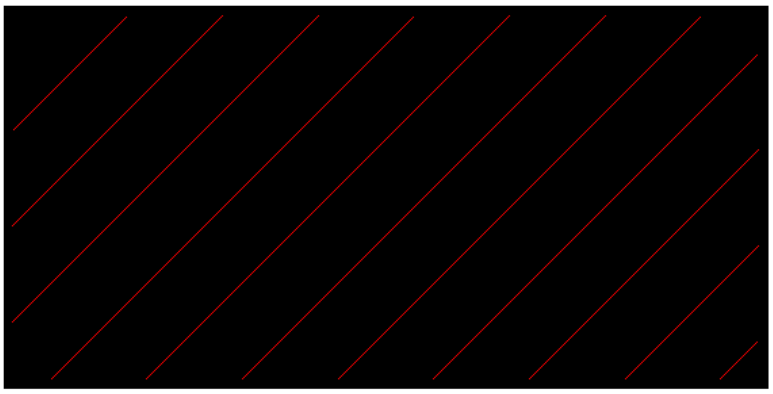
Braižant figūrą tokiu būdu nesunku pakoreguoti algoritmą ir brėžti tik kas n-tąją liniją, praleidžiant likusias.

Pavyzdžiui, kai brėžiama kiekviena linija, gaunamas Pav. 1.6 rezultatas, kai brėžiama tik kas ketvirta linija tas pats paveikslėlis subraižomas taip: žr. Pav. 1.7. Algoritmo realizacijoje šis linijų brėžimo žingsnis saugojamas „`EXAMINE_PICTURE_EVERY_NTH_LINE`" parametre.

Linijų praleidimas, braižant figūrą, naudingas tada, kai analizuojama figūra yra didelių išmatavimų ir tikslumas dėl praleidžiamų linijų nukenčia mažai.



Pav. 1.6 Subraižytas stačiakampis, kai analizuojama kiekviena linija

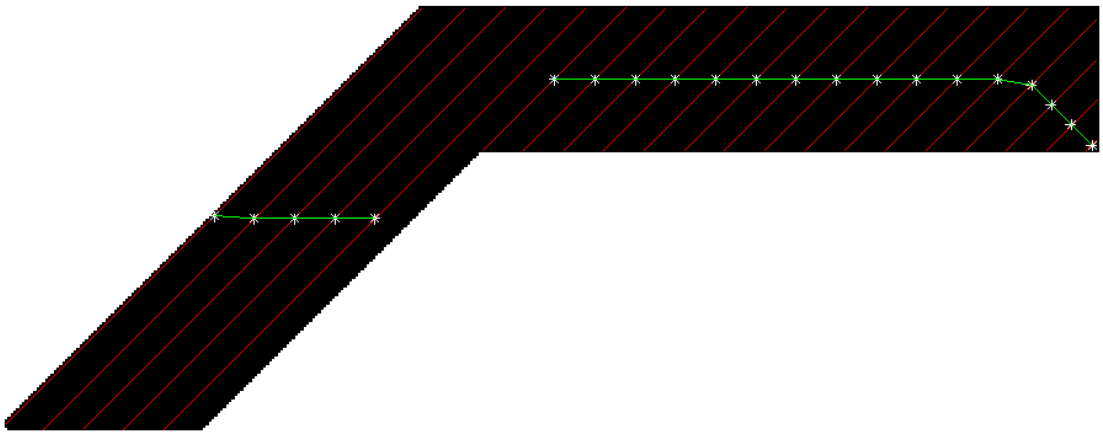


Pav. 1.7 Subraižytas stačiakampis, kai praleidžiamos 4 linijos

2) Vidurio taškų sujungimas

Antrajame algoritmo žingsnyje vidurio linijos taškai sujungiami. Šaltinyje [1] siūloma įvesti parametrą, kuris nurodytų tam tikrą maksimalų atstumą, kurį viršijus taškai nebūtų jungiami. Skaičiuojant šį atstumą būtų galima atsižvelgti ir į tą faktą ar pirmajame žingsnyje nagrinėjama buvo kiekviena linija ar tik n-toji ir šiuos dydžius susieti automatiškai. Algoritmo realizacijoje tai ir buvo atlikta. Galutinis šios vidurio taškų sujungimo dalies rezultatas – išskirtos sujungtų linijų grupės (žr. Pav. 1.8).

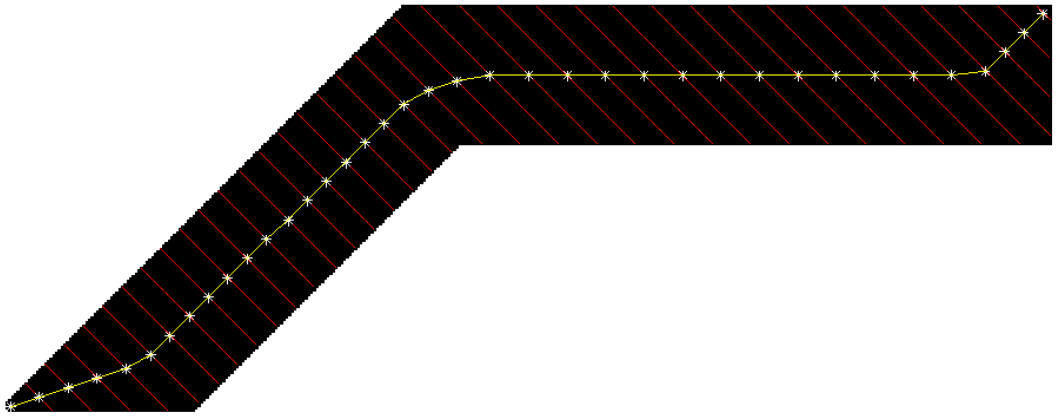
Kadangi trečiame žingsnyje („Sujungtų linijų susikirtimo taškų aptikimas“) bus atliekama linijų sankirta, kuri yra reikli skaičiavimo resursų prasme, tai jau šiame žingsnyje kaip tik galima pritaikyti kelias optimizacijas. Šaltinyje [1] siūloma nustatyti maksimalų vieną linijų grupę sudarančių taškų kiekį arba sujungti gautos linijų grupės pradžios ir galo taškus, praleidžiant visus vidurinius taškus, arba įvesti tam tikrą paklaidą, kurios neviršijus naujas vidurio taškas nekurtų naujos linijos, o pratęstų prieš tai sukurtą. Tačiau galima pasiūlyti nenaudoti tam tikros iš anksto nustatytos galimos paklaidos vertės, pratęsiant liniją ir pratęsti senąją liniją, tuomet kai linijų kampai Ox ašies atžvilgiu, sutampa, taip neprarandant tikslumo, o jei kampai nesutampa, tai tik tuomet iš naujo taško kurti naują liniją ir ją pridėti prie linijų grupės. Taip pat šiame žingsnyje kiekvienai linijų grupei būtų galima sukonstruoti po ribojantįjį stačiakampį (angl. *bounding box*), taip visiškai eliminuojant tarpusavyje nesusijusių linijų grupių susikirtimo tikrinimą. Šios pasiūlytos idėjos yra realizuotos sukurtame algoritme ir aptartos realizacijos skyriuje.



Pav. 1.8 Sujungti 45 laipsnių linijų vidurio taškai, priklausantys tai pačiai linijų grupei

1-2) Braižymas kitokio kampo linijomis

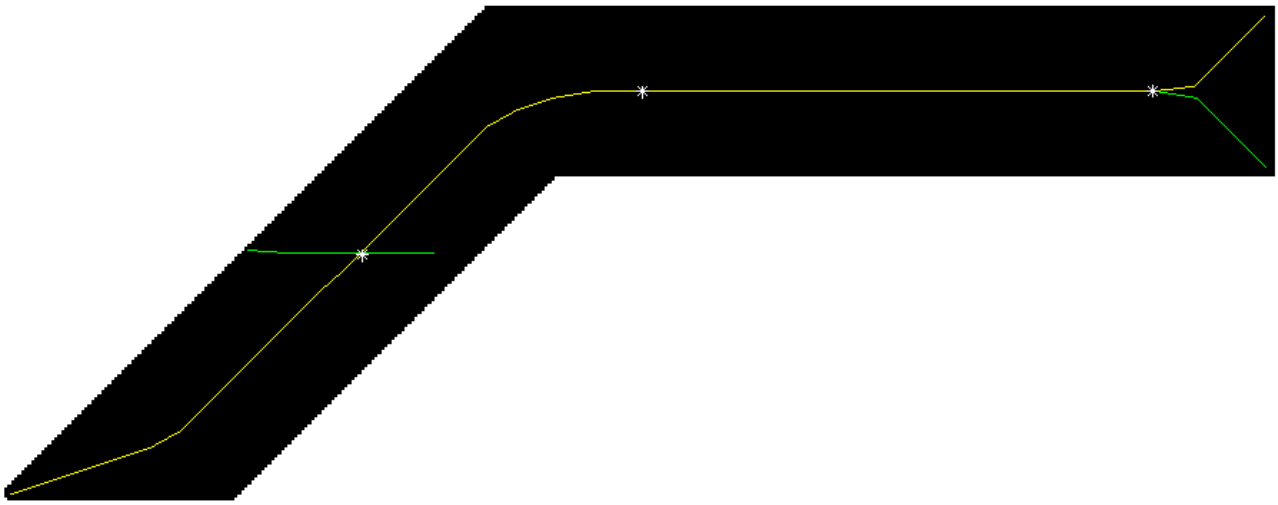
Pakartojamas pirmasis žingsnis („Figūros subraižymas linijomis ir vidurio taškų išskyrimas“), tačiau figūrą braižant kitokio kampo, tarkim, 135°, linijomis, tuomet randami kiekvienos figūrą kertančios linijos vidurio taškai, jie sujungiami, pagal antro žingsnio („Taškų sujungimas“) logiką ir gaunamas rezultatas (žr. Pav. 1.9).



Pav. 1.9 Sujungti 135 laipsnių linijų vidurio taškai, priklausantys tai pačiai linijų grupei

3) Linijų grupių susikirtimo taškų aptikimas

Šiame žingsnyje ieškoma sankirtos tarp išskirtųjų 135 ir 45 linijų grupių. Rezultatas (žr. Pav. 1.10).



Pav. 1.10 Taškai, gauti skirtingų linijų grupių susikirtimo taškuose

Kaip galima pastebėti Pav. 1.10, atlikus skaičiavimus, gaunami trys taškai, kurie yra vidurio linijos taškai, tačiau jų nepakanka tikslios vidurio linijos sukonstravimui, todėl ta pati figūra turi būti subraižoma ir kitokio laipsnio linijomis, pvz. 90 ir 180 laipsnių, ir algoritmas kartojamas. Šaltinyje [1] siūloma figūrą subraižyti linijomis, kurių pasvyrimo laipsniai kinta tam tikru žingsniu. Žingsnio dydis galėtų būti toks skaičius, kuris dalina skaičių 360 be liekanos, t.y. 90, 45, 22.5, 11.25, 5.625 ar net mažesnės reikšmės.

1.4.1. Metodo apibendrinimas

Metodas remiasi paprastomis geometrinėmis operacijomis. Plėtojant galima pritaikyti keletą optimizavimo strategijų.

2. PROJEKTAVIMAS

Šiame skyriuje pateikiama informacija, susijusi su sukurto figūros vidurio linijos radimo algoritmo realizacija.

2.1. Programavimo aplinka bei pasirinkta duomenų struktūra

Algoritmas yra realizuotas „Matlab 2016a“ programavimo aplinkoje, naudojant „Matlab“ programavimo kalbą.

Pirmojoje algoritmo versijoje vidurio taškų „x“ ir „y“ koordinačių saugojimui buvo naudojami „Matlab“ struktūrų masyvai, bei „Matlab“ celės (angl. *cell*) (žr. Pav. 2.1). Struktūrose buvo saugojamos taškų koordinačių reikšmės, susiję taškai talpinami į struktūrų masyvus, o šie struktūrų masyvai talpinami į atskirus celių masyvo elementus, taip pat struktūrų masyvams vieta iš anksto nebuvo išskiriama (žr. Pav. 2.1). Tokia duomenų struktūra efektyviai išnaudojo turimą atmintį, tačiau buvo sugaištama daug laiko iš principo paprastoms operacijoms – taško koordinačių nuskaitymui, naujo taško struktūros pridėjimui prie jau turimo taškų struktūrų masyvo. Todėl struktūrų buvo atsisakyta ir jos buvo pakeistos paprastomis skaičių matricomis (po vieną n*m matricą „x“ ir „y“ koordinatėms), kurioms kompiuterio atmintis yra išskiriama iš anksto, pagal jau paskaičiuotą maksimalų galimą atminties sunaudojimo kiekį konkrečiam atvejui, taip pat pridėtas n*1 pagalbinis masyvas, kuris saugo kiekvienoje matricos eilutėje saugomų taškų kiekį (žr. Pav. 2.2). Atlikus šiuos esminius pakeitimus algoritmas ėmė veikti sparčiau, nors turima atmintis tapo ne taip efektyviai išnaudojama. (Pav. 2.2 pateikiama duomenų struktūra saugo identiškus duomenis, kaip ir pateikiama Pav. 2.1 struktūra, tačiau veikimo sparta žymiai geresnė).

1	{[x:7;y:8],[x:1;y:3],[x:6;y:4],[x:2;y:5]}
2	{[x:7;y7]}
3	{}
…	{...}
n	{}

Pav. 2.1 n elementų dydžio celių masyvas, talpinantis įvairaus ilgio taškų struktūrų masyvus

x koordinačių matrica (n*m)

	1	2	3	4	5	…	m
1	7	1	6	2	0	…	0
2	7	0	0	0	0	…	0
3	0	0	0	0	0	…	0
…	…	…	…	…	…	…	…
n	0	0	0	0	0	…	0

y koordinačių matrica (n*m)

	1	2	3	4	5	…	m
1	8	3	4	5	0	…	0
2	7	0	0	0	0	…	0
3	0	0	0	0	0	…	0
…	…	…	…	…	…	…	…
n	0	0	0	0	0	…	0

taškų kiekis matricų eilutėse

1	4
2	1
3	0
…	…
n	0

Pav. 2.2 Duomenų modelis vidurio taškų saugojimui, naudojantis skaičių matricas, kurioms atmintis yra išskirta iš anksto

2.2. Pradiniai algoritmo duomenys

Pradiniai algoritmo duomenys – monochrominiai (angl. *monochrome*) „Bitmap“ tipo paveikslėliai, kuriuose kiekvienas pikselis koduojamas 1 bitu (0 arba 1). Čia „0“ – juoda spalva, o „1“ – balta. Ši informacija saugoma n*m išmatavimo matricose. Pavyzdžiui 22 pikselių ilgio ir 15 pikselių aukščio paveikslėlio (žr. Pav. 2.3) duomenų matrica atrodo taip: žr. lentelė 2.1.

Pav. 2.3 Pavyzdinis algoritmo duomuo

lentelė 2.1 Nuskaityto paveikslėlio 15x22 duomenų matrica

1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Juoda spalva paveikslėlyje (žr. Pav. 2.3) arba „0“ duomenų matricoje (žr. lentelė 2.1) simbolizuoja figūrą, o paveikslėlio balta spalva arba „1“ duomenų matricoje - figūrai nepriklausančią sritį.

2.2.1. Pradinių duomenų apribojimai

Be reikalavimų (žr. skyrių „Pradiniai algoritmo duomenys“) paveikslėlio tipui bei spalvoms, paveikslėlio figūrai taip pat yra keliami tam tikri reikalavimai.

Figūra turi turėti tik vieną kontūro liniją. Keturios figūros, kurios tenkina pradiniams duomenis keliamus reikalavimus pateiktos Pav. 2.4, o trys figūros, netenkinančios keliamų reikalavimų pateiktos Pav. 2.5.

Pav. 2.4 Figūros, atitinkančios keliamus reikalavimus

Pav. 2.5 Figūros, netenkinančios keliamų reikalavimų

Algoritmui pateikus figūras (žr. Pav. 2.5), kurios neatitinka figūros formai keliamų reikalavimų, gaunama figūros vidurio linija yra trūki (žr. Pav. 2.6).

Pav. 2.6 Rastos trūkios vidurio linijos trims paveikslėliams, netenkinantiems reikalavimų paveikslėlio formai

2.3. Algoritmo etapai

Šiame skyriuje aptariami sukurto figūros vidurio linijos paieškos algoritmo etapai.

Lyginant su analizės dalyje apžvelgta ir šaltinyje [1] pateikiama algoritmo idėja, prie algoritmo buvo pridėti papildomi du žingsniai – „matomumo grafo konstravimas“ ir „vidurio linijos išskyrimas iš matomumo grafo“, taip pat anksčiau pateikti algoritmo etapai tam tikrose vietose patobulinti. Naujasis algoritmas turi 5 etapus:

1) Paveikslėlio subraižymas ir vidurio taškų suradimas;
2) Vidurio taškų sujungimas;
3) Linijų grupių sukirtimas;
4) Matomumo grafo konstravimas;
5) Vidurio linijos išskyrimas iš matomumo grafo.

Tolesniuose poskyriuose aptariamas kiekvienas iš išvardintų algoritmo etapų.

2.3.1. Paveikslėlio subraižymas ir vidurio taškų suradimas

Linijos, atitinkančios duotą kampą, konstravimas

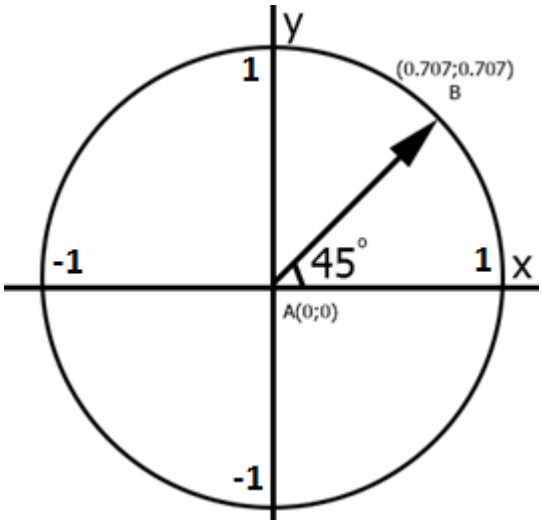
Subraižant figūrą tam tikro kampo linijomis iš pradžių iš duotosios kampo reikšmės išskiriamas vienetinis vektorius. Vienetinio vektoriaus pabaigos „x“ ir „y“ koordinatės apskaičiuojamos taip:

$$x = \cos\alpha$$ **formulė 2.1 Vienetinio vektoriaus pabaigos „x“ koordinatė**

$$y = \sin\alpha$$ **formulė 2.2 Vienetinio vektoriaus pabaigos „y“ koordinatė**

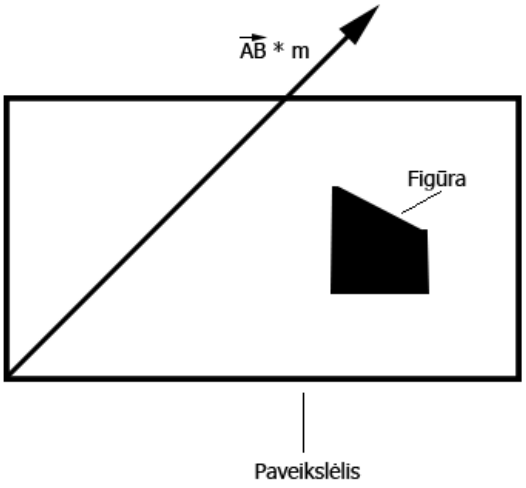
Čia α – duotoji kampo reikšmė.

Gaunamas vienetinis krypties vektorius (žr. Pav. 2.7).



Pav. 2.7 Vienetinis 45° krypties $\overrightarrow{AB}$ vektorius

Tuomet surandamas toks skaičius „m“, iš kurio padauginus vienetinį vektorių, naujai gauto vektoriaus pabaigos taškas visuomet bus už paveikslėlio išmatavimų, kai vektoriaus pradžios taškas sutampa su bet kuriuo paveikslėlio sienos tašku (žr. Pav. 2.8).



Pav. 2.8 Vienetinis vektorius padaugintas iš „m“

Koeficientas „m“ gali būti apskaičiuojamas pagal pateiktą pseudokodą (žr. lentelė 2.2).

lentelė 2.2 Minimalaus sandaugos koeficiento radimo pseudokodas

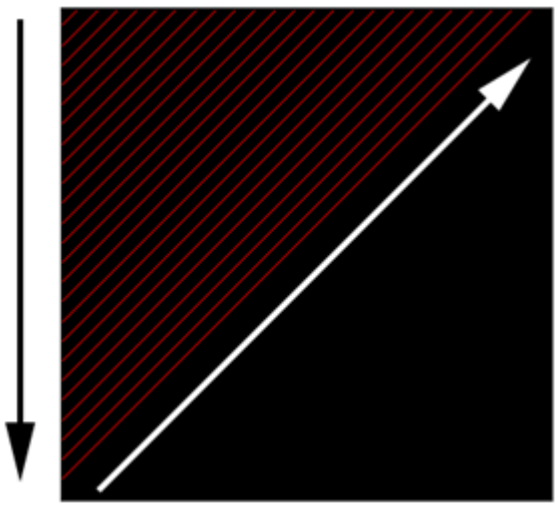
```
procedure skaičiuotiM(pavPlotis, pavAukštis, ABx, ABy)
// in: pavPlotis – paveikslėlio plotis
// in: pavAukštis – paveikslėlio aukštis
// in: ABx – vienetinio vektoriaus x koordinatė
// in: ABy – vienetinio vektoriaus y koordinatė
// out: m – koeficientas,  iš kurio reikia padaugint vienetinį vektorių, taip
jog naujai gauto vektoriaus pabaigos taškas visuomet būtų už paveikslėlio
ribų, kai vektoriaus pradžios taškas gali sutapti su bet kuriuo paveikslėlio
kontūro tašku
begin
   m = infinity;
   if ABx != 0
      m = round(pavPlotis/(abs(ABx))) + 1;
   end if

   if ABy != 0 && (round(pavAukštis /(abs(ABy))) + 1) < m
      m = round(pavAukštis /(abs(ABy))) + 1;
   end if

   return m;
end
```

Apskaičiuotojo vektoriaus (žr. Pav. 2.8) pradžios koordinatės yra (0;0) taške, todėl šio vektoriaus pabaigos koordinatės nurodo pokyčius x ir y ašyse. Apskaičiuotos pokyčių reikšmės naudojamos brėžiant kiekvieną tolimesnę to paties kampo liniją, prie linijos pradžios koordinatės, pridedant apskaičiuotas pokyčių reikšmes.

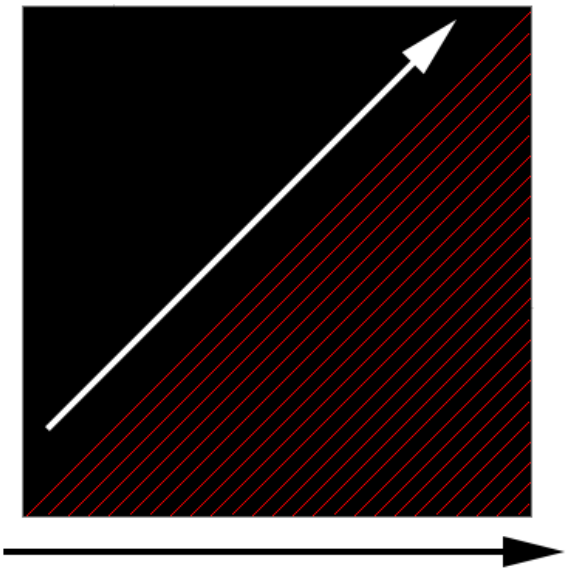
Kiekvienos paveikslėlį analizuojančios linijos pradžia – taškas, esantis ant paveikslėlio kontūro, pabaiga – taškas esantis už paveikslėlio kontūro (žr. Pav. 2.8 $\overrightarrow{AB} * m$ vektorių). Duotasis paveikslėlis analizuojamas brėžiant liniją Bresenhamo metodu tarp pradžios ir pabaigos taško. Taškas, esantis už paveikslėlio kontūro niekada nėra pasiekiamas, kadangi visuomet greičiau pasiekiamas pats paveikslėlio kontūras (žr. Pav. 2.8). Šis pabaigos taškas už paveikslėlio kontūro tik nurodo kryptį linijos brėžimo algoritmui Bresenhamo metodu.

Linijų rikiavimas

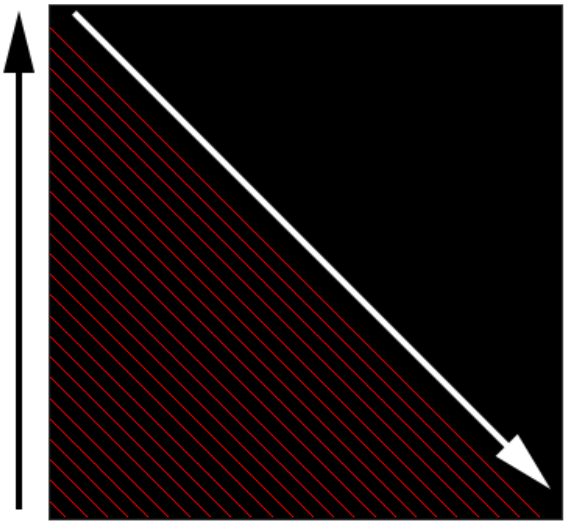
Išrikiuotos linijos paspartina algoritmo veikimą, todėl linijų brėžimo eiliškumas yra svarbu. Linijoms, kurių kampas yra nuo „0“ laipsnių iki „90“ laipsnių (įskaitant abi reikšmes) naudojami Pav. 2.9 (pirmam etapui), Pav. 2.10 (antram etapui) braižymo būdai, o linijoms, kurių kampas yra nuo „90“ laipsnių iki „180“ (arba nuo „270“ iki „0“) laipsnių (įskaitant abi reikšmes) Pav. 2.11 bei Pav. 2.12 linijų braižymo būdai.



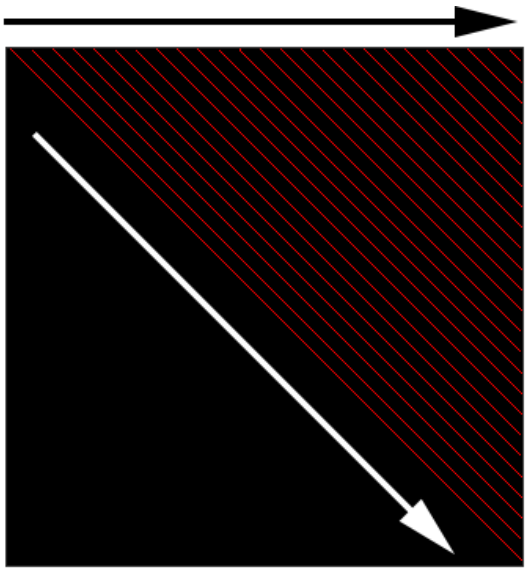
Pav. 2.9 Pirmasis 0-90 laipsnių linijų brėžimo etapas



Pav. 2.10 Antrasis 0-90 laipsnių linijų brėžimo etapas



Pav. 2.11 Pirmasis 90-180 (270-0) laipsnių linijų brėžimo etapas



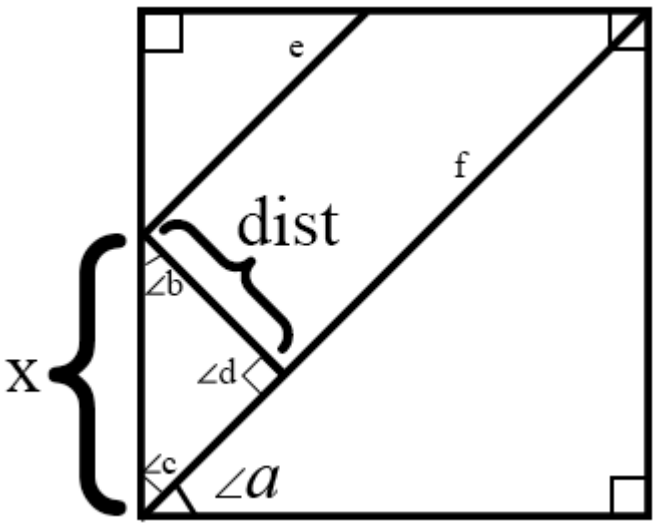
Pav. 2.12 Antrasis 90-180 (270-0) laipsnių linijų brėžimo etapas

Pav. 2.9, Pav. 2.10, Pav. 2.11 ir Pav. 2.12 viename iš paveikslėlio šonų esanti juoda rodyklė žymi paveikslėlio pusę, kurioje yra linijų pradžios taškai, o kryptis - linijų sekos didėjimo kryptį, baltos rodyklės kryptis žymi linijų brėžimo kryptį. Kai paveikslėlis yra apdorotas abiem etapais (Pav. 2.9 ir Pav. 2.10 arba Pav. 2.11 ir Pav. 2.12), galima teigti, kad visas paveikslėlis buvo subraižytas to paties kampo linijomis.

Iš išrikiuotų linijų gauti vidurio taškai bus taip pat surikiuoti, o surikiavimas pagelbės linijų vidurio taškų jungimo etape, nes nereikės perrinkinėti visų vidurio taškų - užteks tik apdoroti greta esančių vidurio taškų duomenis.

Atstumas tarp braižomų linijų

Braižant paveikslėlį linijomis, taip pat labai svarbu išlaikyti vienodą atstumą tarp pačių linijų, tam, kad visi gauti vidurio taškai būtų gauti esant vienodoms sąlygoms (čia atstumas tarp linijų – statmenos ir jungiančios dvi gretimas lygiagrečias linijas linijos ilgis). Atstumas tarp braižančiųjų linijų gali būti koreguojamas, pasirenkant kitą linijos pradžios tašką, esantį ant paveikslėlio kontūro linijos. Atstumo problemų tarp paveikslėlį analizuojančių linijų nekyla, kai pasirinktas linijų braižymo kampas yra lygiai „0“ ar „90“ laipsnių dydžio, tačiau pasirinkus paveikslėlį analizuoti pvz. „22.5“ laipsnių linijomis, kiekvienam iš linijos brėžimo etapų (pvz. „22.5“ laipsnių linijoms brėžti naudojami Pav. 2.9, Pav. 2.10 etapai) reikia perskaičiuoti žingsnį kuriuo bus renkamasi sekanti brėžiamos linijos pradžios koordinatė, esanti ant paveikslėlio kontūro. Šis uždavinys pirmam 0-90 laipsnių linijų etapui (žr. Pav. 2.9) pailiustruotas Pav. 2.13.



Pav. 2.13 Uždavinys pirmam linijų braižymo etapui (žr Pav. 2.9)

Čia *dist* – atstumas tarp lygiagrečių linijų, lygus parametrui linijų brėžimo žingsnio parametrui, kuris buvo aptartas analizės skyriaus 1.4 dalyje.

$\angle\alpha$ – duotasis linijų braižymo kampas

x – ieškomas žingsnio dydis paveikslėlio kontūru, kuris leidžia brėžti linijas nutolusias per atstumą „*dist*“ vienai nuo kitos.

e ir *f* – linijos, kurios yra lygiagrečios.

Uždavinys spendžiamas remiantis sinusų teorema (žr. formulė 2.3):

$$\frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma}$$ **formulė 2.3 Sinusų teorema**

Įstatomos reikšmės iš Pav. 2.13 ir gaunamas reiškinys (žr. formulė 2.4)

$$\frac{dist}{\sin(90° - \angle\alpha)} = \frac{x}{\sin 90°}$$ **formulė 2.4 Įstatytos reikšmės**

Galiausiai randamas sprendinys (žr. formulė 2.5), kuris vėliau bus suapvalintas iki sveikosios dalies.

$$x = \frac{dist}{\sin(90° - \angle\alpha)}$$ **formulė 2.5 Atstumo tarp linijų pradžios taškų lygtis**

Šis uždavinys (žr. Pav. 2.13), įvertinant šiek tiek pasikeitusią uždavinio sąlygą, analogiškai sprendžiamas Pav. 2.10, Pav. 2.11 ir Pav. 2.12 linijų braižymo etapams.

Atminties išskyrimas rezultatų matricoms

Atmintis rezultatų matricoms išskiriama pagal maksimalų teorinį galimą elementų kiekį matricoje. Pseudokodas, aprašantis atminties išskyrimą rezultatų matricoms, pateikiamas „ lentelė 2.3“ lentelėje. Čia „maxLinesCount“ yra realus maksimalus brėžiamų linijų kiekis, kai linijų brėžimo žingsnis yra 1, o linijos brėžiamos 45° kampu, „maxPossibleMidPointsInOneLine“ – maksimalus teorinis vienos brėžiamos linijos rastas vidurio taškų kiekis – paveikslėlio įžambinės ilgis. Išskirta atmintis bus pilnai sunaudojama, kai paveikslėlį sudarys daug vieno pikselio storio statmenų linijų, o bent vienos brėžiamosios linijos pradžios ir pabaigos taškai sutampa su paveikslėlio įžambinės pradžios ir pabaigos taškais.

„midPtsX“ ir „midPtsY“ kintamiesiems išskirta atmintis atsižvelgiant į maksimalų linijų kiekį ir maksimalų vidurio taškų kiekį, gautą brėžiant vieną liniją. „midPtsX“ ir „midPtsY“ matricų eilučių indeksai žymi brėžtos linijos numerį, o stulpelio indeksas – vidurio taško koordinatės reikšmę rastą brėžiant tą liniją.

lentelė 2.3 Rezultatų matricų išskyrimas pirmam algoritmo etapui

```
procedure allocateMemoryForMidPts(xLength, yLength)
// in: xLength – pradinio paveikslėlio plotis
// in: yLength – pradinio paveikslėlio aukštis
// out: midPtsX – linijų vidurio taškų x koordinačių
// out: midPtsY - linijų vidurio taškų y koordinačių
// out: midPtsCountForEachLine – masyvas, saugantis kiekvienos brėžtosios
linijos rastą vidurio taškų kiekį
// local: maxLinesCount – maksimalus galimas linijų kiekis
// local: maxPossibleMidPointsInOneLine – maksimalus galimas vidurio linijos
taškų kiekis vienoje linijoje
begin
  maxLinesCount = xLength + yLength +1;
  maxPossibleMidPointsInOneLine = round(sqrt(xLength*xLength +
yLength*yLength)/2)+1;

  midPtsX = allocate(maxLinesCount, maxPossibleMidPointsInOneLine);
  midPtsY = allocate (maxLinesCount, maxPossibleMidPointsInOneLine);

   midPtsCountForEachLine = allocate(1, maxLinesCount);

  return [midPtsX, midPtsY, midPtsCountForEachLine];
end
```

Paveikslėlio subraižymo ir vidurio taškų išskyrimo pseudokodas

Toliau pateikiamas pseudokodas (žr.
lentelė 2.4) atitinkantis pirmąjį linijų brėžimo etapą (žr. Pav. 2.9) kai brėžiančiosios linijos kampos Ox ašies atžvilgiu yra tarp 0°-90° imtinai. Kitiems brėžimo etapams (žr. Pav. 2.10, Pav. 2.11 ir Pav. 2.12) kodas yra analogiškas, tačiau atsižvelgiama į kitokias sąlygas.

lentelė 2.4 Paveikslėlio subraižymo pseudokodas pirmam subraižymo etapui (žr. Pav. 2.9)

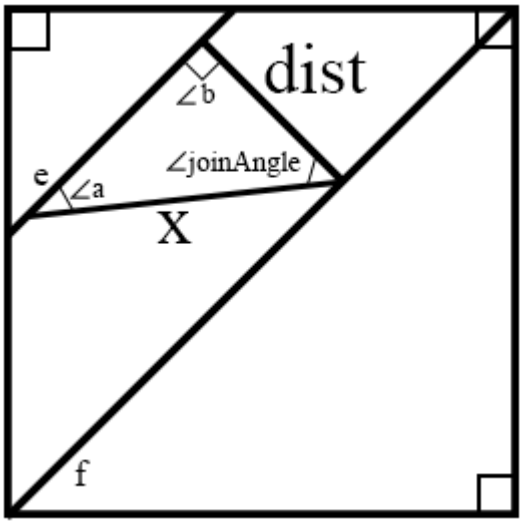
```
procedure getMidPtsFor0to90AngleLnsPart1(yLength, xLength, p,
nthRowToAnalyzeOnYAxis, angle)
// in: yLength – duotojo paveikslėlio aukštis
// in: xLength – duotojo paveikslėlio ilgis
// in: p – duotasis paveikslėlis
// in: nthRowToAnalyzeOnYAxis – konstanta, nurodo brėžiamų linijų intervalą,
tam, kad būtų išlaikytas norimas atstumas tarp dviejų lygiagr. linijų.
// in: angle – kampas, kuriuo braižomos linijos
// out: midPtsX – užpildyta linijų vidurio taškų x koordinačių matrica
kiekvienai brėžtai linijai
// out: midPtsY - užpildyta linijų vidurio taškų y koordinačių matrica
kiekvienai brėžtai linijai
// out: midPtsCountForEachLine – užpildytas masyvas, saugantis kiekvienos
brėžtos linijos rastą vidurio taškų kiekį
// local: pixel – 1x2 matrica, talpinanti brėžiamos linijos apdorojamo
pikselio x ir y coordinates
// local: sequenceNumber – apdorojamos linijos numeris
// local: startPointX – brėžiamos linijos pirmojo sutikto figūros taško x
koordinatė
// local: startPointY - brėžiamos linijos pirmojo sutikto figūros taško y
koordinatė
// local: endPointX - brėžiamos linijos paskutinė sutikto figūros taško x
koordinatė
// local: endPointY - brėžiamos linijos paskutinė sutikto figūros taško y
koordinatė
// local: midPointX - linijos vidurio taško x koordinatė
// local: midPointY - linijos vidurio taško y koordinatė
begin
   [midPtsX,midPtsY,midPtsCountForEachLine] = allocateMemoryForMidPts(xLength,
                                                                    yLength);

   for y = 1: yLength
     if mod(y, nthRowToAnalyzeOnYAxis) == 0
       pixel(1) = 1;
       pixel(2) = y;
       sequenceNumber = sequenceNumber + 1;
       while(pixel(1) <= xLength && pixel(2) > 0)
         if p(pixel(2), pixel(1)) == 0
           startPointX = pixel(1);
           startPointY = pixel(2);
           while pixel(1) <= xLength && pixel(2) > 0 && p(pixel(2), pixel(1))
                                                                        == 0
             endPointX = pixel(1);
             endPointY = pixel(2);

             pixel = getNextPixelUsingBresenhamsMethod(pixel, angle);
           end while

           midPointX = (startPointX + endPointX) / 2;
           midPointY = (startPointY + endPointY) / 2;

           midPtsCountForEachLine(sequenceNumber) =
                              midPtsCountForEachLine(sequenceNumber) + 1;
           midPtsX(sequenceNumber, midPtsCountForEachLine(sequenceNumber)) =
                                                                  midPointX;
           midPtsY(sequenceNumber, midPtsCountForEachLine(sequenceNumber)) =
                                                                  midPointY;
```

```
          else
            pixel = getNextPixelUsingBresenhamsMethod(pixel, angle);
          end if
        end while
      end if
    end for

    return [midPtsX,midPtsY,midPtsCountForEachLine];
end
```

2.3.2. Vidurio taškų sujungimas į linijų grupes

Maksimalaus taškų jungimo nuotolio apskaičiavimas

Algoritmo realizacijoje maksimalus linijų vidurio taškų jungimo nuotolis apskaičiuojamas, remiantis linijų brėžimo žingsniu, t.y. remiamasi atstumu tarp dviejų lygiagrečių braižančių linijų. Taip pat vartotojos gali įvesti ne tam tikrą maksimalų atstumą iki kurio du vidurio taškai dar bus jungiami, tačiau kampo reikšmę, kuria remiantis maksimali vidurio taškų jungimo nuotolio reikšmė bus apskaičiuota automatiškai (žr. Pav. 2.14).



Pav. 2.14 Uždavinio formuluotė maksimalaus taškų jungimo nuotolio apskaičiavimui

Čia *dist* – atstumas tarp lygiagrečių linijų, $\angle joinAngle$ – vartotojo nurodytas vidurio taškų jungimo kampas, *x* – maksimalus vidurio taškų jungimo atstumas, *e* ir *f* – lygiagrečios linijos.

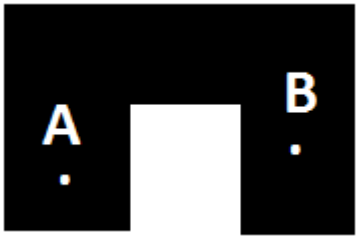
Pav. 2.14 uždavinys išsprendžiamas pasinaudojus sinusų teorema (žr. formulė 2.3). Gaunama lygtis formulė 2.6.

$$x = \frac{dist}{\sin(180° - 90° - \angle joinAngle)}$$

formulė 2.6 Maksimalaus taškų jungimo nuotolio apskaičiavimas

Taškų jungimas į linijų grupes

Norint išsiaiškinti ar galima sujungti du gretimų braižančiųjų linijų rastus vidurio taškus, reikia įvertinti ne tik atstumą nuo vieno vidurio taško iki kito, bet ir tai ar jungianti linija nekirs figūros sienos (žr. Pav. 2.15).



Pav. 2.15 Taškų jungimas

Norint išspręsti šią problemą (žr. Pav. 2.15) nuo taško A iki taško B brėžiama linija naudojant Bresenhamo algoritmą, tuomet pakeliui tikrinami visi tai linijai priklausantys pikseliai, jei kuris nors pikselis nepriklauso figūrai, laikoma, kad tų taškų jungti tarpusavyje negalima.

Tikrinimo galima atsisakyti ir taškus jungti iškart, tuomet, kai du vidurio taškai yra gretimuose pikseliuose (žr. Pav. 2.16), čia taškui P gretimi pikseliai yra P1, P2, P3, P4, P5, P6, P7, P8.

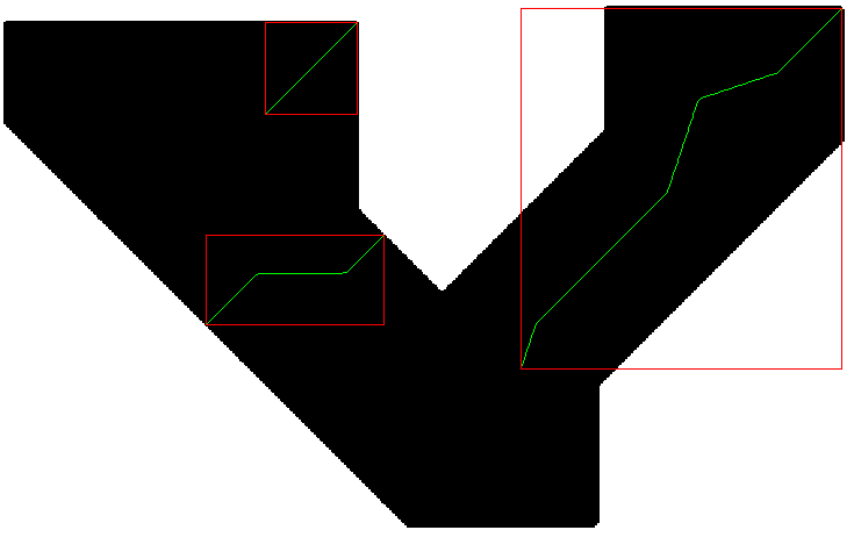
P1	P2	P3
P8	P	P4
P7	P6	P5

Pav. 2.16 Pikseliui P gretimi pikseliai

Ribojantys stačiakampiai

Norint optimizuoti kito algoritmo etapo linijų grupių susikirtimo taškų paieškos operaciją, šiame etape kiekvienai sudarytai linijų grupei reikia paskaičiuoti ribojantįjį stačiakampį. Ribojantysis stačiakampis yra sudarytas iš dviejų porų koordinačių, iš kurių viena pora nurodo tam stačiakampiui priklausantį pikselį su mažiausiomis „x“ ir „y“ koordinačių reikšmėmis, o kita pora – pikselį su maksimaliomis „x“ ir „y“ koordinačių reikšmėmis.

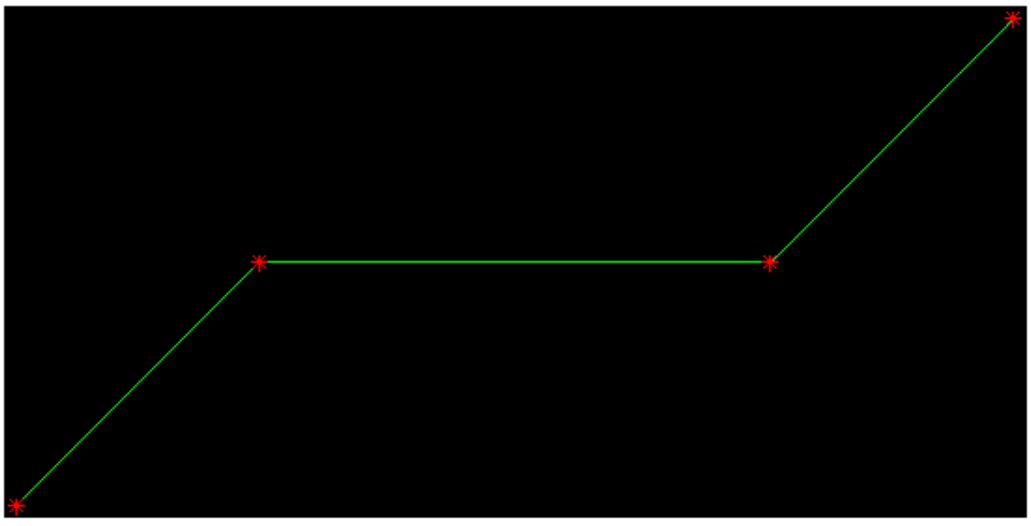
Linijų grupės ir jas apgaubiantys ribojantys stačiakampiai pavaizduoti Pav. 2.17.



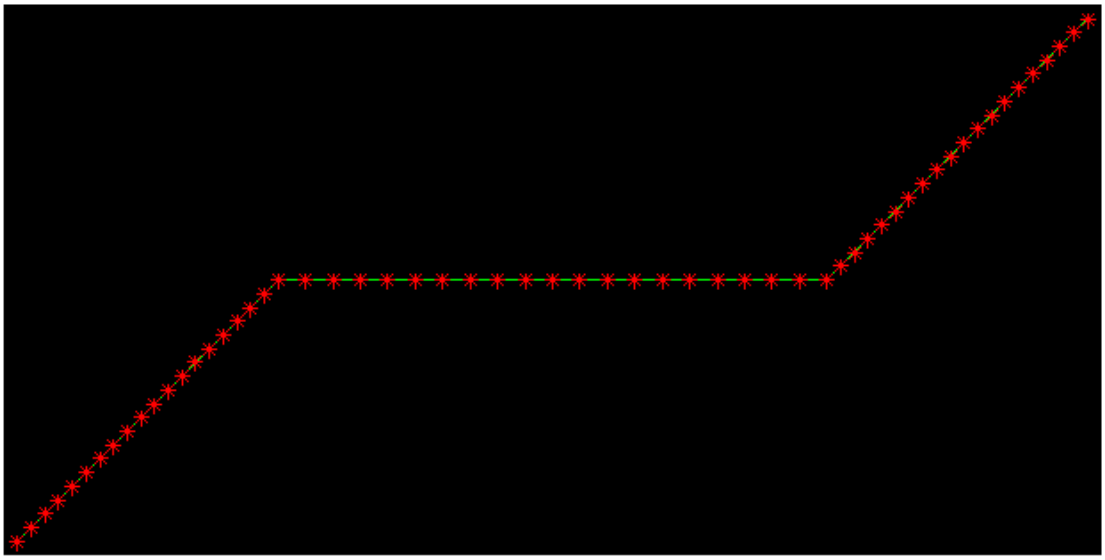
Pav. 2.17 Linijų grupės, gautos sujungus linijų vidurio taškus (figūra buvo braižoma 135° linijomis), bei tas linijų grupės gaubiantys ribojantieji stačiakampiai

Linijos pratęsimas

Dar viena optimizacija, kuri pagreitina sekančio etapo linijų grupių susikirtimo taškų paieškos operaciją yra linijos pratęsimas. Pratęsiant liniją, buvusi linija pratęsiama iki naujojo taško, taip sumažinant liniją sudarančių taškų kiekį. Linija pratęsiama tik tuo atveju jei pratęstos linijos kampas *Ox* ašies atžvilgiu sutaptų su jau esamos linijos kampu *Ox* ašies atžvilgiu. Linijų grupė, gauta pritaikius šią optimizaciją yra pateikta Pav. 2.18 (čia žvaigždutės žymi linijų, sudarančių šią liniją pradžios ir pabaigos taškus), galima palyginti su Pav. 2.19, kur ši optimizacija nėra pritaikyta. Akivaizdu, jog toks linijų apdorojimas smarkiai sumažina bendrą linijų grupę sudarančių taškų kiekį.



Pav. 2.18 Gauta linijų grupė pritaikius linijos pratęsimo optimizaciją



Pav. 2.19 Gauta linijų grupė be linijos pratęsimo optimizacijos

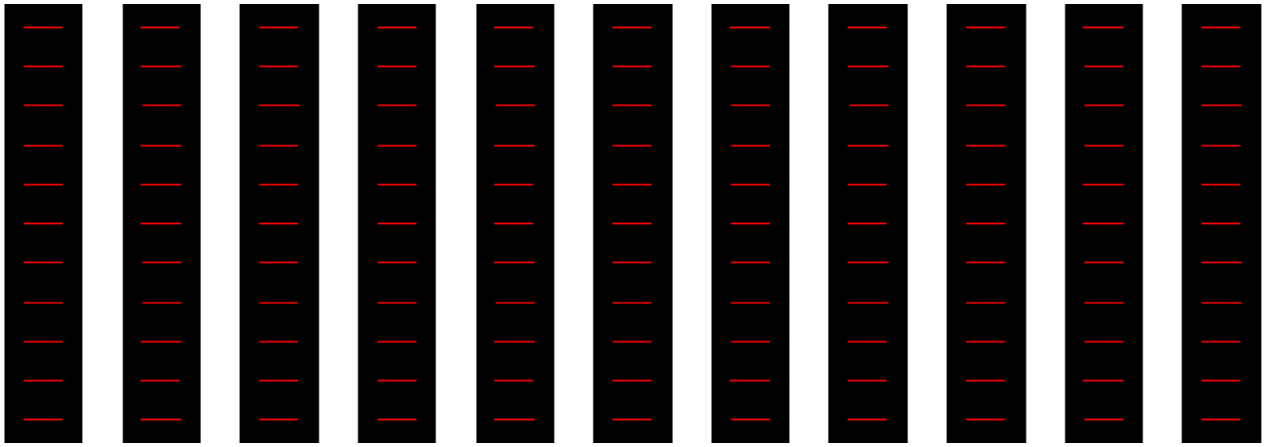
Atminties išskyrimas rezultatų matricoms

Šio etapo rezultatų matricoms taip pat iš anksto išskiriamas atminties kiekis, atsižvelgiant į galimą maksimalų teorinį rezultatų skaičių (žr. lentelė 2.5).

lentelė 2.5 Atminties išskyrimo pseudokodas linijų jungimo etapui

```
procedure allocateMemory(rowsCount, maxMidPointsPerLine)
// in: rowsCount – brėžtų linijų kiekis
// in: maxMidPointsPerLine – maksimalus vidurio taškų kiekis rastas brėžiant
vieną liniją
// out: joinedMidpointsX – linijų grupių x koordinačių matrica
// out: joinedMidpointsY – linijų grupių y koordinačių matrica
// out: joinedMidpointsBoundingBoxes - linijų grupių ribojančiųjų
stačiakampių matrica
// out: joinedMidpointsCountForEachLine – matrica, kurioje saugojamas
kiekvienos linijų grupės taškų kiekis
// local: maxPossibleLines – didžiausias galimas linijų kiekis
begin
   if maxMidPointsPerLine <= 0
        maxPossibleLines = 0;
   end if

   if maxMidPointsPerLine == 1
        maxPossibleLines = rowsCount+1;
   end if

   if maxMidPointsPerLine > 1
        maxPossibleLines =
maxMidPointsPerLine*round(rowsCount/(maxMidPointsPerLine-1))+1;
   end if

   joinedMidpointsX = allocate(maxPossibleLines, rowsCount);
   joinedMidpointsY = allocate (maxPossibleLines, rowsCount);
   joinedMidpointsCountForEachLine = allocate (maxPossibleLines, 1);

   joinedMidpointsBoundingBoxes = allocate (maxPossibleLines, 4);

end
```

Maksimalus galimas atminties kiekis sunaudojamas tada, kai iš paveikslėlio išskiriama labai daug linijų, kurios susideda tik iš dviejų taškų (žr. Pav. 2.20), čia juodi stulpeliai yra dviejų pikselių pločio, tarpai tarp stulpelių yra 1 pikselis.



Pav. 2.20 Sąlyga kai, išskirta atmintis sunaudojama maksimaliai

2.3.3. Linijų grupių sukirtimas

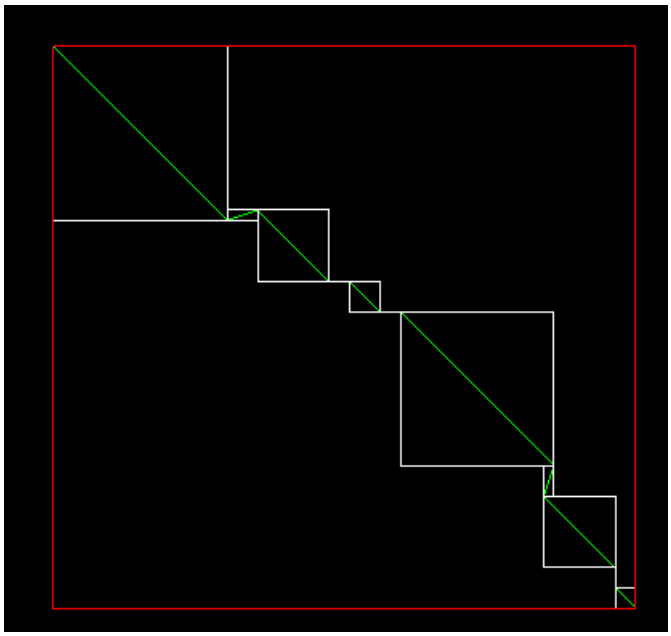
Šiame etape yra ieškoma visų dviejų linijų grupių elementų susikirtimo taškų. Linijų grupės buvo išskirtos iš dviejų skirtingų figūros subraižymų, o tų subraižymų laipsnių skirtumas yra 90°.

Viena linijų grupė yra sudaryta iš daugybės sujungtų mažesnių linijų. Todėl linijų grupių sukirtimo taškų ieškojimas yra brangus skaičiavimo prasme. Sudėtingumas $O(n \cdot m \cdot n_{max} \cdot m_{max})$, kur n – pirmo linijų grupių sąrašo dydis, m – antro linijų grupių sąrašo dydis, n_{max} – didžiausias linijų grupės elementų kiekis pirmame linijų grupių sąraše, m_{max} - didžiausias linijų grupės elementų kiekis antrame linijų grupių sąraše. Labai daug skaičiavimų galima išvengti, sudarant ribojančių talpų hierarchiją (angl. *bounding volume hierarchy*) (šiame skyriuje pritaikyta idėjos esmė atitinka šaltinio [9] keliamas idėjas).

Ribojančių talpų hierarchija

Šiame etape yra naudojami antrame („Vidurio taškų sujungimas į linijų grupes“) etape sukurti ribojantys stačiakampiai. Susikirtimo taškų tarp dviejų linijų grupių nebus ieškoma, jei tų linijų grupių ribojantieji stačiakampiai nepersidengs. Jei ribojantieji linijų grupių stačiakampiai persidengia, galima pradėti vertinti ar kuri nors linijų grupės linija kertasi su kitos linijų grupės linija. Tačiau šiuo atveju pritaikyta dar viena optimizacija – prieš bandant atlikti susikirtimo patikrinimą, sugeneruojami dviejų tikrinamų linijų ribojantieji stačiakampiai ir patikrinama ar jie persidengia, jei persidengia, tik tuomet atliekama linijų susikirtimo tikrinimo operacija, tai -

įgyvendinta ribojančių talpų hierarchijos idėja. Ribojančių talpų hierarchija gali būti pavaizduota taip: žr. Pav. 2.21.



Pav. 2.21 Ribojančiųjų talpų hierarchija

Čia raudonas stačiakampis – ribojantysis linijų grupės stačiakampis, baltas stačiakampis – ribojantysis vienos linijos stačiakampis, žalios linijos – linijų grupei priklausančios linijos. Šiame paveikslėlyje kai kurie ribojantieji linijų stačiakampiai sutampa su linijų grupės linijomis, kadangi tų linijų kampas *Ox* ašies atžvilgiu yra lygus 0°. Visos linijų grupei priklausančios linijos yra sujungtos.

Linijų susikirtimo tikrinimo operacija

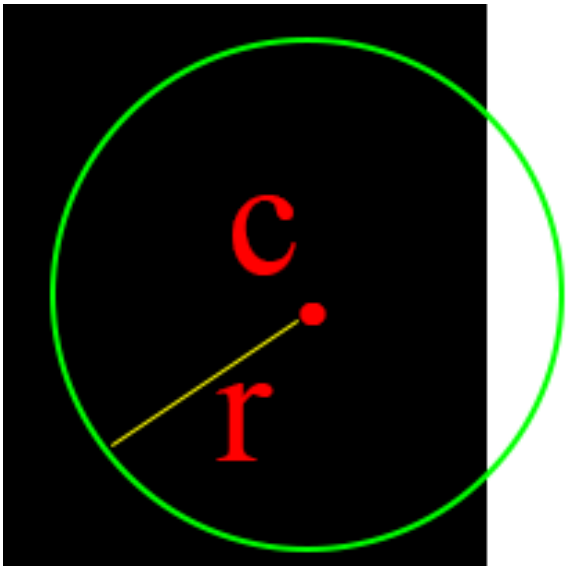
Bendruoju atveju linijų susikirtimo taškai aptinkami iš kiekvienos linijos gaunant tiesės lygtį, tuomet iš šių lygčių sudarant lygčių sistemą (žr. formulė 2.7) ir ją sprendžiant, po to patikrinama ar gautasis susikirtimo taškas priklauso abiejų linijų ribojantiesiems stačiakampiams, jei priklauso, vadinasi linijos kertasi.

$$\begin{cases} k_1 \cdot x + b_1 = 0 \\ k_2 \cdot x + b_2 = 0 \end{cases}$$ **formulė 2.7 sistema, skirta rasti tiesių susikirtimo taškui**

Linijos turi begalybę bendrų taškų, kai tų linijų kampai *Ox* ašies atžvilgiu sutampa ir pačios linijos persidengia. Tokiu atveju išskiriami tik du susikirtimo taškai – susikirtimo pradžios taškas ir susikirtimo pabaigos taškas. Taip pat sprendžiamos specialios situacijos, kai viena iš linijų ar net abi yra statmenos.

Susikirtimo taškų filtravimas

Analizuojant sudėtingas figūras, kartais gali būti gaunami tokie linijų grupių susikirtimo taškai, kurie yra ties figūros kontūro linija arba tiesiog per arti kontūro. Šiai problemai spręsti įvedamas „MIN_INT_PT_DIST_TO_BOUNDARY“ parametras, pagal nutylėjimą reikšmė lygi „0“ (filtravimas išjungtas). Filtravimo metu iš kiekvieno linijų grupių susikirtimo taško brėžiamas apskritimas, panaudojant Bresenhamo algoritmą apskritimo brėžimui. Apskritimo spindulio dydis yra nurodytasis parametras „MIN_INT_PT_DIST_TO_BOUNDARY“, brėžiant apskritimą analizuojamas kiekvienas pikselis, esantis ant apskritimo linijos ir jei aptinkamas figūrai nepriklausantis taškas, vidurio taškas, kuris buvo brėžto apskritimo centras, į rezultatų sąrašą nėra įtraukiamas (žr. Pav. 2.22).



Pav. 2.22 Susikirtimo taškas „c“ yra arčiau figūros kontūro linijos nei spindulys „r“

Analizuojant sudėtingas figūras rankiniu būdu gali nepavykti nustatyti visiems atvejams tinkamos „MIN_INT_PT_DIST_TO_BOUNDARY“ parametro reikšmės, kuri išfiltruotų visus netinkamus vidurio taškus, ši filtravimo reikšmė turėtų būti paskaičiuota dinamiškai kiekvienam linijų grupių susikirtimo taškui, tačiau šio algoritmo realizacijoje tai nėra atlikta.

Galima pasiūlyti idėją. Kadangi linijų grupių susikirtimo taškas žymi, dviejų statmenų, figūrą dalinančių linijų, bendrą vidurio tašką. Galima būtų paimti trumpesniąją liniją, ją padalinti per pusę ir tai galėtų būti filtravimo operaciją atliekančio apskritimo spindulys.

Atminties išskyrimas

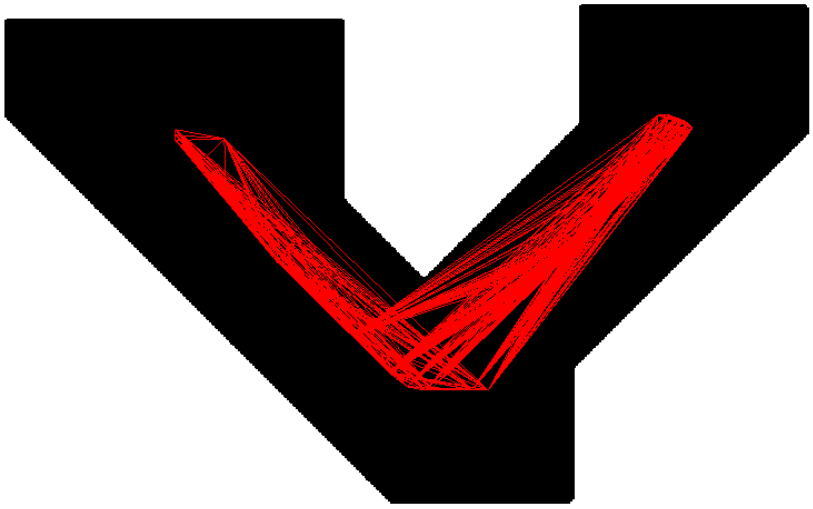
Šiame žingsnyje atmintis rezultatų matricai iš anksto nėra išskiriama, kadangi maksimalus teorinis linijų susikirtimų kiekis yra labai didelis.

2.3.4. Matomumo grafo konstravimas

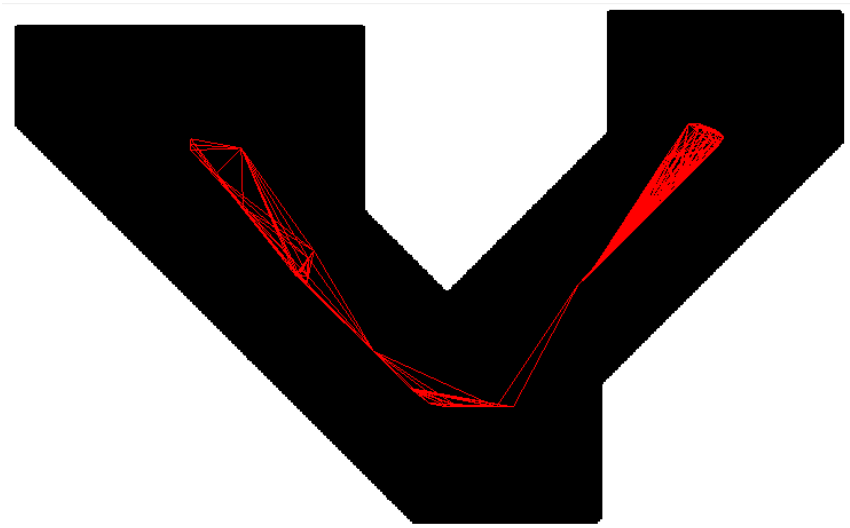
Atlikus algoritmo žingsnius „Paveikslėlio subraižymas ir vidurio taškų suradimas“, „Vidurio taškų sujungimas į linijų grupes“, „Linijų grupių sukirtimas“ operacijas buvo gauta aibė taškų, priklausančių figūros vidurio linijai. Prieš išskiriant tikrąją figūros vidurio liniją reikia nustatyti, kurie taškai tarpusavyje gali jungtis, o kuriuos taškus skiria figūros siena (žr. Pav. 2.15) ir jų jungti negalima – tai matomumo grafo (angl. *visibility graph*) konstravimo uždavinys (žr. šaltinį [10]).

Optimizacija

Matomumo grafo konstravimo sudėtingumas yra $O(n^2)$, kur n – grafo viršūnių kiekis, išskiriamos atminties kiekis - $O(n^2)$. Algoritmo realizacijoje iš kiekvienos grafo viršūnės brėžiama linija į kitą viršūnę, naudojant Bresenhamo algoritmą ir tikrinama ar kiekvienas pikselis tarp dviejų taškų priklauso figūrai (žr. Pav. 2.15). Linijos brėžimas atima daug laiko, todėl įvedamas parametras „`MAX_INT_PTS_JOIN_DISTANCE`“ (reikšmė pagal nutylėjimą yra lygi „0“ t.y. optimizacija išjungta). Parametras nurodo maksimalų atstumą tarp taškų iki kurio dar atliekamas matomumo tikrinimas, brėžiant liniją, o viršijus šią maksimalaus atstumo reikšmę iškart pažymima, kad taškai vienas kito „nemato“. Deja šio parametro optimali reikšmė nėra apskaičiuojama automatiškai ir turi būti įvedama paties vartotojo. Jei parametro reikšmė yra per maža, galutinė vidurio linija gali gautis trūki ar tam tikri vidurio taškai praleisti, o jei reikšmė per didelė, algoritmo veikimas žymiai nepaspartės. Palyginimui galima pateikti du paveiksliukus su sukonstruotais matomumo grafais , Pav. 2.23 optimizacija nėra pritaikyta, Pav. 2.24 - pritaikyta.



Pav. 2.23 Sukonstruotas matomumo grafas, be optimizacijos



Pav. 2.24 Sukonstruotas matomumo grafas su pritaikyta optimizacija

Yra laikoma, kad maksimali vidurio linijos taškų jungimo atstumo reikšmė yra parinkta tinkamai, tada, kai iš sukonstruoto matomumo grafo, kuriam pritaikyta optimizacija, gauta vidurio linija yra identiška vidurio linijai, gautai iš matomumo grafo, kuriam ši optimizacija pritaikyta nebuvo.

lentelė 2.6 Matomumo grafo konstravimo pseudokodas

```
procedure constructVisibilityGraph(p, graphPoints, MAX_INT_PTS_JOIN_DISTANCE)
// in: p – duotasis paveikslėlis
// in: graphPoints – grafą sudarančių taškų sąrašas (2xN matrica)
// in: MAX_INT_PTS_JOIN_DISTANCE – maksimalaus atstumo tarp dviejų taškų
dydis, kuriam esant dar atliekamas jungumo patikrinimas ir taškai jungiami
// out: visibilityGraph – išretinta matomumo grafo matrica (angl. sparse
matrix)
// local: pointsStart - briaunų pradžių masyvas
// local: pointsEnd – briaunų pabaigų masyvas
// local: weights – atstumas tarp dviejų grafo briaunų (briaunos svorio
reikšmė)
begin
   pointsStart = allocate(0);
  pointsEnd = allocate(0);
  weights = allocate(0);

  for i = 1 : size(graphPoints , 2)
     for j = i + 1 : size(graphPoints , 2)
       weight = calculateDistance(graphPoints(1, i),
                          graphPoints(2, i),
                          graphPoints(1, j),
                          graphPoints(2, j));

        if (MAX_INT_PTS_JOIN_DISTANCE == 0 || …
                   weight <= MAX_INT_PTS_JOIN_DISTANCE) && …
                   canPointsSeeEachOther(p, graphPoints(1, i),…
                   graphPoints(2, i), graphPoints(1, j), graphPoints(2, j))

            pointsStart.add( i );
            pointsEnd.add( j );
```

```
            weights.add( weight );
            weights.add( weight );

            pointsStart.add( j );
            pointsEnd.add( i );
        end if
    end for
  end for

  visibilityGraph = sparse(pointsStart,pointsEnd,weights);
   return [visibilityGraph];
end
```

2.3.5. Vidurio linijos išskyrimas iš matomumo grafo

Vidurio linijos išskyrimo idėja – matomumo grafe yra labai daug vidurio taškų, kurie nutolę mažu atstumu vienas nuo kito, todėl pritaikius minimalaus dengiančio medžio algoritmą (angl. *minimal spanning tree*) matomumo grafui, visi šie gretimi taškai yra sujungiami, taip suformuojant figūros vidurio liniją (žr. Pav. 2.25).

Buvo pasirinktas Kruskalo metodas minimalaus dengiančio medžio radimui. Viena iš priežasčių yra ta, jog jei apdorojamos figūros matomumo medis nėra jungus, tai algoritmas vis vien randa minimaliuosius dengiančius medžius kiekvienai nejungiai matomumo grafo medžio daliai. Kruskalo metodo panaudojimas taip pat leidžia analizuoti kelias nesusijusias figūras, esančias viename paveikslėlyje, to nebūtų galima pasiekti naudojant Primo minimalaus dengiančio medžio radimo metodą, realizacijoje naudojamam duomenų modeliui. Kruskalo metodo sudėtingumas $O(e + x \cdot \log n)$, kur e – grafo briaunų kiekis, n – viršūnių kiekis, x – briaunų kiekis medyje, kurios yra trumpesnės arba lygios ilgiausiai minimaliame dengiančiame medyje esančiai briaunai (pagal šaltinį [11]).

Iš kitos pusės, dėl šio metodo panaudojimo kyla skyriuje „Pradinių duomenų apribojimai“ nurodyti reikalavimai figūros formai. Taip yra dėl to, jog minimalus dengiantis medis negali turėti ciklų, todėl figūroms, kurios turi daugiau nei vieną kontūrą, vidurio linija yra trūki.

Pav. 2.25 Rasta vidurio linija (balta spalva)

2.4. Algoritmo sudėtingumo įvertinimas

Sudėtingumas blogiausiu atveju $O\left(\sum_{i=1}^{\frac{90}{brž}}(n_i \cdot m_i \cdot n_{i\,max} \cdot m_{i\,max}) + vid^2\right)$

(logaritminiai ir linijiniai sudėtingumai nebuvo įtraukti)

Kur i – i-toji linijų grupių sąrašų sukirtimo taškų paieškos operacija, $brž$ – linijų braižymo žingsnis, n_i – pirmo linijų grupių sąrašo dydis, m_i – antro linijų grupių sąrašo dydis, $n_{i\,max}$ – didžiausias linijų grupės elementų kiekis pirmame linijų grupių sąraše, $m_{i\,max}$ - didžiausias linijų grupės elementų kiekis antrame linijų grupių sąraše , *vid* – išskirtas vidurio taškų kiekis.

Sudėtingumo įvertinimo dedamosios: $\sum_{i=1}^{\frac{90}{brž}}(n_i \cdot m_i \cdot n_{i\,max} \cdot m_{i\,max})$ – trečias algoritmo etapas „Linijų grupių sukirtimas“, vid^2 – ketvirtas algoritmo etapas „Matomumo grafo konstravimas“.

Bendro algoritmo atminties sunaudojimo vertinimas: $O\left(\frac{180}{brž} \cdot \left((w+h) \cdot \sqrt{w^2+h^2}\right) + \sum_{i=1}^{\frac{180}{brž}}\left(2 \cdot maxMidPts_i \cdot \frac{lnCnt_i}{maxMidPts_i - 1}\right) + intPts + 2 \cdot intPts^2\right)$

Čia w – paveikslėlio plotis, h – paveikslėlio aukštis, $brž$ - linijų braižymo žingsnis laipsniais; i – i - tasis linijų subraižymas; $lnCnt_i$ – braižančiųjų linijų kiekis i – tajame linijų subraižyme, $maxMidPts_i$ – maksimalus vidurio taškų kiekis vienoje linijoje i – tajame figūros subraižyme, $intPts$ – susikirtimo (vidurio) taškų kiekis.

Atminties sunaudojimo dedamosios: $\frac{180}{brž} \cdot \left((w+h) \cdot \sqrt{w^2+h^2}\right)$ – pirmas algoritmo etapas „Paveikslėlio subraižymas ir vidurio taškų suradimas“, $\sum_{i=1}^{\frac{180}{brž}}\left(2 \cdot maxMidPts_i \cdot \frac{lnCnt_i}{maxMidPts_i - 1}\right)$ – antras algoritmo etapas „Vidurio taškų sujungimas į linijų grupes“, $intPts$ - trečias algoritmo etapas „Linijų grupių sukirtimas“, $intPts^2$ – ketvirtas algoritmo etapas „Matomumo grafo konstravimas“, $intPts^2$ - penktas algoritmo etapas „Vidurio linijos išskyrimas iš matomumo grafo“

Algoritmui išskiriama labai daug atminties. Algoritmą būtų galima perrašyti ir prieš skaičiavimus rezultatų masyvams vietos neišskirti, tačiau dėl to nukentėtų veikimo sparta.

3. EKSPERIMENTINĖ DALIS

3.1. Eksperimento aprašymas

Eksperimente patobulintasis vidurio linijos paieškos algoritmas yra palyginamas su kitais algoritmais, skirtais rasti figūros vidurio liniją. Vertinamas veikimo greitis, bei linijos radimo tikslumas. Algoritmai su kuriais buvo atliktas palyginimas:

- „Zhang-Suen“ topologinio ploninimo algoritmas (kodas prieinamas šaltinyje [12]);
- Algoritmas, naudojantis Voronoi diagramas (kodas prieinamas šaltinyje [5]);
- Atstumų transformacijų algoritmas, naudojantis daugiatrafaretį greitąjį žingsniavimą (angl. *multistencil fast marching*) (kodas prieinamas šaltinyje [8]).

3.2. Naudojami kintamieji

Eksperimente naudojami parametrai:

- Linijų brėžimo žingsnis („EXAMINE_PICTURE_EVERY_NTH_LINE“) – kas kelinta paveikslėlį braižanti linija yra brėžiama. Pagal nutylėjimą šis parametras lygus „1“, t.y. visos linijos yra brėžiamos ir nei viena nepraleidžiama;
- Minimalus susikirtimo taško atstumas iki figūros kontūro („MIN_INT_PT_DIST_TO_BOUNDARY“). Pagal nutylėjimą šis parametras lygus „0“ t.y. susikirtimo taško atstumo iki sienos tikrinimas nėra atliekamas;
- Vidurio linijos taškų jungimo reikšmė („MAX_INT_PTS_JOIN_DISTANCE“). Pagal nutylėjimą šis parametras lygus „0“ t.y. atstumas tarp dviejų jungiamų vidurio taškų nėra ribojamas;
- Linijų vidurio taškų jungimo kampas laipsniais („JOIN_ANGLE“), žr. skyrių „Vidurio taškų sujungimas į linijų grupes“, kuriame yra plačiau paaiškintas šis dydis. Pagal nutylėjimą šis parametras lygus 60°.

Jei atliekant eksperimentus šių parametrų reikšmės nėra atskirai nurodytos, vadinasi parametrų reikšmės naudojamos tokios, kokios yra pagal nutylėjimą.

3.3. Eksperimento aplinka

Eksperimentas atliekamas, naudojant „Matlab 2016a“ programavimo terpę, bei „Matlab“ programavimo kalbą.

Operacinė sistema – Windows 7 x64 bitų

Procesorius – Intel Pentium T4500 dual core @ 2,30 GHz

Operatyvinė atmintis – 3GB DDR 3 @ 1066 MHz

Algoritmui, naudojančiam Voronoi diagramas, Voronoi diagramos gaunamos panaudojus „Qhull“ įrankį (http://www.qhull.org). Algoritmo „Matlab“ failai turi būti patalpinti „Qhull“ įrankio „bin“ aplanke.

Atstumų transformacijų algoritmui papildomai reikia sukompiliuoti „msfm2d.c“, „rk4.c“ failus, paleidžiant „compile_c_files.m“ funkciją su „Matlab 2016a“ įrankiu.

3.4. Veikimo tikslumo vertinimo metodika

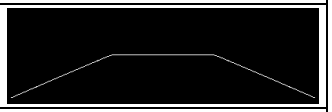
Algoritmo tikslumas vertinamas naudojant Sorenseno-Daiso (*Sørensen–Dice*) koeficientą (žr. formulė 3.1). Pirmas duomuo yra iš anksto paskaičiuota vidurio linija, tai - rezultatas, kurio tikimasi, antrasis duomuo - algoritmo apskaičiuota vidurio linija, kurios tikslumą reikia įvertinti.

$$QS = \frac{2|X \cap Y|}{|X| + |Y|}$$ **formulė 3.1 Sorenseno-Daiso koeficiento formulė**

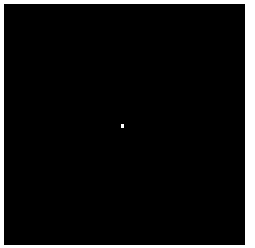
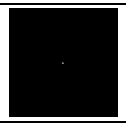
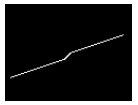
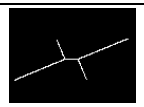
Formulės „formulė 3.1“ paaiškinimas: randamas abiejų vidurio linijų bendrų taškų kiekis, jis padauginamas ir „2“. Ši sandauga padalinama iš sumos, kur kiekvienas iš dėmenų – kiekvienai vidurio linijai priklausančių taškų kiekis. Koeficiento dydis gali svyruoti nuo 0 iki 1 imtinai, kur 0 – rodo, jog vidurio linijos nesutampa, o 1 – kad abi vidurio linijos yra identiškos.

3.5. Veikimo tikslumo eksperimentas

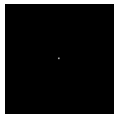
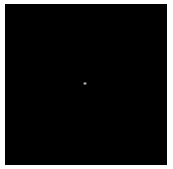
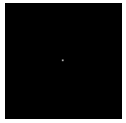
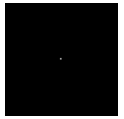
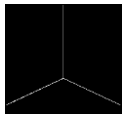
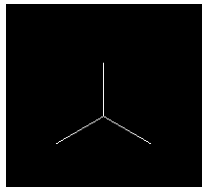
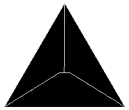
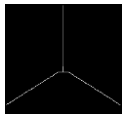
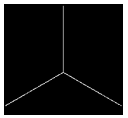
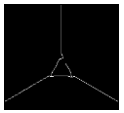
lentelė 3.1 Algoritmų rezultatų tikslumų palyginimas (trapecija, stačiakampis)

Nr.	Algoritmo pavadinimas	Parametrai	Figūra	Figūra ir algoritmo apskaičiuota vidurio linija	Rasta vidurio linija be figūros	Lauktas rezultatas	Sorenseno-Daiso koeficientas
1	Zhang Suen						0,233
	Gr.žingsniavimo						0,473
	Voronoi						0,833
	Sukurtas algoritmas						0,988
2	Zhang Suen						0,998
	Gr.žingsniavimo						0.819
	Voronoi						0,268
	Sukurtas algoritmas						1,000

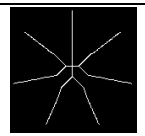
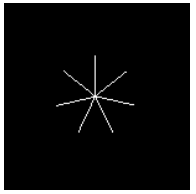
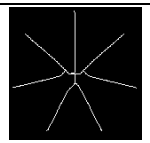
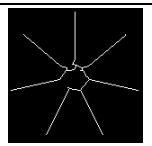
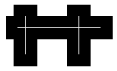
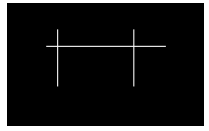
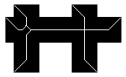
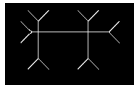
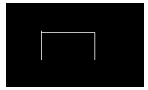
lentelė 3.2 Algoritmų rezultatų tikslumų palyginimas (kvadratas, lygiagretainis)

Nr.	Algoritmo pavadinimas	Parametrai	Figūra	Figūra ir algoritmo apskaičiuota vidurio linija	Rasta vidurio be figūros	Lauktas rezultatas	Sorenseno-Daiso koeficientas
3	Zhang Suen						1,000
	Gr.žingsniavimo	Faile skeleton.m sandauga maxD*2 pakeista į maxD*1.2					0,014
	Voronoi	Faile voronoiSkel.m trim kintamojo reikšmė pakeista iš 0 į 1					0,010
	Sukurtas algoritmas						1,000
4	Zhang Suen						0,032
	Gr.žingsniavimo						0,226
	Voronoi						0,140
	Sukurtas algoritmas	MIN_INT_PT_DIST-_TO_BOUNDARY = 1					0,923

lentelė 3.3 Algoritmų rezultatų tikslumų palyginimas (skritulys, trikampis)

Nr.	Algoritmo pavadinimas	Parametrai	Figūra	Figūra ir algoritmo apskaičiuota vidurio linija	Rasta vidurio linija be figūros	Lauktas rezultatas	Sorenseno-Daiso koeficientas
5	Zhang Suen						1.000
	Gr.žingsniavimo	Faile skeleton.m sandauga maxD*2 pakeista į maxD*1					0,036
	Voronoi	Faile voronoiSkel.m trim kintamojo reikšmė pakeista iš 0 į 1.5					1,000
	Sukurtas algoritmas						1,000
6	Zhang Suen						0,225
	Gr.žingsniavimo	Faile skeleton.m sandauga maxD*2 pakeista į maxD*1.9					0,214
	Voronoi	Faile voronoiSkel.m trim kintamojo reikšmė pakeista iš 0 į 2					0,499
	Sukurtas algoritmas	JOIN_ANGLE = 89					0,191

lentelė 3.4 Algoritmų rezultatų tikslumų palyginimas (žvaigždė, figūrų jungimas)

Nr.	Algoritmo pavadinimas	Parametrai	Figūra	Figūra ir algoritmo apskaičiuota vidurio linija	Rasta vidurio linija be figūros	Lauktas rezultatas	Sorenseno-Daiso koeficientas
7	Zhang Suen						0,178
	Gr.žingsniavimo	Faile skeleton.m sandauga maxD*2 pakeista į maxD*1.5					0,077
	Voronoi						0,351
	Sukurtas algoritmas						0,087
8	Zhang Suen						0,993
	Gr.žingsniavimo	Faile skeleton.m sandauga maxD*2 pakeista į maxD*1					0,770
	Voronoi	Faile voronoiSkel.m trim kintamojo reikšmė pakeista iš 0 į 1					0,614
	Sukurtas algoritmas	JOIN_ANGLE = 45					0,812

3.5.1. Veikimo tikslumo eksperimento apibendrinimas

Išanalizavus veikimo tikslumo rezultatus (žr. lentelė 3.1, lentelė 3.2, lentelė 3.3 ir lentelė 3.4), galima teigti, jog sukurtas algoritmas tiksliai veikia su paprastosiomis figūromis, tokiomis, kaip skritulys, stačiakampis gretasienis, lygiagretainis, keturkampis bei trapecija. Prastai randamos vidurio linijos iš trikampių sudarytoms figūroms (trikampis, žvaigždė). Pagal paskutinės lentelės (žr. lentelė 3.4 figūra nr. 8) duomenis, taip pat galima teigti, jog algoritmas nėra jautrus mažiems figūros topologijos pokyčiams.

3.6. Veikimo spartos eksperimentai

Šiame skyriuje atliekami trys algoritmų spartos palyginimo eksperimentai. Pirmasis eksperimentas yra skirtas nustatyti algoritmų spartos priklausomybę nuo plačiausios piešinėlyje esančios figūros dalies. Antrasis - skirtas įvertinti algoritmo veikimo spartą, kai iš paveikslėlio figūros gaunamas didelis kiekis tolygiai pasiskirsčiusių vidurio linijos taškų. Trečiasis – skirtas nustatyti kiek daugiausiai veikimo spartos galima išgauti iš skirtingų figūrų, kai didinamas linijos brėžimo žingsnis ir žiūrima ar vidurio linija neviršija nustatytos linijos panašumo koeficiento reikšmės.

3.6.1. Veikimo spartos priklausomybės nuo plačiausios figūros dalies eksperimentas

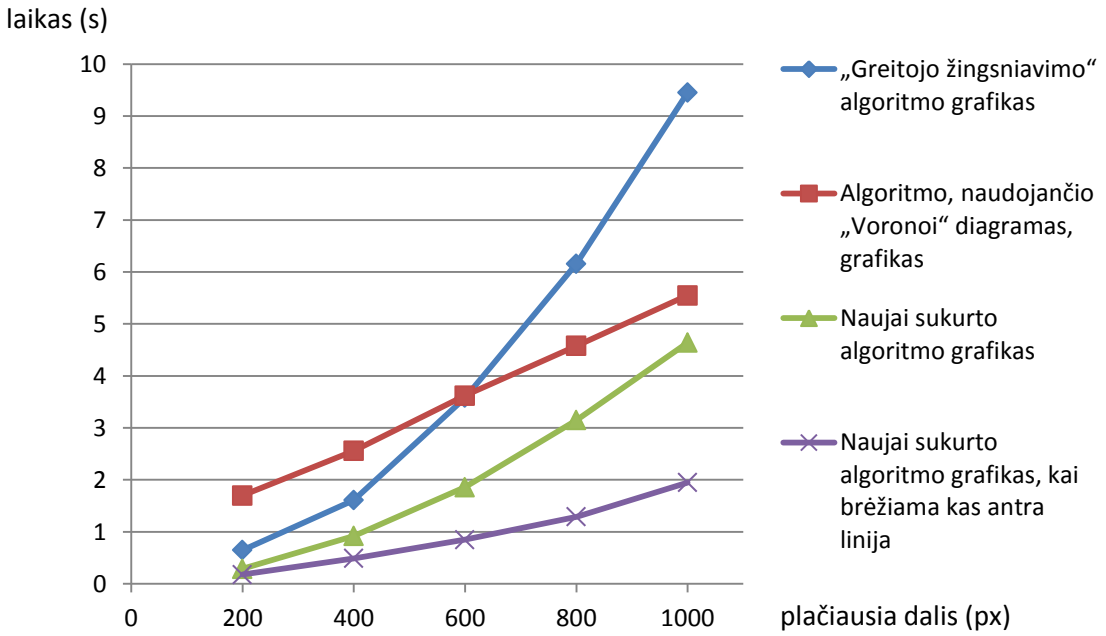
Su kiekvienu algoritmu nagrinėjama po 5 skirtingo dydžio kvadratus, kurių išmatavimai atitinkamai yra 200*200px, 400*400px, 600*600px, 800*800px, 1000*1000px. Priklausomybės nuo plačiausios paveikslėlio dalies pateikiamos sekančioje lentelėje (žr. lentelė 3.5) bei diagramoje „lentelė 3.6“. Prieš atliekant šį eksperimentą greitojo žingsniavimo algoritmui buvo pakeistas vienas iš šio algoritmų parametrų: faile skeleton.m sandauga „maxD*2“ pakeista į „maxD*1.2“. Voronoi algoritmui faile voronoiSkel.m pakeista „trim“ kintamojo reikšmė iš „0“ į „1“.

lentelė 3.5 Algoritmų veikimo laikų (s) priklausomybės nuo plačiausios figūros dalies

Plotis / Algoritmas	200px	400px	600px	800px	1000px
Zhang Suen	11,24	86,45	296,78	681,23	1363,21
Gr.žingsniavimo	0,65	1,61	3,58	6,16	9,45
Voronoi	1,70	2,56	3,62	4,58	5,55
Sukurtas algoritmas	0,29	0,92	1,86	3,15	4,64
Sukurtas algoritmas brėžiant kas antrą liniją	0,18	0,49	0,85	1,29	1,95

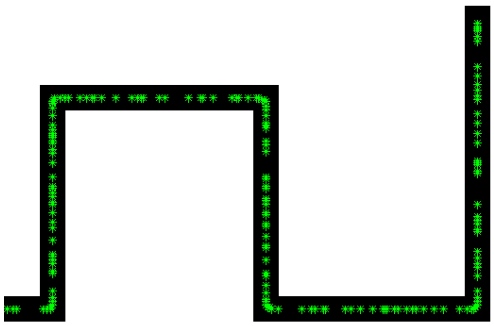
Geriausiai pasirodė naujai sukurtas algoritmas. Naudojant nustatymus pagal nutylėjimą šis algoritmas veikė 99,6% greičiau nei Zhang Suen algoritmas, 50,8% greičiau nei „Greitojo žingsniavimo“ algoritmas, 16,3% greičiau nei algoritmas, naudojantis Voronoi diagramas, kai vidurio linijos buvo ieškoma figūrai, kurios plačiausia dalis 1000 pikselių. Brėžiant kas antrą liniją algoritmo veikimo sparta pagerėjo papildomais ~57%, lyginant su laiku, kuris buvo gautas naudojant parametrus pagal nutylėjimą. Blogiausiai pasirodė Zhang Suen algoritmas, kurio veikimo sparta tiesiogiai priklauso nuo plačiausios figūros dalies. Į sekančią diagramą (žr. lentelė 3.6) Zhang Suen algoritmo rezultatai nebus įtraukti, kadangi jie per daug skiriasi nuo likusiųjų algoritmų rezultatų.

lentelė 3.6 Algoritmų veikimo laikų (s) priklausomybės nuo plačiausios figūros dalies grafikas



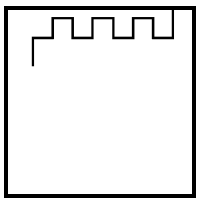
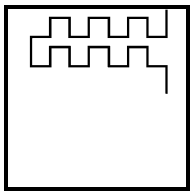
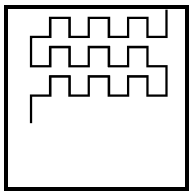
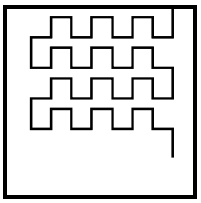
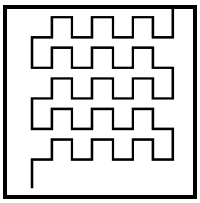
3.6.2. Veikimo spartos priklausomybės nuo figūroje rastų vidurio linijos taškų kiekio eksperimentas

Šiame eksperimente visiems algoritmams bus pateikiama penkių 1000*1000px paveikslėlių seka, kurioje kiekvieno sekančio elemento paveikslėlis yra papildytas fragmentu, paimtu iš pirmojo paveikslėlio (žr. lentelė 3.7). Naujai sukurtas algoritmas šiems paveikslėliams išskiria vidurio linijos taškus, kurie yra beveik tolygiai pasiskirstę (žr. Pav. 3.1). Eksperimento tikslai – įvertinti naujai sukurto algoritmo veikimo spartos priklausomybę nuo išskirtų vidurio taškų kiekio, bei palyginti veikimo spartą su kitais algoritmais, kai rasti vidurio taškai yra pasiskirstę tolygiai.



Pav. 3.1 Beveik tolygus vidurio linijos taškų pasiskirstymas

lentelė 3.7 Pradiniai algoritmų duomenys

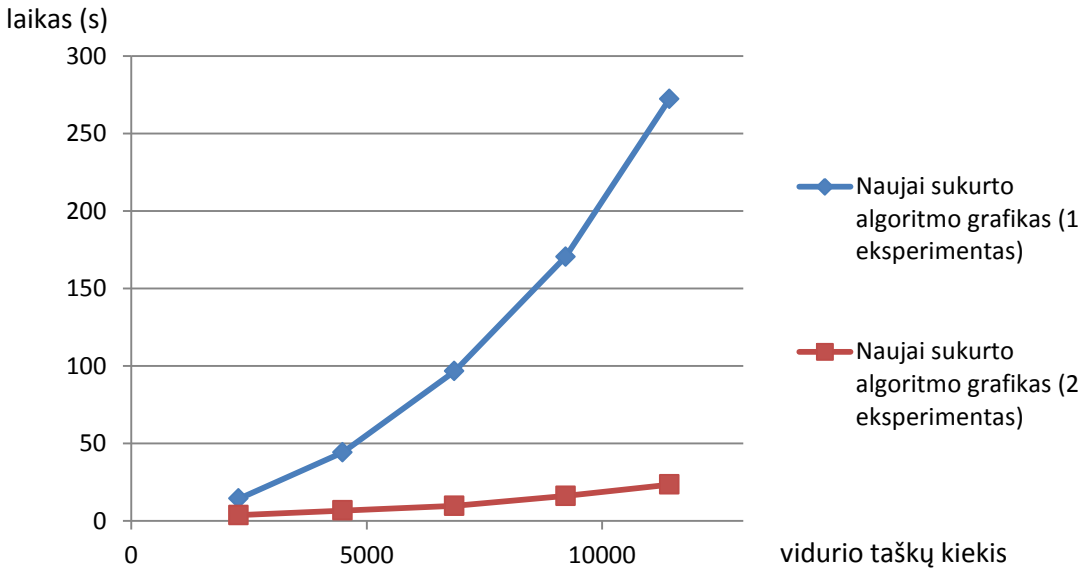
Nr.	**1)**	**2)**	**3)**	**4)**	**5)**
Pav.					

lentelė 3.8 Naujai sukurto algoritmo veikimo spartos rezultatai

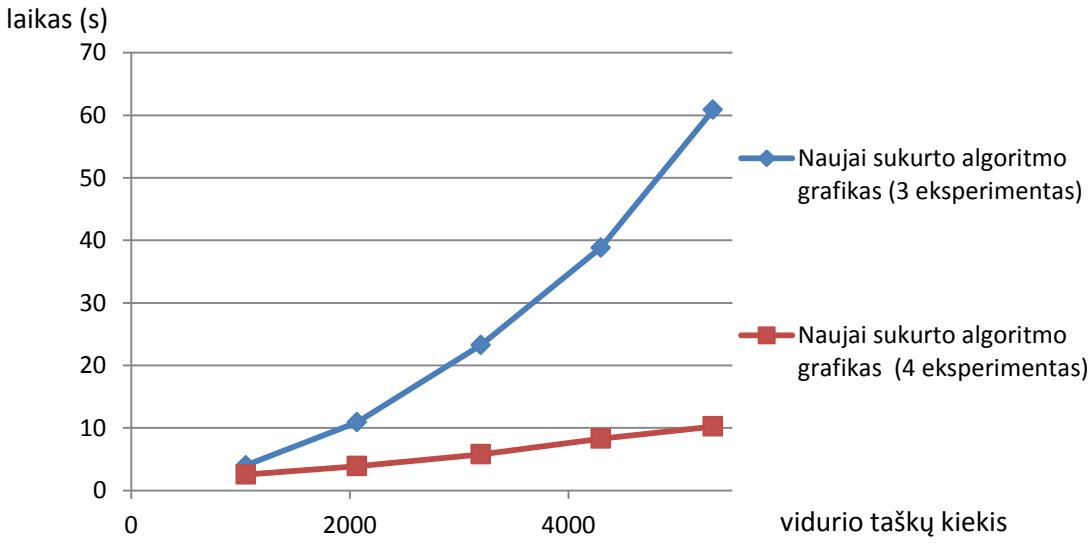
Eksperimento nr.	**Pav. nr.**	**Rastas vid. taškų kiekis**	**Optimizacija, bei komentaras**	**Laikas (s)**
1	1	2281	Naudojami parametrai pagal nutylėjimą.	14,36
	2	4492		44,21
	3	6866		96,63
	4	9230		170,45
	5	11433		272,22
2	1	2281	Parametro „MAX_INT_PTS_JOIN_DISTANCE“ reikšmė lygi „8“. Parametras parinktas taip, kad gauta vidurio linija visiškai sutaptų su eksperimento nr. 1 vidurio linija.	3,71
	2	4492		6,75
	3	6866		9,73
	4	9230		16,12
	5	11433		23,47
3	1	1048	Parametro „EXAMINE_PICTURE_EVERY_NTH_LINE“ reikšmė lygi „2“. Naudojant šį parametrą	4,02
	2	2065		10,90
	3	3199		23,26

	4	4296	prarandamas vidurio linijos tikslumas. Lyginant „5 pav.“ vidurio liniją su eksperimento „nr. 1“ „5 pav.“ vidurio linija, Sorenseno-Daiso koeficientas yra 0,995.	38,83
	5	5322		60,86
4	1	1048	„EXAMINE_PICTURE_EVERY_NTH_LINE“ = „2“; „MAX_INT_PTS_JOIN_DISTANCE“ = „100“. Antrasis parametras parinktas taip, kad gautos vidurio linijos visiškai sutaptų su eksperimento nr. 3 vidurio linijomis.	2,56
	2	2065		3,87
	3	3199		5,77
	4	4296		8,28
	5	5322		10,23

lentelė 3.9 Naujai sukurto algoritmo veikimo spartos (s) priklausomybės nuo išskirtų vidurio taškų grafikas (žr. lentelė 3.8 „1“ ir „2“ eksperimentus)



lentelė 3.10 Naujai sukurto algoritmo veikimo spartos (s) priklausomybės nuo išskirtų vidurio taškų grafikas (žr. lentelė 3.8 „3“ ir „4“ eksperimentus)

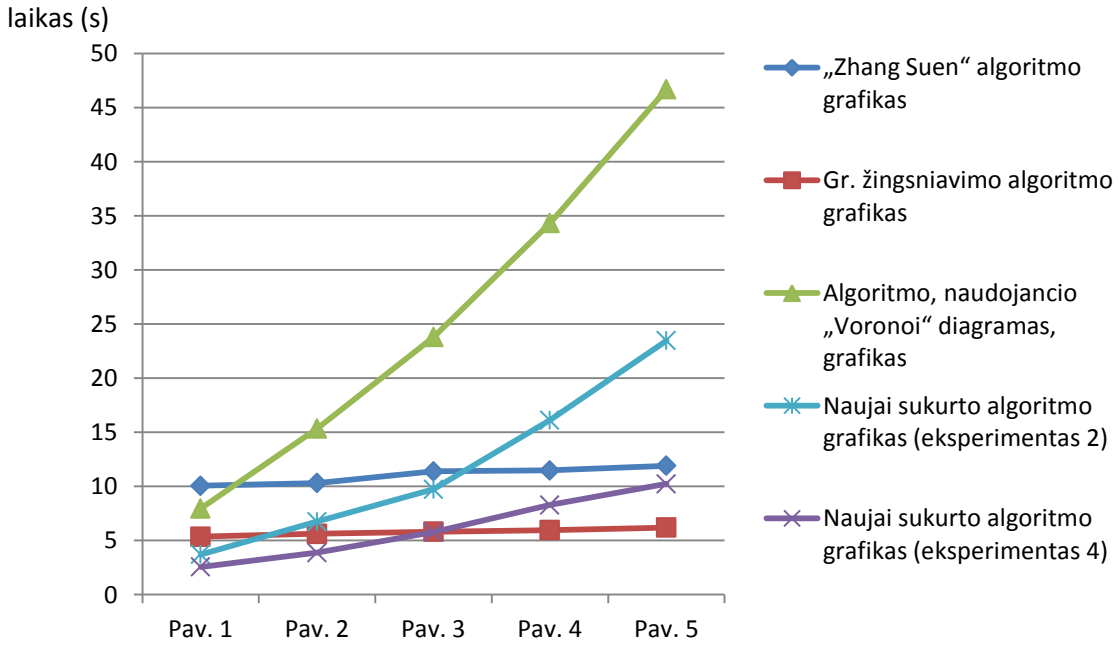


Iš diagramų „lentelė 3.9“ bei „lentelė 3.10“ grafikų galima pastebėti, kokia svarbi yra optimaliai parinkta vidurio linijos taškų jungimo („MAX_INT_PTS_JOIN_DISTANCE“) reikšmė. Pritaikius šį parametrą algoritmas ne tik smarkiai pagreitėja, tačiau pats veikimo spartos grafikas tampa panašus į tiesinės priklausomybės grafiką, o ne kvadratinės priklausomybės grafiką. Šis dėsnis galioja figūroms, kurių vidurio linijos taškai yra pasiskirstę tolygiai. Deja, bet šį dydį reikia įvesti pačiam vartotojui ir jis nėra automatiškai paskaičiuojamas ir kiekvienai figūrai yra vis kitoks.

Pagal lentelės „lentelė 3.8“ „1“ ir „2“ eksperimentų rezultatus, ieškant vidurio linijos 5-ajam paveikslėliui, algoritmo veikimo laiką pavyko pagerinti 91,4%, išlaikant nepakitusią vidurio liniją ir visiškai nemažinant išskirtų vidurio linijos taškų skaičiaus. Lyginant „1“ ir „4“ eksperimentų rezultatus galima pastebėti, jog algoritmas paspartėjo iki 96,2% , kai linijų brėžimo žingsnis buvo pakeistas į „2“ ir kai buvo parinktas optimalus vidurio linijos taškų jungimo parametras, nors buvo prarasta šiek tiek vidurio linijos tikslumo.

Toliau pateikiamas grafikas, kuriame lyginami visų likusių algoritmų rezultatai su „2“ ir „4“ eksperimentų rezultatais iš lentelės „lentelė 3.8“, kai pradiniai duomenys yra paveikslėliai iš lentelės „lentelė 3.7“.

lentelė 3.11 Algoritmų veikimo spartos (s) priklausomybių nuo figūros sudėtingumo grafikai

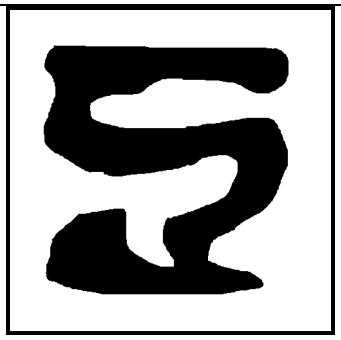
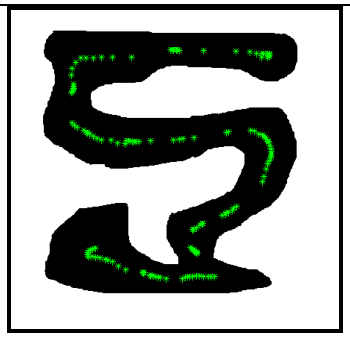
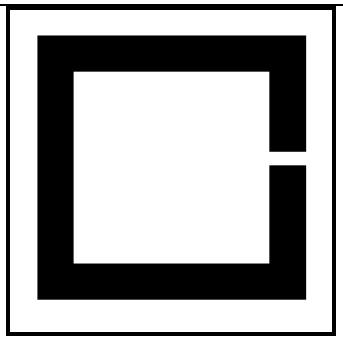
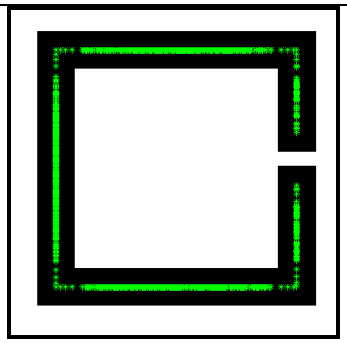


Palyginus lentelės „lentelė 3.11“ grafikus, galima teigti, jog algoritmas gali veikti greičiausiai tuomet, kai figūrai bus rasta mažai vidurio taškų. Vidurio taškų kiekį galima mažinti didinant linijų subraižymo žingsnį, tačiau dėl to gali nukentėti gautos vidurio linijos tikslumas.

3.6.3. Algoritmo veikimo spartos eksperimentas, kai keičiamas vien tik braižančiosios linijos žingsnis

Šiame eksperimente bus palaipsniui didinamas braižančiosios linijos žingsnis kiekvienai iš dviejų figūrų ir gautai vidurio linijai randamas Sorenseno-Daiso koeficientas, lyginant gautą vidurio liniją su vidurio linija, kai braižančiosios linijos žingsnis nebuvo pakeistas. Kai Sorenseno-Daiso panašumo koeficientas tampa mažesnis nei „0,6“, eksperimentas figūrai yra stabdomas. Toliau pateikiama lentelė (žr. lentelė 3.12), kurioje aprašyti du 500*500px išmatavimų pradinių duomenų paveikslėliai. Eksperimentu siekiama išsiaiškinti figūros vidurio linijos radimo greičio optimizavimo potencialą, kai vidurio linijų taškai pasiskirstę tolygiai bei grupėmis ir kai leidžiama tam tikra vidurio linijos tikslumo paklaida.

lentelė 3.12 Pradiniai trečiojo veikimo greičio eksperimento duomenys

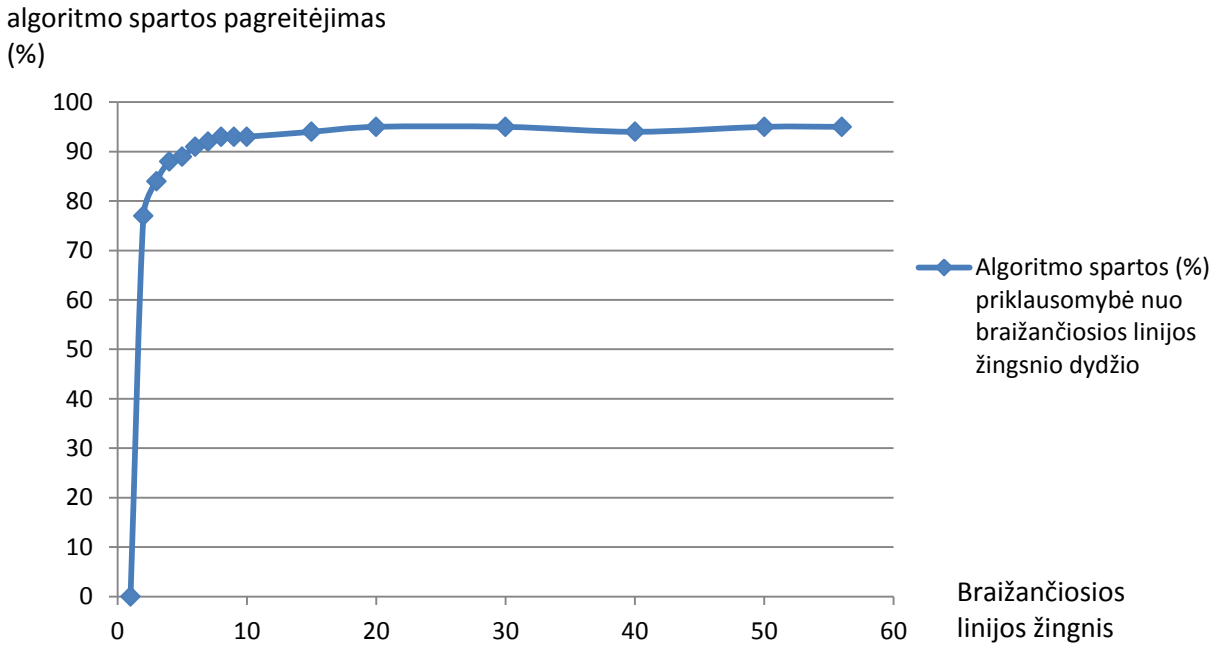
Pav. nr.	Paveikslėlis	Paveikslėlis ir išskirti vidurio linijos taškai	Vidurio linijos taškų pasiskir-stymas	Figūros sudėtin-gumas
1			Į grupes	Didelis
2			Tolygus	Mažas

lentelė 3.13 Tikslumo ir veikimo laiko priklausomybės nuo linijos brėžimo žingsnio lentelė

Pav. nr.	Linijos brėžimo žingsnis	Iškirtų linijos vidurio taškų kiekis	Santykinis iškirtų linijos vidurio taškų kiekio sumažėjimas, lyginant su pirmu rezultatu	Laikas (s)	Santykinis greičio pokytis lyginant su pirmojo rezultato laiku	Vid. linijos Sorenseno-Daiso panašumo koeficientas, lyginant su pirmojo rezultato vidurio linija
1	1	144	0%	1,25	0%	1,00
	2	129	10%	0,73	41%	0,65
2	1	1139	0%	6,18	0%	1,00
	2	444	61%	1,42	77%	0,99
	3	297	73%	0,94	84%	0,98
	4	227	80%	0,70	88%	0,98
	5	204	82%	0,62	89%	0,95
	6	186	83%	0,50	91%	0,95
	7	176	84%	0,46	92%	0,95
	8	166	85%	0,42	93%	0,95
	9	155	86%	0,40	93%	0,95
	10	155	86%	0,39	93%	0,92
	15	134	88%	0,37	94%	0,90
	20	123	89%	0,30	95%	0,87
	30	133	88%	0,30	95%	0,76
	40	132	88%	0,32	94%	0,71
	50	122	89%	0,28	95%	0,66
	56	106	90%	0,25	95%	0,62

Pagal lentelę „lentelė 3.13“ antrajai figūrai galima brėžti veikimo laiko spartos priklausomybės nuo braižančios linijos žingsnio grafiką.

lentelė 3.14 Algoritmo spartos priklausomybės nuo braižančiosios linijos žingsnio grafikas antrajai „lentelė 3.12“ figūrai



Atsižvelgus į diagramos „lentelė 3.14“ bei lentelės „lentelė 3.13“ duomenis, galima teigti, jog esant tolygiam ir glaustam vidurio linijos taškų pasiskirstymui, galima keisti braižančiosios linijos žingsnį, prarandant tik mažą dalį vidurio linijos tikslumo ir gaunant didelį veikimo spartos prieaugį. Taip pat buvo aptikta, jog egzistuoja tam tikra optimali braižančiosios linijos žingsnio reikšmė, kurią peržengus sparta didės nežymiai nors vidurio linijos tikslumas toliau mažės. Diagramoje nagrinėtam atvejui optimali braižančiojo linijos žingsnio reikšmė yra lygi „4“.

4. IŠVADOS

1. Algoritmas negarantuoja, kad vidurio linija visoms figūroms bus randama tiksliai. Algoritmas nepriekaištingai tiksliai arba tiksliausiai tarp lygintų algoritmų veikia tik su tam tikra figūrų imtimi. Taip pat algoritmui padavus tam tikras sudėtingas figūras gali būti neatsižvelgiama į silpnai išreikštas tų figūrų sritis.
2. Pagal atliktus eksperimentus, skirtus įvertinti algoritmo greitį, buvo išsiaiškinta, jog algoritmo spartai didesnę įtaką daro išskirtų vidurio linijos taškų kiekis, o ne figūros išmatavimų pokytis, todėl galima teigti, jog algoritmas apskaičiuos figūros vidurio liniją greičiau nei kiti algoritmai tada, kai figūra bus didelių išmatavimų ir turės mažai vidurio linijos taškų.
3. Algoritmo veikimo spartą galima pagerinti keičiant algoritmo parametrų reikšmes, pavyzdžiui, šiame darbe yra aprašytas atvejis, kai pakeitus vidurio linijos taškų jungimo parametro reikšmę algoritmas ėmė veikti 91% greičiau, visiškai nepraradus vidurio linijos tikslumo, o sukombinavus vidurio linijos taškų jungimo parametrą su linijos brėžimo žingsnio parametru algoritmas paspartėjo iki 96%, nors buvo prarasta maža dalis vidurio linijos tikslumo.
4. Buvo nustatyta, jog figūroms, kurių vidurio linijos taškai yra pasiskirstę glaustai ir tolygiai, galima rasti optimalų linijos brėžimo žingsnį, kuris vidurio linijos tikslumą sumažintų nežymiai, tačiau veikimo spartą padidintų gana reikšmingai. Viename iš aptartų atvejų, radus optimalų linijų subraižymo žingsnį, algoritmo veikimo sparta pagerėjo 88%, kai vidurio linijos Sorenseno-Daiso panašumo koeficientas, lyginant su pradine vidurio linija, buvo lygus 0,98.
5. Plėtojant šį algoritmą reikėtų surasti būdą kaip paskaičiuoti bent jau pradinį vidurio linijos taškų jungimo dydį kiekvienam paveikslėliui. Taip pat reikėtų įgyvendinti funkciją, kuri aptiktų klaidingai rastus figūros vidurio linijos taškus.

5. LITERATŪROS SĄRAŠAS

[1] Riškus, A.; Ostreika, A., „A simple centerline extraction approach for 2D polygons“, Kaunas: Multimedia Engineering department, Kaunas University of technology, 2016.

[2] Khanyile, N.P.;Tapamo, J.R.; Dube, E., „A Comparative Study of Fingerprint Thinning Algorithms,“ įtraukta *„Information Security South Africa Conference 2011“*, Johannesburg, 2011.

[3] Worboys, M.F.; Duckham, M., „GIS: A Computing Perspective,“ įtraukta *Presentation and algorithms*, New York, CRC Press, 2004, pp. 209-210.

[4] Preparata, F.P.;Shamos, M.I., „A catalog of Voronoi properties,“ įtraukta *Computational geometry - An introduction*, New York, Springer-Verlag, 1985, pp. 205-211.

[5] Bar-Sinai, Y., „Skeletonization using voronoi,“ 28 Gegužė 2010. [Tinkle]. Available: https://se.mathworks.com/matlabcentral/fileexchange/27543-skeletonization-using-voronoi. [Kreiptasi 30 Balandis 2017].

[6] Hassouna, M.S.; Farag, A.A., „Multistencils Fast Marching Methods: A Highly Accurate Solution to the Eikonal Equation on Cartesian Domains,“ *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, t. 29, nr. 9, pp. 1563-1574, 2007.

[7] Monneau, R., „Introduction to the Fast Marching Method,“ 31 Spalio 2010. [Tinkle]. Available: https://hal.archives-ouvertes.fr/hal-00530910/document. [Kreiptasi 1 Gegužės 2017].

[8] Kroon, D.J., „Accurate Fast Marching,“ 14 Sausis 2011. [Tinkle]. Available: https://www.mathworks.com/matlabcentral/fileexchange/24531-accurate-fast-marching/. [Kreiptasi 2 Gegužė 2017].

[9] Haverkort, H.J., „Introduction to bounding volume hierarchies,“ Utrecht universitetas, Utrecht, 2004.

[10] Ghosh, S.K., „Visibility Graphs,“ įtraukta *Visibility algorithms in the Plane*, Mumbai, Cambridge University Press, 2007, pp. 136-137.

[11] „graphminspantree,“ MathWorks, [Tinkle]. Available: https://www.mathworks.com/help/bioinfo/ref/graphminspantree.html. [Kreiptasi 9 gegužė 2017].

[12] Jin, L., „Skeletonization by Zhang-Suen Thinning Algorithm, Python and Matlab Implementation,“ 29 Spalis 2014. [Tinkle]. Available: https://github.com/linbojin/Skeletonization-by-Zhang-Suen-Thinning-Algorithm. [Kreiptasi 2 Gegužė 2017].

6. PRIEDAI

Prie ataskaitos pridedamas kompaktinis diskas, kuriame yra:

- Įgyvendintas plokščios figūros vidurio linijos radimo algoritmas
- Eksperimente naudoti paveikslėliai
- Algoritmai, su kuriais buvo lygintas sukurtas algoritmas