KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

MULTIMEDIJOS INŽINERIJOS KATEDRA

VARTOTOJIŠKOJI IR MOKOMOJI FRAKTALINIŲ VAIZDŲ KŪRIMO SISTEMA

Informacinių sistemų inžinerijos magistro baigiamasis darbas

Recenzentas
dr. doc. Bronius Paradauskas

Vadovas
dr. doc. Antanas Lenkevičius

Atliko
IFM 0/4 gr. stud.
Gytis Urbonavičius
2006.01.05

KAUNAS, 2006

TURINYS

ĮVADAS

Debesys - ne sferos, kalnai - ne kankorėžiai, pakrančių linijos - ne apskritimai, žievė neglodi, net šviesa nesklinda tiesia linija.

B. Mandelbrot

Fraktalinė geometrija verčia jus į viską pažvelgti kitaip …Jūs rizikuojate netekti savo naivaus debesų, miškų, galaktikų, lapų, plunksnų, gėlių, akmenų, kalnų, tekančios srovės, kilimų, plytų ir daugelio kitų dalykų suvokimo. Niekada daugiau jūs nesuvoksite šitų objektų taip, kaip interpretuodavote juos iki susipažinimo su fraktalais.

Michael Barnsley

Sąvokos „fraktalas“ ir „fraktalinė geometrija“ atsirado aštuntojo dešimtmečio pabaigoje, o nuo devintojo dešimtmečio vidurio jau ilgam įsitvirtino programuotojų ir matematikų žodyne. Terminas „fraktalas“ paimtas iš lotynų kalbos: „fractus“ reiškia nereguliarų fragmentą. Beje, šį terminą 1975 m. pasiūlė amerikiečių matematikas Benua Mandelbrotas (Benoit Mandelbrot) nereguliarioms, bet panašioms į save, struktūroms pažymėti. Fraktalinės geometrijos gimimas siejamas su 1977 m. pasirodžiusia B. Mandelbroto knyga „Fraktalinė gamtos geometrija“[5]. Knygoje panaudoti ir kitų mokslininkų, dirbusių šioje srityje 1875—1925 metų laikotarpiu (Dž.Kantoro (Georg Cantor), Dž.Peano (Giuseppe Peano), D.Hilberto (David Hilbert), H.Kocho (Helge von Koch), V.Sierpinskio (Waclaw Sierpinski), G.Žulia (Gaston Julia), F.Hausdorfo (Felix Hausdorff), A.Bezicovičiaus (Abram Besicovitch), P.Fatu (Pierre Fatou) mokslinių tyrimų rezultatai.

Svarbu tai, kad tik pastaruoju metu (B. Mandelbroto dėka) visi tie darbai buvo apjungti į vientisą sistemą. Visos figūros, kurias B.Mandelbrotas tyrė ir vadino fraktalais, turėjo savybę – nereguliarios, bet atsikartojančios savyje, t.y. žiūrint pro padidinamąjį stiklą į atskirą figūros fragmentą, pastarasis „primindavo“ visą figūrą. Be to, fraktalus (kaip aibes) sudaro be galo daug taškų, kurių tarpusavio išsidėstymas toks sudėtingas, kad neįmanoma nei perprasti, nei aprašyti jų „geometrijos“.

Fraktalų įvairovę patogu pateikti, atsižvelgiant į visuotinai priimtą jų klasifikaciją, būtent: geometriniai fraktalai, algebriniai fraktalai, stochastiniai fraktalai.

Geometriniai fraktalai vaizdžiausi. Dvimačiu atveju jie gaunami kokios tai laužtės (arba paviršiaus, trimačiu atveju), vadinamos generatoriumi, pagalba. Kiekviename algoritmo žingsnyje, laužtę sudaranti atkarpa keičiama laužte—generatoriumi (atitinkamai parinkus mastelį). Daugkartinio šios procedūros taikymo rezultatas—geometrinis fraktalas. Kompiuterinėje grafikoje geometriniai fraktalai yra būtina priemonė medžių, krūmų, kalnų (kranto) linijos ir kitiems vaizdams sukurti. Dvimačiai geometriniai fraktalai naudojami erdvinėms tekstūroms (objektų paviršiaus piešiniams) išgauti.

Algebriniai fraktalai—tai pati didžiausia fraktalų grupė. Jie generuojami (gaunami) netiesinių iteracinių procesų *n*—matėse erdvėse pagalba. Geriausiai išanalizuoti yra dvimačiai iteraciniai procesai. Interpretuojant tokį netiesinį iteracinį procesą kaip diskrečiąją dinaminę sistemą, galime naudotis šių sistemų teorijos terminija: fazinis portretas, nusistovėjęs procesas, atraktorius ir panašiai. Žinoma, jog netiesinės dinaminės sistemos turi kelias stabilias būsenas (padėtis). Būsena, kurioje atsidūrė dinaminė sistema po baigtinio iteracijų skaičiaus, priklauso nuo jos pradinės padėties. Todėl kiekviena stabili būsena (dar vadinama atraktoriumi) turi fiksuotą aibę pradinių padėčių, iš kurių sistema būtinai pereina į analizuojamą galutinę stabilią būseną. Tokiu būdu fazinė sistemos erdvė gali būti suskaidyta į atraktorių pritraukimo zonas. Jei fazinė erdvė—dvimatė erdvė, tai, parinkdami pritraukimo zonoms skirtingas spalvas, galime nesunkiai gauti spalvinį fazinį šios sistemos (iteracinio proceso) portretą.

Dar vieną fraktalų klasę sudaro stochastiniai fraktalai. Jie gaunami iteracinio proceso metu atsitiktinai keičiant vienus ar kitus proceso parametrus. Generuojami objektai labai panašūs į gamtinius objektus—nesimetrinius medžius, išraižytas kranto linijas ir kita. Dvimačiai stochastiniai fraktalai paprastai naudojami modeliuojant vietovės reljefą bei jūros paviršių.

Daugumos ženklių mokslo apie fraktalus laimėjimų nebūtų buvę įmanoma pasiekti be skaičiavimo matematikos priemonių, kurios tiesiogiai remiasi šiuolaikinių kompiuterių taikymu. Milžiniški kompiuterių skaičiavimo pajėgumai leido pakankamai išsamiai apžvelgti įvairias fraktalines struktūras bei suprasti jų atsiradimo priežastis. Kita vertus, gana dažnai teorinis šių struktūrų modeliavimas aplenkdavo eksperimentinius metodus, kuriais tiriami realūs sudėtingos formos gamtiniai objektai.

Fraktalinės geometrijos srityje taip pat daug yra nuveikę Klifordas A. Pikoveris (Clifford A. Pickover), Džeimsas Gleikas (James Gleick), G. O. Peitgenas (H. O. Peitgen) bei kiti. Šie mokslininkai stengėsi išplėsti naująją geometriją taip, kad ją būtų galima taikyti nuo vertybinių popierių biržos prognozių iki naujų atradimų teorinės fizikos srityje. JAV fraktalinių modelių tyrimu užsiima specialus centras NCSA (Nacionalinis papildymų superkompiuteriams centras).

Pagrindinis darbo tikslas - suprojektuoti ir sukurti vartotojišką/mokomąją fraktalinių vaizdų kūrimo sistemą, panaudojant naujas technologijas, kuri atliktų šias funkcijas:

- Fraktalo generavimas.
- Spalvų gamos pritaikymas.
- Fraktalo redagavimas.
- Fraktalo eksportavimas į kitus formatus.
- Meniu pritaikymas įvairiom aplinkom (namų, darbo ir t.t.)

Vienas iš pagrindinių reikalavimų, iškeltų šiam projektui, yra nesudėtingas visos sistemos bei jos komponentų naudojimas: kadangi ši sistema yra orientuota į namų vartotojus yra svarbu padaryti ją suprantamą ir prieinamą įvairaus amžiaus ir išsilavinimo asmenims.

Projekto užsakovas: Kauno Technologijos Universiteto Informatikos fakulteto Praktinės informatikos katedra. Atstovas: Docentas dr. Antanas Lenkevičius (antanas.lenkevicius@ktu.lt)

Projekto koncepciją analizavo, vartotojo sąsają projektavo ir pritaikė jai pagrindines sistemos funkcijas Gytis Urbonavičius (gytis.u@vici.lt).

1. ANALITINĖ DALIS

Fraktalinė geometrija — tai šiuolaikinės matematikos šaka, apjungianti klasikinę (Euklido) geometriją, topologiją, mato bei dinaminių sistemų teorijas. Benua Mandelbrotas — fraktalų geometrijos pradininkas — šią geometriją pavadino gamtos geometrija [5]. Būtent jisai surado nišą įvairiems „patologiniams" (klasikinei matematinei analizei „nepaklūstantiems") objektams bei reiškiniams (Kantoro dulkės, Peano kreivės, Vejerštraso funkcijos ir kt). Būtent B. Mandelbroto dėka fraktalinė geometrija tapo taikomuoju mokslu. Fraktalų panaudojimo sritys nuolat plečiasi. Jeigu prieš dešimt metų didžiausias dėmesys buvo skiriamas realaus pasaulio objektų (debesys, kalnai, jūros paviršius ir pan.) modeliavimui, tai dabar fraktalinės geometrijos, kaip matematinio įrankio, taikymo sričių spektras kur kas platesnis. Tai duomenų suglaudinimas (kompiuterinė grafika), techninė kainų analizės teorija (ekonomika), kietųjų kūnų paviršių analizė ir sintezė (fizika), biosensorinių tarpusavio poveikių tyrimas (medicina) ir t.t.

Žemiau glaustai pristatomas kontekstas, reikalingas fraktalo sąvokos įvedimui, pateikiami pakankamai detalūs fraktalų sintezės (generavimo) algoritmų aprašai bei komentuojami fraktalų tipai.

1.1. Tyrimo sritis ir objektas

Kadangi tiriamojo darbo objektas yra teoriniai ir praktiniai fraktalinių interpoliacinių funkcijų sudarymo aspektai, tai pagrindinį dėmesį toliau skirsime geometriniams fraktalams, gaunamiems taikant keruotąsias (afiniųjų) funkcijų sistemas.

1.1.1. Keruotųjų funkcijų sistemos ir jų atraktoriai (fraktalai)

Tarkime, turime metrinę erdvę (X,d) ir joje apibrėžtą transformaciją $f:X->X$. Pastebėsime, jog $f(S)=\{f(x)|\ x \in S$, kai $S \subset X$. Transformacija f yra apgręžiama, jeigu ji yra abipusiškai vienareikšmė ir $f(x) = X$. Šiuo atveju galima apibrėžti atvirkštinę transformaciją $f^{-1} : X \longrightarrow X$ tokią, kad $f^{-1}(y) = x$, ir $x \in X$ yra vienintelis taškas, su kuriuo $f(x) = y$.

Transformacijos $f : X \longrightarrow X$ iteracijomis pirmyn vadinamos transformacijos $f^{0n} : X \longrightarrow X$, apibrėžiamos lygybėmis:

$$f^{00}(x)= x, \quad f^{01}(x)= f(x), \quad f^{0(n+1)}(x)= f(f^{0n}(x)),$$ su visais $n = 0, 1, 2,.....$

Jeigu f yra apgręžiama, tai transformacijos f iteracijomis atgal vadinamos transformacijos $f^{0(-m)} : X \longrightarrow X$, apibrėžiamos lygybėmis:

$$f^{0(-1)}(x)= f^{-1}(x), \quad f^{0(-m)}(x) = (f^{0m})^{-1}(x),$$ su visais $m = 0, 1, 2,.....$

Praktiniuose taikymuose svarbiausia yra akcentuoti ryšį tarp transformacijas apibūdinančių formulių ir jų poveikyje atsirandančių geometrinių pasikeitimų (ištempimų, poslinkių, lenkimų ir pan.) erdvėse. Be to, svarbu tai, kaip transformacijos veikia ne atskirus erdvės X taškus, o jos poaibius.

Kalbant apie geometrinius fraktalus, jų prigimtį, paprastai imamos Euklido metrinės erdvės - (R, d), (R^2, d) arba $(R.^3, d)$ - ir joje veikiančios afiniosios transformacijos.

Bendru atveju, afinioji transformacija, veikianti (tarkime, dvimatėje) Euklido erdvėje (R^2, d), užrašoma taip - ω: $R^2 \longrightarrow R^2$ ir apibrėžiama lygybe –

$$\omega(x)= \omega(x_1,x_2) = (ax_1 + bx_2 + e,\ cx_1 + dx_2 + f),$$ su visais $x = (x_1,x_2) \in R^2$', čia a, b, c, d, e, f yra realieji skaičiai (afiniosios transformacijos ω parametrai). Afiniosios transformacijos turi daug svarbių geometrinių ir algebrinių savybių. Jų pagalba galima realizuoti posūkio, atspindžio, panašumo, pražulniąsias ir kitas transformacijas Euklido erdvėje (R^2, d), [4].

Jeigu su visais $x,y \in R^2$ teisinga nelygybė $d(\omega(x), \omega(y)) \le s \cdot d(x, y)$, $0 \le s < 1$, tai afinioji transformacija $\omega : R^2 \longrightarrow R^2$ vadinama suspaudžiančiaja, o realusis skaičius s, tenkinantis šią sąlygą, — afiniosios transformacijos suspaudimo koeficientu.

Imkime afiniųjų suspaudžiančiųjų transformacijų $\omega_i : R^2 \longrightarrow R^2$, $i = 1, 2,...,N$ rinkinį; atskirų afiniųjų transformacijų suspaudimo koeficientus pažymėkime s_i, $i=1,2,\ldots,N$. Tada Euklido erdvė (R^2, d) su joje veikiančių suspaudžiančiųjų afiniųjų transformacijų rinkiniu vadinama iteruotųjų funkcijų sistema ir žymima – IFS $\{R^2; \omega_1, \omega_2,... ,\omega_N\}$. IFS suspaudimo koeficientu laikomas skaičius $s = max\{s_1, s_2, ..., s_N\}$.

Apibrėžkime dar vieną metrinę erdvę ('H(R^2),h) tokiu būdu: 'H(R^2) - visų netuščių uždarųjų aibės R^2 poaibių aibė: *h* - metrika, nusakanti atstumą tarp bet kurių dviejų aibės 'H(R^2) elementų (aibės R^2 poaibių), būtent:

$$h(A, B) = max\ \{ d(A, B), d(B, A)\};$$

čia $d(A,B)$= max{mim{d(x,y)}}; $d(B,A\} = max\{min\{d(y,x)\}\}$.

Įvestoji erdvė (H(R^2),h) vadinama „fraktalų" erdve arba tiesiog fraktaline erdve.

Perkelkime IFS sudarančias afiniąsias transformacijas į erdvę $(H(R^2),h)$. Tada ω_i, : 'H(R^2) —>'H(R^2), apibrėžiama lygybe $\omega_i\ (B) = \{\ \omega_i(y)|\ y \in B\}, B \in$ 'H(R^2), yra suspaudžiančioji transformacija erdvėje (H(R^2),h) ir jos suspaudimo koeficientas s_i, i= l, 2, ..., N.

Vienintelis nejudamasis transformacijos W :'H(R^2)—> 'H(R^2) taškas A (A $\in$ 'H(R^2)) toks, kad

$$A = W(A)\bigcup_{i=1}^{N}\omega_i(A) = \lim_{n\to\infty} W^{0n}(B), \forall B \in 'H(R^2),$$

vadinamas IFS atraktoriurni (arba fraktalu), [4].

1.1.2. Fraktalų algoritmai

Yra žinomi ir plačiau praktikoje taikomi keli IFS (integracinių funkcijų sitema) atraktorių sintezės (generavimo) algoritmai. Tai – determinuotasis algoritmas, atsitiktinių iteracijų algoritmas bei „pabėgimo laiko“ algoritmas.

Žemiau pateikiami detalesni šių algoritmų aprašai.

Determinuotojo fraktalų (IFS atraktorių) generavimo algoritmo „veikimas“ tiesiogiai remiasi IFS atraktoriaus apibrėžimu.

Tarkime, kad {R^2; $\omega_1, \omega_2,..., \omega_N$ } yra iteruotųjų funkcijų sistema (IFS); čia ω_1 (i = 1, 2, ...,N) yra suspaudžiančiosios afiniosios transformacijos. Parenkame (pradinę) uždarąją aibę $A_0 \subset R^2$ ($A_0 \in$ 'H(R^2)) ir nuosekliai formuojame aibių seką {A_n} ; čia :

$$A_n = W^{0n}(A_0) = W(A_{n-1}) = \bigcup_{i=1}^{N}\omega_i(A_{n-1})$$

su visais n= 1,2,... . Įrodoma, jog tai (Koši) seka, kuri konverguoja į IFS atraktorių A. Kai n yra pakankamai didelis skaičius, aibė A_n vizualiai yra „artima“ aibei (fraktalui, IFS atraktoriui) A.

Visiškai nesvarbu, kokia imama pradinė uždaroji aibė A_0. Iteracinės procedūros rezultatas yra vienas ir tas pats — IFS atraktorius A. Kitaip sakant, aibė A pilnai nusakoma afiniųjų transformacijų ω_i , i = 1,2,...,N , veikiančių fraktalinėje erdvėje (H(R^2), h) , išraiškomis (2.1 pav.).

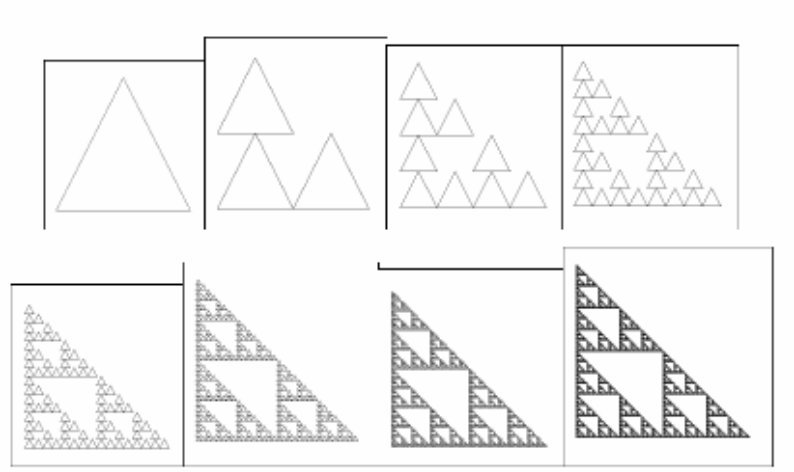
Pateiksime atsitiktinių iteracijų algoritmo aprašą. Tarkime, kad $\{R^2; \omega_1, \omega_2,..., \omega_N\}$ yra iteruotųjų funkcijų sistema. Kiekvienai suspaudžiančiajai afiniajai transformacijai ω_i (i=1,2,...,N) yra priskiriamas teigiamas skaičius (tikimybė) p_i taip, kad $p_1 + p_2 + ... + p_n = 1$. Jeigu

$$w_i = w_i\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \end{pmatrix} = A_i \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + T_i, i = 1,2,..., N,$$

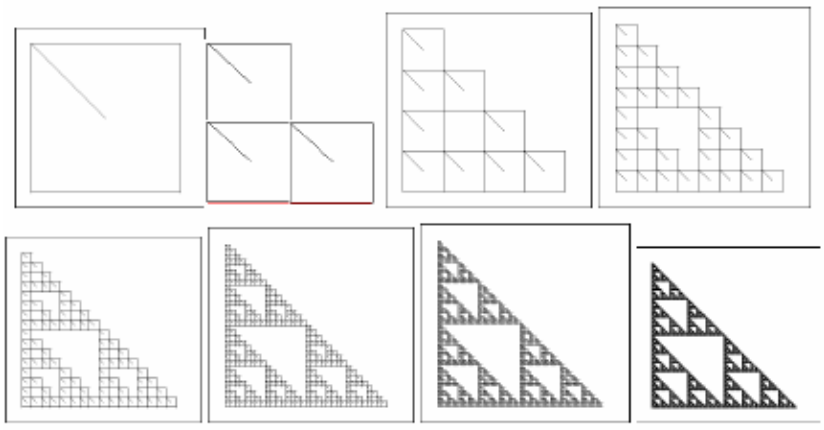
tai apytikslės tikimybių p_i reikšmės randamos iš formulės

$$p_i \cong \frac{|a_i d_i - b_i c_i|}{\sum_{j=1}^{N} |a_i d_i - b_i c_i|};$$

Jeigu su kuria nors reikšme i, $a_i d_i - b_i c_i = 0$, tai p_i yra pakankamai mažas teigiamas skaičius (tarkim, $p, = 0.001$).



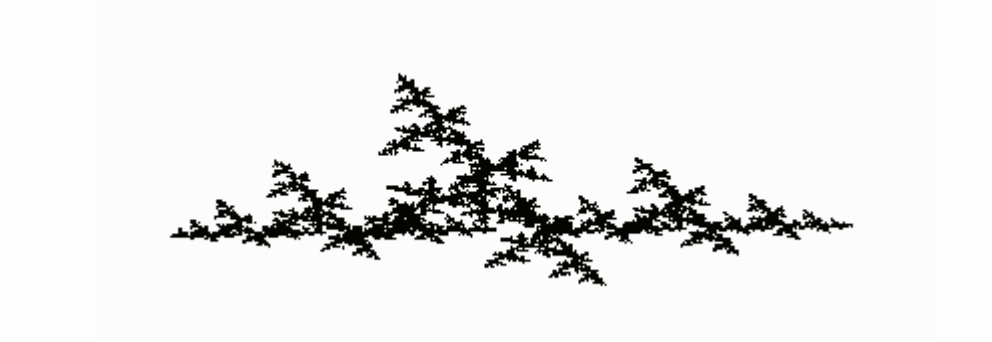
arba



1.1 pav. Determinuotojo IFS atraktoriaus sintezės algoritmo veikimo rezultatas

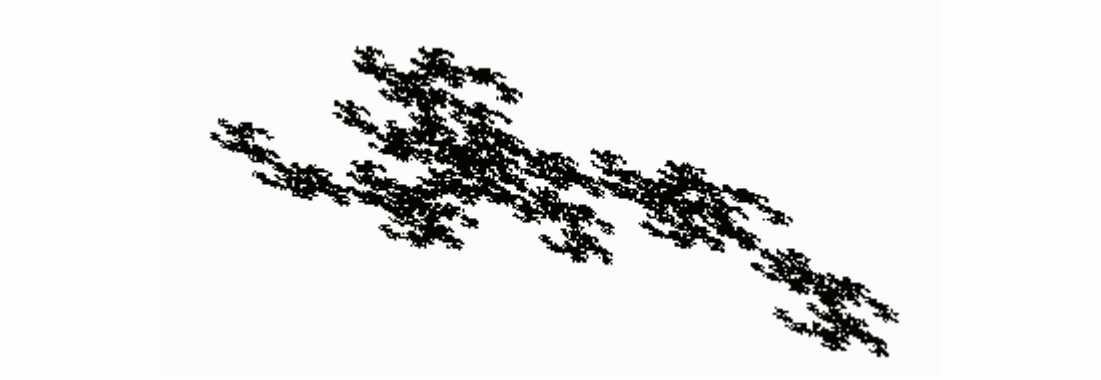
Parenkamas pradinis taškas x_0 ($x_0 \in R^2$) . Rekursyviai ir nepriklausomai konstruojama erdvės taškų seka $\{x_n\}^{\infty}_{n=0}$; $x_n \in \{ \omega_1(x_{n-1}), \omega_2(x_{n-1}),..., \omega_N(x_{n-1})\}$, su visais n = 1, 2,..., ir įvykio , jog x_n įgys reikšmę $\omega_i(x_{n-1})$, tikimybė yra lygi p_i, t.y. $P\{x_n = \omega_i(x_{n-1})\} = p_i$, i = 1,2,....,N.

Atsitiktinių iteracijų algoritmo „veikimas“ yra grindžiamas fraktalo (IFS atraktoriaus) taškams būdinga chaotiška judesio dinamika. Keletas IFS kodų bei juos atitinkančių fraktalinių vaizdų parodyti 1.2 pav.



(a) IFS { X; ω_1, ω_2, ω_3} atraktorius, gautas taikant atsitiktinių iteracijų algoritmą; IFS kodas:

{0.48 0.01 -0.01 0.47 0.80 -0.34 0.26
0.47 0.01 -0.01 0.47 -1.01 -0.28 0.25
-0.28 0.38 -0.96 -0.19 0.03 -0.06 0.48}



(b) IFS { X; ω_1, ω_2} atraktorius, gautas taikant atsitiktinių iteracijų algoritmą; IFS kodas:

{-0.46 -0.50 0.58 -0.58 0.46 -0.24 0.60
0.52 -0.45 0.45 0.32 -0.85 0.43 0.40}

1.2 pav. Iteruotųjų funkcijų sistemų atraktoriai (fraktaliniai vaizdai), gauti taikant atsitiktinių iteracijų algoritmą

„Pabėgimo laiko" algoritmas žinomas daugiau kaip teorinis rezultatas (idėja), jo praktinė realizacija iki šiol dar kelia tam tikrų problemų.

„Pabėgimo laiko" algoritmo „veikimas" remiasi tuo, jog IFS atraktorius (aibė *A)* yra transformacijos $S : 'H(R^2) \rightarrow 'H(R^2)$ atstūmiantysis nejudamasis taškas. Kitaip tariant taškų, esančių arčiau (metrikos *h* prasme) atraktoriaus *A,* orbitos ilgiau „užsibūna" skritulyje, negu orbitos taškų, labiau nutolusių nuo atraktoriaus.

Pagrindinė kliūtis, dėl kurios šis algoritmas sunkokai pritaikomas praktikoje, — tai nebuvimas afiniųjų transformacijų veikimo zonų atskyrimo kriterijaus.

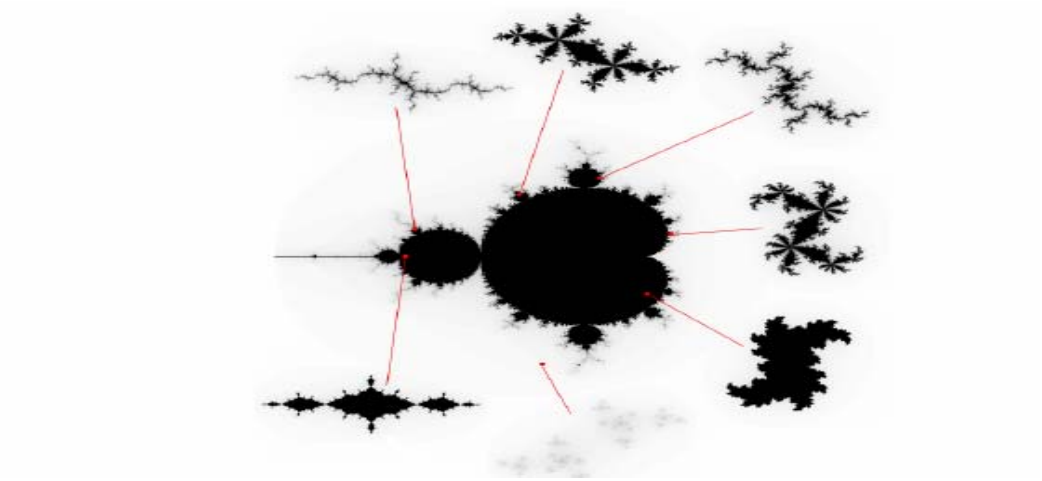
Šiame darbe fraktalinių interpoliacinių funkcijų analizei mes naudojome atsitiktinių iteracijų algoritmą. Būtent šis algoritmas pasižymi spartumu bei nebloga sintezuojamų fraktalinių vaizdų kokybe.

1.2.3 Fraktalų tipai

Fraktalinėje geometrijoje išskiriami šie fraktalų tipai — geometriniai fraktalai, algebriniai fraktalai, stochastiniai fraktalai.

Geometriniai fraktalai siejami su ankstyvuoju fraktalinės geometrijos (kaip mokslo šakos) vystymosi periodu. Faktiškai, tai iteruotųjų funkcijų sistemų, charakterizuojamų afiniųjų transformacijų, veikiančių įvairaus matavimo Euklido erdvėse, rinkiniais, atraktorių aibė. Šio tipo fraktalai yra vaizdžiausi ir, galima sakyti, geriausiai išanalizuoti.

Algebriniai fraktalai priklauso pačiai didžiausiai fraktalų grupei. Jie siejami su netiesinėmis dinaminėmis sistemomis, veikiančiomis kompleksinėje plokštumoje. Šio tipo fraktalų sintezei gana efektyviai panaudojamas anksčiau aptartas „pabėgimo laiko“ algoritmas (1.1.2 skyrelis). Vienas ryškiausių šio tipo fraktalų yra Mandelbroto aibė ir su ja susijusios Žulija aibės (1.3 pav.). Pastarosios generuojamos tiriant netiesinės parametrizuotos sistemos $\{C; f_\lambda\}, f_\lambda(z) = z^2 - \lambda,$ dinamiką. Čia pat pastebėsime, jog algebriniai fraktalai nėra pilnai ištirti, vis dar slepia savyje daug paslapčių.



1.3 pav. Mandelbroto ir Žulija aibės; {C; z^2- λ} (centre—Mandelbroto (parametro reikšmių λ, kurias atitinkančios Žulija aibės yra jungios) aibė)

1.1.2.1 Mandelbroto aibė

Mandelbroto aibė turbūt labiausiai žinomas fraktalas pasaulyje. Nepaisant to, mažai kur kalbama, kas tai yra ir kaip ją gauti. Mandelbroto aibė gaunama tokiu paprastu rekursiniu procesu:

$$Z(n) = Z(n\text{-}1)^2 + C$$

Čia konstanta C ir funkcija Z yra kompleksiniai skaičiai. Norėdami pavaizduoti Mandelbroto aibę kompleksinėje plokštumoje, turėtume imti visus plokštumos taškus ir su kiekvienu iš jų atlikti aukščiau aptartąjį rekursinį procesą, C laikydami tuo tašku. Paimtąjį tašką pažymime atitinkama spalva, priklausomai nuo to, kiek kartų atlikome rekursiją.

Divergavimo schema: xmin=-2.1, xmax=2.1, ymin=-2.1, ymax=2.1,

$$z_{n+1} = z_n^2 + c, \quad \mathrm{n} = 0,1,\cdots,\mathrm{M}. \ (\mathrm{M} = 50\text{-}5000)$$

Pradedame nuo $z_0 = 0$ ir randamas pirmas n: $|z_n|^2 > 4$. Pagal n reikšmę taške c priskiriama spalva, pvz. : jei n lyginis - raudona, nelyginis – juoda. Rezultatas: Mandelbroto fraktalas 6 pav.

Konvergavimo schema: xmin=-2.1, xmax=2.1, ymin=-2.1, ymax=2.1,

$$z_{n+1} = z_n^2 + c, \quad \mathrm{n} = 0,1,\cdots,\mathrm{M}. \ (\mathrm{M} = 50\text{-}5000)$$

Pradedame nuo $z_0 = 0$ ir randamas pirmas n: $|z_{n+1} - z_n|^2 < 10^{-10}$. Pagal n reikšmę taške c priskiriama spalva, pvz.: diverguojantiems - gelsvai rudas, konverguojantiems melsvas. Rezultatas: Mandelbroto fraktalas 7 pav.

xmin=-2.1, xmax=2.1, ymin=-2.1, ymax=2.1,

$$z_{n+1} = cz_n(1 - z_n), \quad \text{n} = 0,1,\cdots,\text{M}. \ (\text{M} = 500 - 5000)$$

Pradedame nuo z = (0, 0.1), taško c = (0.085, 1.073) aplinkoje. Rezulatas: Mandelbroto stiliaus fraktalas 1.4. pav.

1.4. pav. Mandelbroto fraktalas

Mandelbroto aibė yra nuostabios vienspalvės figūros, esančios skritulyje su centru (0, 0) ir spinduliu 2. Atlikdami rekursinį procesą su šios aibės taškais pastebėtume, kas Z (n) reikšmės didėja ir mažėja chaotiškai, tačiau niekada neauga į begalybę. Tuo tarpu už Mandelbroto aibės taškai anksčiau ar vėliau „nubėga“ į begalybę. Kaip greitai tai įvyksta, pažymi atitinkama taško spalva. Todėl Mandelbroto aibė apsupta nuostabių, nepaprastai sudėtingų spalvotų darinių, kurių ypatingas grožis atsiskleidžia juos daug kartų padidinus.

1.1.2.2 Julia (Žulija) aibė

Tarkime, kad iteracinė seka konverguoja $z_0 \in C$ taške . Tada tašką z_0 vadinsime iteracinės sekos konvergavimo tašku. Aibė, sudaryta iš iteracinės sekos konvergavimo taškų vadinsime šios iteracinės sekos konvergavimo aibe. Konvergavimo aibė žymėsime raide K. Tašką z_0, vadinsime iteracinės sekos divergavimo tašku, jeigu šiame taške iteracinės sekos modulis yra neaprėžtas. Aibę, kurią sudaro iteracinės sekos divergavimo taškai, vadinsime iteracinės sekos divergavimo aibe, kurią žymėsime raide *D*. Tarkime, kad $f : C \to C$ kokia nors polinominė transformacija, kurios laipsnis didesnis už vienetą. Tada šios transformacijos konvergavimo ir divergavimo aibes žymėsime simboliais K_f ir D_f .

Aibę

$$F_f = \{z \in C; \overline{\lim_{n\to\infty}} \mid f^n(z) \mid < \infty\}$$

vadinsime pilnąja transformacijos f Julijaus (Julia) aibe. Šios aibės sieną J_f vadinsime transformacijos f Julijaus aibė. Kitais žodžiais tariant, J_f yra aibė, kuri skiria D_f ir K_f aibes.

Tegul $[a,b] \subset X$, čia X kokia nors metrinė erdvė. Tarkime, kad intervalas $f([a,b]) \subset [a,b]$, tada šį intervalą vadinsime invariantiniu intervalu arba tiesiog invariantu, transformacijos f atžvilgiu. Kitaip tariant, invariantai yra aibės K_f poaibiai. Pasirodo, kad jei stačiakampis $[a,b] \times f([a,b]) \subset [a,b] \times [a,b]$, tai intervalas [*a, b*] yra invariantas transformacijos f atžvilgiu. Pastaroji savybė literatūroje vadinama *dėžės testu*. Pastaba. Galima įsitikinti, kad norint rasti maksimalų, kvadratinės transformacijos $f(x) = x^2 + c$ invariantinį intervalą, pakanka rasti šios transformacijos ir tiesės $y = x$ bendrus taškus. Tada maksimalus invariantinis intervalas bus apibrėžtas bendrų taškų abscisėmis. Aišku, kad šiuo atveju invariantinis intervalas sutampa su konvergavimo aibe. Pasirodo, kad jei iteruojame apibrėžtą kvadratinę transformaciją, tai konvergavimo aibė yra susijusi, jeigu $0 \in D_f$. Jeigu $0 \in K_f$, tai šios transformacijos konvergavimo aibė yra nesusijusi. Šiame skyriuje nagrinėsime atvaizdžių sekų (iteracinių sekų) konvergavimo bei divergavimo aibių struktūrą. Nagrinėsime transformacijos $f(z) = z^2$ iteracinę seką. Aišku, kad šios iteracinės sekos elementai priklauso aibei $F_f = \{z \in C; |z| \le 1\}$ $D_f = \{z \in C; |z| > 1\}$

$$\{z_{n+1} = z_n^2, n = 0,1,...\}$$

.

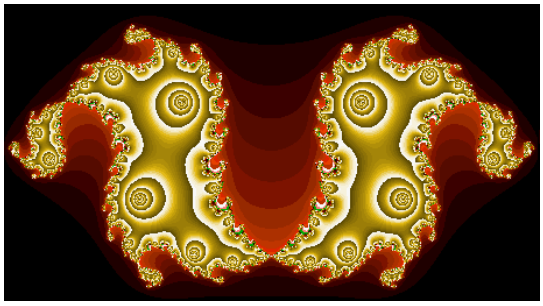
Transformacija $f(z) = z^2$ kompleksinį skaičių z transformuoja į skaičių z_1 taip, kad $|z_1| = |z|^2$, o skaičiaus z_1 argumentas yra dvigubai didesnis už pirmavaizdžio z argumentą. Nesunku suprasti, kad transformacijos $f(z)$ divergavimo aibę sudaro visi kompleksiniai skaičiai, kurių moduliai didesni už vienetą. Šios transformacijos konvergavimo aibė. Pastebėtina, kad ne visus konvergavimo aibės taškus transformacija veikia vienodai. Jei taško modulis mažesnis už vienetą, tai tada šio taško riba lygi nuliui. O jei taško modulis lygus vienam, tai žinome, kad šie taškai priklauso vienetiniam apskritimui. Bet jei taškas priklauso vienetiniam apskritimui, tai nebūtinai jis priklauso transformacijos konvergavimo aibei. Pavyzdžiui, taškas $z = 1$ yra konvergavimo aibės taškas. Deja, to paties negalime pasakyti apie tašką $z = i$. Šis taškas nėra konvergavimo aibės taškas, bet kadangi iteracinė seka šiame taške yra aprėžta, tai jis nepriklauso ir divergavimo aibei. Taigi jis priklauso transformacijos, Julijaus aibei. Atkreipkime dėmesį, kad visi kompleksinės plokštumos taškai, priklausantys vienetiniam apskritimui, yra invariantiški minėtosios transformacijos atžvilgiu, t.y. bet koks apskritimo taškas, veikiamas iteracinės sekos, pasilieka apskritime. Beje, transformacijos $f(z)$ iteracijų seka turi du nejudamus taškus: *0* ir *1*.

Julia vaizdai, *xmin=-2.1, xmax=2.1, ymin=-2.1, ymax=2.1,*

$$z_{n+1} = z_n^3 + z_n + p, \quad \text{n} = 0,1,\cdots,\text{M}. \ (\text{M} = 500 - 5000)$$

$$J_f = \{z \in C; |z| = 1\} \qquad K_f = \{z \in C; |z| \le 1, \{f^n(z)\} \textit{konverguoja}\}$$

Fiksuojamas p=(0.171,0) ir kaitaliojamos z pradinės reikšmės, spalvojami konvergavimo ir divergavimo taškai. Konvergavimo/divergavimo riba vadinama Julia aibe. Rezultatas: Julia stiliaus fraktalas 1.5. pav.



1.5. pav. Julia stiliaus fraktalas

1.1.2.3 Niutono aibė

Tarkime, kad duotas polinomas $F(z) = z^4 - 1$, $z \in C$. Žinoma, kad šis polinomas kompleksinių skaičių aibėje turi keturis nulius, t.y. egzistuoja taškai $a_1, a_2, a_3, a_4 \in C$ tokie, kad $F(a_i) = 0$, $i = 1, 2, 3, 4$. Beje, nesunkiai galime nustatyti, kad šie polinomo nuliai yra tokie skaičiai: $1, -1, i, -i$. Jei polinomo išraiška bendresnė, tuomet rasti nulius nėra taip paprasta. Niutono metodu galime rasti apytikslius polinomo nulius. Sakykime, kad duota dinaminė sistema $\{\overline{C}; f(z) = \frac{F(z)}{F'(z)}\}$

.

Transformacija $f(z)$ vadinama funkcijos $F(z)$ *Niutono transformacija*. Niutono metodo esmė- tinkamai parinkę pradinį tašką $z_0 \in C$ pasiekti, kad seka $\{f^n(z_0)\}$ konverguotų į funkcijos $F(z)$ nulį. Funkcijos $F(z) = z^4 - 1$ Niutono transformacija yra

$$f(z) = \frac{3z^4 + 1}{4z^3} .$$

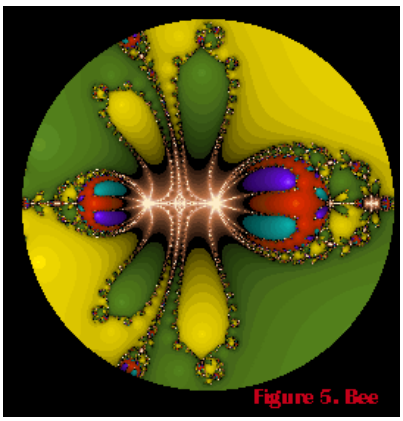
Niutono vaizdai: *xmin, xmax, ymin, ymax*

$$z_{n+1} = z_n - g(z_n)/g'(z_n), \quad \mathrm{n} = 0,1,\cdots,\mathrm{M}. \ (\mathrm{M} \cong 50)$$

(p nėra) Keičiamos z pradinės reikšmės ir spalvojami tik konvergavimo taškai. Pvz. :

$$g(z) = z^7 - 3z^5 + 6z^3 - 3z + 3$$

Rezultatas: Niutono stiliaus fraktalas 1.6. pav.



1.6. pav. Niutono stiliaus fraktalas

Stochastinius fraktalus, tam tikra prasme, galima būtų priskirti geometriniams fraktalams. Skirtumas tas, jog vietoj įprastinių iteruotųjų funkcijų sistemų imamos parametrizuotos sistemos. Iteracinio proceso metu keičiamos parametrų reikšmės ir tai daroma atsitiktinai (atsitiktine tvarka). Rezultatas — gaunami fraktalai labai panašūs į realaus pasaulio objektus („asimetriniai“ medžiai, atkarpytos pakrantės linijos ir pan.). Dvimačiai stochastiniai fraktalai naudojami modeliuojant vietovės reljefą, jūros paviršių ir kt.

1.2. Fraktalinės dimensijos sąvoka

Skaičiai, charakterizuojantys fraktalus ir skirti pastarųjų palyginimui, paprastai vadinami fraktalinėmis dimensijomis. Tai gana svarbios kiekybinės fraktalų charakteristikos. Jos leidžia įvertinti subjektyvų požiūrį apie tai, kaip „tirštai“ (tankiai) realaus pasaulio objektas (fraktalas) užpildo erdvę, kurios poaibis jis pats yra. Fraktalinė dimensija—tai objektyvi priemonė fraktalų palyginimui.

Istoriškai, svarbu apibrėžti Hausdorfo ir Bezicovičiaus matą bei dimensiją ir kartu atskleisti su jų apskaičiavimu susijusius sunkumus[4].

Imkime poaibį $A \subset R^n$; čia (R^n, d) - n-matė Euklido erdvė. Tegu

$|A| = diam(A) = sup\{d(x,y)|\ x,y \in A\}|$ žymi poaibio A diametrą.

Fiksuokime teigiamą skaičių $0 > \varepsilon$. Imkime skaičią poaibių A_i ($|A| \le \varepsilon$) aibę $\{A_i\}$ ir tokią, kad ji padengtų poaibį A, t.y.

$$A \subset \bigcup_{i=1}^{\infty} A_i ;$$

rinkinys (aibė) $\{ A_i\}$ vadinamas poaibio A ε-denginiu.

Toliau, tarkime, kad $p>0$, ir apibrėžkime dydį $H^p{}_\varepsilon(A)$ tokiu būdu:

$$H_\varepsilon^p(A) = \inf\left\{\sum_{i=1}^{\infty} |\ A_i\ |^p \ | \{A_i\} yra \cdot poaibio \cdot A \cdot \varepsilon - denginys \right\}$$

Faktiškai – tai minimizavimo uždavinys, kai stengiamasi minimizuoti dengiančiųjų aibių diametrų, pakeltų *p*—tuoju laipsniu, sumą (pagal visus įmanomus poaibio *A* ε-denginius). Kita vertus, dydis $H^p{}_\varepsilon(A)$, kaip funkcija, yra nemažėjantis, kai ε artėja prie nulio. Pereidami prie ribos, kai $0 \to \varepsilon$, apibrėšime Hausdorfo ir Bezicovičiaus matą (poaibiui *A*) tokiu būdu

$$H^p(A) = \lim_{\varepsilon \to 0} H^p_\varepsilon(A).$$

Dydžio $H^p(A)$ priklausomybės nuo parametro *p* grafikas rodo, jog yra kritinė *p* reikšmė, kai $H^p(A)$ „šoka“ nuo ∞ prie 0. Tą kritinį tašką atitinkanti *p* reikšmė vadinama Hausdorfo ir Bezicovičiaus dimensija (poaibiui *A*), t.y.

$$D_H(A) = \sup\{p \mid H^p(A) = \infty\} = \inf\{p \mid H^p(A) = 0\}$$

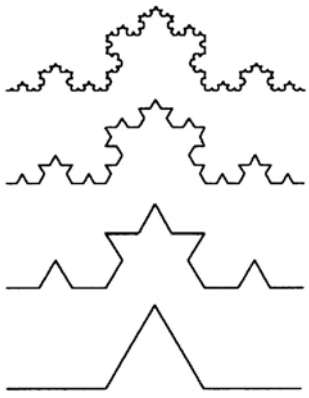
Pastebėsime, jog šią dimensiją apskaičiuoti netgi „paprastoms“ aibėms (fraktalams) yra pakankamai sudėtinga. Kaip pavyzdį panagrinėkime Kocho kreivę. Po *k* iteracijų kreivę sudaro 4^k segmentai, kurių kiekvieno ilgis 3^{-k}. Todėl kreivę galima padengti (tiksliai) 4^k aibėmis, kurių diametrai lygūs 3^{-k}. Tokiu būdu,

$$H^p_\varepsilon(A) = H^p_{3^{-k}}(A) = 4^k (3^{-k})^p.$$

Pertvarkykime šį rezultatą taip:

$H^p_\varepsilon(A) = H^p_{3^{-k}}(A) = 3^{k(D-p)}$; čia $D = ln4/ ln3$. Pereidami prie ribos, kai dengiančiųjų aibių diametrai artėja prie nulio (t.y. $k \to \infty$), gauname

$$H^p = \lim_{k \to \infty} H^p_{3^{-k}} = \begin{cases} \infty, & p < D \\ 1, & p = D \\ 2, & p > D \end{cases}$$



1.7 pav. Van Kocho kreivė (fraktalas)

Taigi, galima teigti, jog Van Kocho kreivės Hausdorfo ir Bezicovičiaus dimensija lygi $D = ln4 / ln3 \cong 1.262$.

Vėliau B. Mandelbrotas šią dimensiją pavadino fraktaline dimensija. Faktiškai, tai nauja erdvės (poaibių) matavimo priemonė. Daugelyje šaltinių fraktalinės dimensijos sąvokos įvedimui naudojamas šiek tiek supaprastintas požiūris (interpretacija).

Imkime metrinę erdvę (R^n,d). Tegul $A \in {}'H(R^n)$. Tarkime, kad $\varepsilon > 0$ ir $N(A, \varepsilon)$ žymi mažiausią uždarųjų rutulių $B(x, \varepsilon) = \{y \in R^n | d(x, y) \le \varepsilon\}$, sudarančių baigtinį poaibio A denginį, skaičių, t.y. $N(A, \varepsilon)$ – mažiausias sveikasis skaičius toks, kad

$$A \subset \bigcup_{i=1}^{N(A,\varepsilon)} B(x_i, \varepsilon);$$

čia $\{x_1,x_2,...,x_{N(A,\varepsilon)}\} \subset R^n$.

Fraktalinės dimensijos sąvoka įvedama, remiantis intuityvia idėja, jog aibė A ($A \subset R^n$) turi fraktalinę dimensiją D, jeigu

$$N(A, \varepsilon) \approx C\,\varepsilon^{-D}; \qquad (1.1)$$

čia: C — tam tikra konstanta; „≈ “ reiškia, jog $\lim_{\varepsilon\to\infty}(\ln f(\varepsilon) / \ln g(\varepsilon)$, kai $f(\varepsilon) \approx g(\varepsilon)$.

Dabar, išsprendę (1.1) „lygybę“ (D atžvilgiu), gauname

$$D \approx \frac{\ln N(A,\varepsilon) - \ln C}{\ln \frac{1}{\varepsilon}}.$$

Perėję prie ribos ($\varepsilon\to 0$), turime:

$$D = \lim_{n\to\infty} \frac{\ln N(A,\varepsilon)}{\ln\left(\frac{1}{\varepsilon}\right)}.$$

Ši riba vadinama poaibio A ($A \subset R^n$) fraktaline dimensija ir žymima $D(A) = D$.

Tarkime, kad $A \subset {}'H(R^m)$; be to, erdvė R^m padengta nesusikertančiais m—mačiais „kubiukais“ (dėžutėmis), kurių briaunos ilgis lygus $1/ 2^n$; tegul $N_n(A)$ žymi „kubiukų“, persidengiančių su poaibiu A, skaičių. Tada skaičius

$$D = \lim_{n\to\infty}\left\{\frac{\ln N_n(A)}{\ln\left(2^n\right)}\right\}$$

vadinamas poaibio A fraktaline dimensija, [4].

Būtent pastarasis teiginys sudaro (dažniausiai) pagrindą eksperimentiniam realaus pasaulio (fraktalinio) objekto dimensijos nustatymui (dimensijos įverčio gavimui).

Kaip pavyzdį panagrinėkime fraktalą A (1.8 pav. Sierpinskio trikampis). Nesunku pastebėti, jog:

$N_1(A)=3$, kai $1 = n$;

$$N_2(A) = 9, \text{ kai } n = 2, \text{ t.y. kai } \varepsilon_2 = 1/2^2 = 1/4 \ ;$$

$$N_3(A) = 3^3 = 27, \text{ kai } n = 3 \text{ ir t.t.}$$

Pagaliau

$$N_n(A) = 3^n, \text{ kai } \varepsilon_n = 1/2^n .$$

Taigi, fraktalinė Sierpinskio trikampio dimensija lygi:

$$D = \lim_{n\to\infty} \frac{\ln 3^n}{\ln 2^n} = \frac{\ln 3}{\ln 2} \cong 1.585$$

1.8 pav. Sierpinskio trikampis

1.3. Fraktalinės interpoliacinės funkcijos, jų specifika

Elementariosios funkcijos (tokios kaip sinusas, kosinusas, daugianariai ir panašiai.) sudaro tradicinio, visuotinai priimto eksperimentinių duomenų analizės metodo pagrindą. Tarkime, kad eksperimento metu matuojamos tam tikros realiosios funkcijos $F(x)$ reikšmės. Eksperimento rezultatas - duomenų rinkinys $\{(x_i, F_i)| \ i - 0, 1,....,N\}$; čia $F_i = F(x_i)$, $i = 0, 1, ...,N,$ ir $x_0 < x_1 <...< x_N$. Šie duomenys (kaip erdvės R^2 poaibis) pavaizduojami grafiškai ir analizuojami, t. y. stengiamasi parinkti galimai žemesnio lygio daugianarį (atskiru atveju, „laužtę“) tokį, kuris „neblogai modeliuotų“ duomenis segmente $[x_0, x_N]$, t.y. kurio grafikas „eitų“ per taškus (x_i, F_i) $i = 0, 1,...,N$.

Tačiau grafikos sistemoms kartais keliami šiek tiek didesni reikalavimai, būtent – galimybė modeliuoti realaus pasaulio (fizinius) objektus, tokius kaip debesys, kalnų masyvo profilis, kabantys stalaktitai ir panašiai. Šiems fiziniams objektams (sistemoms) būdinga tai, kad kiekvienos sistemos dalies struktūra dažnai „atkartoja“ visos sistemos struktūrą. Tai galima aiškinti tuo, jog sistemą įvairiuose jos lygiuose veikiančios ir formuojančios jėgos yra panašios. Čia geometrijos ir elementariųjų funkcijų čia jau nebepakanka.

Pateiksime fraktalinės interpoliacinės funkcijos sąvoką, parodysime, jog minėto tipo duomenims galima parinkti funkcijas, kurios (Hausdorfo metrikos prasme) būtų „artimos“

tiems duomenims; be to, įmanoma užtikrinti, kad fraktalinės interpoliacinės funkcijos grafiko dimensija sutaptų su eksperimento duomenų (grafiko) fraktaline dimensija.

Tarkime, kad taškų rinkinys $\{ (x_i, F_i) \in R^2 | \ i= 0, 1,...,N\}$ sudaro duomenų aibę; čia $x_0 < x_1 <... < x_N$. Tolydžioji funkcija $f:[x_0, x_N] \rightarrow R$ tokia, kad $f(x_i) = F_i$, $i = 0, 1, ...,N$, vadinama interpoliacine funkcija, atitinkančia duomenų aibę. Taškai (x_i, F_i), $i = 0, 1,...,N$, vadinami interpoliavimo taškais.

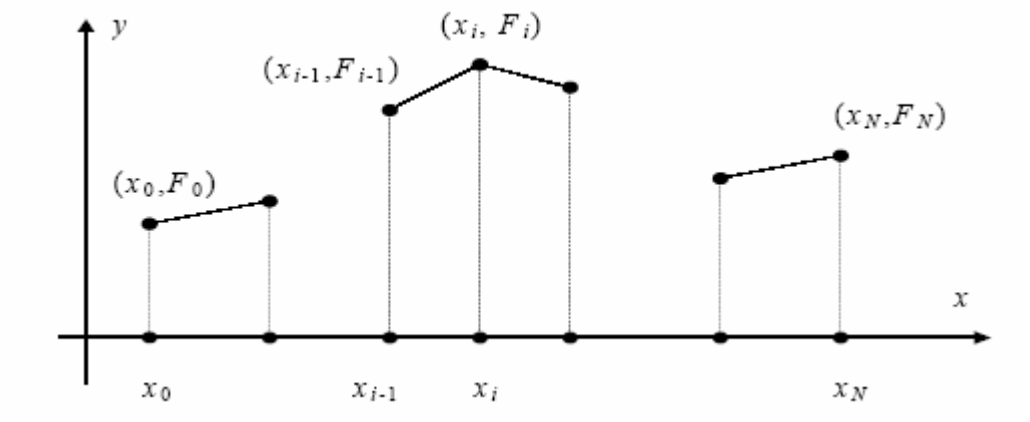
Pasirodo, jog erdvėje R^2 galima sukonstruoti IFS, kurios atraktorius (aibė A) sutaptų su tolydžiosios duomenis interpoliuojančios funkcijos $f:[x_0, x_N] \rightarrow R$ grafiku. Tam pakanka imti pražulniąsias (nėra posūkio y—ašies atžvilgiu) afiniąsias transformacijas.

Kita vertus, afinioji transformacija turi atvaizduoti duomenų aibę (nuo x_0 iki x_N) į „juostą“ (nuo x_{i-1} iki x_i), t.y. turi būti tenkinamos sąlygos:

$$\begin{cases} a_i x_0 + e_i = x_{i-1} \\ a_i x_N + e_i = x_i \\ c_i x_0 + d_i F_0 = f_i = F_{i-1} \\ c_i x_N + d_i F_N + f_i = F_i \end{cases}$$

Nesunku pastebėti, jog šioje tiesinių algebrinių lygčių sistemoje vienas kintamasis (tarsime, kad tai vertikalųjį mastelį keičiantis parametras d_i) yra laisvai pasirenkamas.

Jeigu parinktume d_i=0, su visais $i = 1, 2, \ldots, N$, tai tokios IFS $\{R^2 ;\omega_1, \omega_2,..., \omega_N \}$, atraktorius A sutaptų su laužte (dalimis tiesine interpoliacine funkcija, kurios grafikas „eina“ per interpoliavimo taškus (1.9 pav.))



1.9 pav. Dalimis tiesinės interpoliacinės funkcijos, einančios per interpoliavimo taškus,grafikas; šis grafikas sutampa su IFS $\{R^2 ;\omega_1, \omega_2,..., \omega_N \}$ atraktoriumi

Manipuliuojant parametrų d_i ($i = 1,2,...,N$) reikšmėmis, dydžiui $\sum_{i=1}^{N} | d_i |$ galima priskirti bet kurią reikšmę iš intervalo $[N , 1]$, tuo pačiu fraktalinės interpoliacinės funkcijos (IFS atraktoriaus A) fraktalinę dimensiją prilyginti bet kuriai reikšmei iš intervalo [1,2]. Tai

labai svarbus momentas, kadangi atsiranda galimybė „priderinti“ fraktalines interpoliacinių funkcijų dimensijas prie duomenų aibės fraktalinės dimensijos.

Įdomu ir tai, kad fraktalinė dimensija nepriklauso nuo reikšmių $\{ F_i \mid i = 1,2,...,N\}$, išskyrus tą faktą, jog interpoliavimo taškai neturi priklausyti vienai tiesei. Taigi, galima kalbėti apie fraktalinių interpoliacinių funkcijų, turinčių tą pačią dimensiją D, rinkinį. Tam pakanka išpildyti sąlygą:

$$\sum_{i=1}^{N} | d_i |$$

Nors fraktalinės interpoliacinės funkcijos (FIF) buvo formalizuotos daugiau nei prieš dešimtmetį, tačiau tik pastaraisiais metais imta plačiau jas taikyti kompiuterinėje grafikoje. FIF taikomos kalbos signalų interpoliavimui, modeliuojant kalno vaizdą, seisminius duomenis, elektrokardiogramas. Modeliuojant įvairius paviršius taikomos dvi sudėtinės FIF dalys: FIS(fraktalinės interpoliacinės sistemos) ir FIV (fraktaliniai interpoliaciniai paviršiai).

Be įprastinės FIF dar taikomos ir RFIF (Recurrent Fractal Interpolation Functions) ir PFIF (Partitioned Fractal Interpolation Functions). Rekurentinės FIF — FIF apibendrinimas, sudarantis lankstesnes sąlygas „grubių“ kreivių formavimui. RFIF leidžia naudoti atitinkamos srities dalių sąjungą, o PFIF supaprastina šį apribojimą ir leidžia naudoti laisvai pasirenkamą srities dalį.

Darbe bandoma analizuoti ir spręsti interpoliavimo taškų parinkimo (bendru atveju) problemą. Taipogi siūlomas naujas požiūris į pražulniųjų afiniųjų transformacijų, sudarančių IFS parinkimą.

1.4. Kai kurie fraktalų taikymo pavyzdžiai

Nors fraktalų teorija taikoma tik keletą dešimtmečių, tačiau ja paremti tyrimai (eksperimentai) leido mokslininkams paaiškinti galaktikų evoliuciją, gardelės vystymąsi, kalnodaros ir debesų susidarymo procesus, biržos kainų kaitos procesus ir netgi visuomenės bei šeimos vystymąsi, t.y. vis daugėja sričių, kuriose yra taikomi fraktalinės geometrijos teiginiai ir metodai.

Fraktalai pirmiausiai buvo pritaikyti kompiuterinėje grafikoje (vaizdų suspaudimas, vaizdų modeliavimas ir pan.), kompiuterinėse technologijose (duomenų suspaudimas), fizikoje, medicinoje bei ekonomikoje.

1991 m. M. Barnsley pateikė fraktalinio vaizdų suspaudimo algoritmą, kurio esmė ta, kad IFS parametrų pagalba vaizdas saugomas daug kompaktiškiau. Be to, galima išskirti keletą fraktalinio vaizdų suspaudimo privalumų: suspaustas vaizdas užima nedaug vietos ir jo

atkūrimas užtrunka trumpiau, nei taikant kitus metodus; aukšta atkurto vaizdo kokybė; video suspaudimo galimybė. Savo ruožtu, fraktalinio vaizdų suspaudimo algoritmas leidžia (kaip ir JPEG atveju) užduoti suspaudimo laipsnį (suspaudimas su praradimu). Algoritmo esmė – paieška didelių vaizdo fragmentų, kurie būtų panašūs į mažesnius fragmentus. Faile saugomos tik rastų atitikmenų poros.

Fraktalų panašumas į kalnus, gėles, medžius taikomas kai kuriuose grafiniuose redaktoriuose (fraktaliniai debesys— 3D Studio MAX, fraktaliniai kalnai— World Builder ir kt.).

Be to, eksperimentais yra įrodyta, kad, norint kompiuteryje pavaizduoti trimatę gamtą, būtina kurti trimates fraktalines figūras, nes fraktalai pateikia didžiausią realių detalių kiekį, esant mažiausiam kompiuterio atmintyje saugomų duomenų kiekiui. Vietoj atmintyje saugomų visų kalno detalių galima aprašyti objektą fraktaline lygtimi arba saugoti tik atraktorių, nusakantį visą objektą (analogiškai, trimačiams upės, jūros pakrančių vingių vaizdams gauti ir kt.).

Fraktalų panašumo savybė itin naudinga formuojant naujas tekstūras. Juk kiek bedidintumėm, fraktalinis vaizdas nėra „stambinamas“, praktiškai nuolat atstatomas pradinis vaizdas. Be to, fraktalo „neribotumas“ tiek į plotį, tiek į gylį, leidžia, labai sumažinus ar padidinus fraktalą, gauti praktiškai naujus paveikslėlius.

Fraktalinės geometrijos taikymas (IFS, ar FIF) — tai naujas požiūris į procesų analizę bei prognozes. JAV žemdirbystės departamente (US Department of Agriculture) tyrinėjami fraktalų taikymai fotografijoms (darytoms iš oro), siekiant išanalizuoti augalininkystės rūšių plėtros galimybes. M. Barnsley vienas pirmųjų pasiūlė fraktalinę geometriją taikyti iš palydovų gautų vaizdų analizei, upių pakrančių ir jų baseinų struktūros analizei. Pastaruoju metu fraktalai vis plačiau taikomi modeliuojant socialinių arba civilizacinių procesų modelius.

XX a. 80—ųjų pradžioje analitikas Džonas Merfis sukūrė techninės kainų analizės teoriją, kurios tikslas – kainų raidos prognozavimas. Nors buvo manoma, kad kainų kaita — tai chaotiškas darinys, matematikas B. Mandelbrotas sukūrė universalų techninės kainų analizės metodą, kurio pagalba galima prognozuoti įvairių prekių ir prekybos kainas, kadangi ir kaitos taisyklės, ir kainų grafiko struktūra nekinta.

Techninė analizė taip pat pagrįsta fraktalų geometrijos teiginiais. Apskritai, egzistuoja trys techninės analizės metodų grupės: grafiniai metodai, filtravimo metodai ir ciklų teorija. Tipinis techninės analizės fraktalo pavyzdys yra Elioto bangos.

Fraktalai, pritaikyti labai sudėtingoje antenų konstravimo srityje, pagerino antenų matricų kokybę. Be to, antenoms suteikus fraktalo formą, pagerėjo antenų parametrai.

Pavyzdžiui, antenoms suteikus Kocho kreivės ar Serpinskio trikampio formą, antenų plotas sumažėjo 6 kartus; buvo sukurta elektrinė talpa bei indukcija. To pasekoje, derinant anteną ar didinant jos priimamų dažnių juostos plotį, nebereikia išorinių komponentų. Fraktalų antenos sėkmingai naudojamos Motorola mobiliuosiuose telefonuose.

Apskritai, pastaraisiais dešimtmečiais fraktalai vis dažniau taikomi chemijos, biologijos moksluose, kalbotyros moksluose bei muzikoje. Įvairių fraktalinių dydžių analizė ir fraktalinių objektų modeliavimas, seniai nebe fizikų ir programuotojų prerogatyva, randa pačias netikėčiausias taikymo sritis. Tiriama tekstų ir hipertekstų, muzikinių kūrinių, biržos kotiruočių, smegenų procesų dinamikos, komunikavimo socialinėse sistemose darinių fraktalinės struktūros, globalių informacinių tinklų dinaminiai procesai.

Fraktalinės geometrijos kalba taip pat būtina tiriant spinduliavimo elementariose terpėse sklidimą ir sugėrimą, ryškios turbulencijos charakterizavimui, kietųjų kūnų paviršių savybių modeliavimui, žaibo ir elektros smūgio aprašymui, objektų senėjimo procesų analizėje, difuzijos ir agregacijos įtakos kūnams tyrimui, kvantinėje mechanikoje— banginių funkcijų geometrinių struktūrų Andersono metalo— dielektriko perėjimo taške aprašymui. Astrofizikoje – aprašant Visatos galaktikų klasterizacijos procesus; kartografijoje – tiriant pakrančių linijų formas, upių tinklą; biologijoje – kraujotakos sistemos struktūros analizei arba sudėtingų paviršių ląstelinių membranų apžvalgai. Be to, tiriamos fraktalinės tekstų ir hipertekstų struktūros, muzikos kūriniai, biržos kotiruotės, smegenų procesų požymių dinamikoje, auklėjimo atraktoriai, socialinių sistemų komunikacijos struktūros, globalių informacinių tinklų dinaminiai procesai.

Apskritai, fraktalams atsirado vieta įvairiuose moksluose. Pavyzdžiui, susiformavo fraktalinių trūkimų teorija, fraktalinių paviršių trinties teorija, medžio—polimerų pavidalo kompozitų fraktalinė mechanika, fraktalinė optika bei kitos sritys. Fraktalų pagalba mokslininkai aiškina galaktikų evoliuciją, kalnodaros procesus, debesų susidarymą, biržos kainų pokyčius, visuomenės ir šeimos vystymąsi.

1.5. Išvados.

- ✓ Kadangi tiriamojo darbo objektas yra teoriniai ir praktiniai fraktalinių interpoliacinių funkcijų sudarymo aspektai, tai pagrindinį dėmesį toliau skirsime algebriniams fraktalams, gaunamiems taikant keruotąsias (afiniųjų) funkcijų sistemas.

- ✓ Naudosime atsitiktinių iteracijų algoritmą, gautai iteruotųjų funkcijų sistemų atraktorių (fraktalinius vaizdus).

- ✓ Kadangi elementariosios funkcijos (tokios kaip sinusas, kosinusas, daugianariai ir panašiai.) sudaro tradicinio, visuotinai priimto eksperimentinių duomenų analizės metodo pagrindą, tai naudosime fraktalines interpoliacines funkcijas.

2. PROJEKTINĖ DALIS

2.1. Reikalavimų sistemai specifikacija

2.1.1.Operacinių sistemų apžvalga

Vartotojiška ir mokomoji fraktalinių vaizdų kūrimo sistema dirba Windows 9x, Me, NT ir XP operacinių sistemų platformose. Windows operacinė sistema buvo pasirinkta atsižvelgiant į užsakovo aparatūrą palaikančią sistemą ir jos paplitimą tarp namų vartotojų. Mes rekomenduojame Windows XP operacinę sistemą, kaip labiausiai paplitusią tarp namų vartotojų.

Windows XP operacinės sistemos charakteristikos:

- Palyginti stabili.
- Tinklo palaikymas.
- PnP ir USB palaikymas.
- Nesudėtingas įdiegimas.
- Draugiška vartotojo sąsaja.
- Labai paplitusi tarp namų vartotojų.
- Suderinamumas.

2.1.2. Programavimo kalbų apžvalga

Kuriant vartotojo sąsajos kompiuterinę programą, apžvelgėme keletą programavimo kalbų (Delphi ir C++). Renkantis programavimo kalbą buvo vertinami šie pagrindiniai kriterijai:

- Kodo paprastumas.
- Vizualinio programavimo galimybės.
- Programavimo terpės naujumas.
- Darbo stabilumas Windows aplinkoje.

Kaip daugiausia sąlygų tenkinanti programavimo kalba – Delphi buvo pasirinkta šiam projektui įgyvendinti.

Trumpa programavimo kalbų C++ ir Delphi apžvalga:

C++

Privalumai:

- Programavimas beveik visose operacinėse sistemose.
- Yra patogių meistrų kurti būsimus programos šablonus.
- Lengva parašyti mažai vietos užimamas programas.

Trūkumai:

- Apribojimai kuriant vizualinį vartotojo interfeisą.
- Daug programinio teksto.
- Programa sudaro daugiau nei 1 failas.
- Nebuvo atnaujinta nuo 1998 metų.

Delphi

Privalumai:

- Greitas ir patogus vizualinis programos kūrimas.
- Mažai programinio teksto.
- Programa sudaro 1 failas ir nereikalaujama programos diegimo.
- 2002 metų atnaujinimas.
- Stabilumas.
- Kodo paprastumas.

Trūkumai

- Programavimas tik Windows ir Linux operacinėms sistemoms.
- Didelis paleidžiamasis failas.

Kiekviena programavimo kalba turi savo privalumų ir trūkumų. Ir, jei nors viena kalba atsiliktų, - ji būtų pasmerkta. Problema ta, kad šios kalbos skirtos skirtingoms užduotims spręsti. “Delphi” aplinka tinkamesnė *offise (word, exsel)* tipo programoms kurti, kitoms programoms, naudojančioms vizualią vartotojo sąsają rašyti (pagrindinis kriterijus nulėmęs Delphi pasirinkimą). “C++” kalba patogesnė ten, kur vizualios vartotojo sąsajos nėra. Jeigu programuoji žaidimus, programas palaikančias aparatūra arba servisus, tai geriausia pasirinkti

"C++", bet ir su "Delphi galima kurti programas, palaikančias aparatūrą ar žaidimus, bet tiesiog čia jis nėra toks patogus. Tas pats ir su "C++", kuris tai pat tinka vizualioms programoms rašyti, tiesiog čia daugiau darbo teks įdėti į vizualinį programos apipavidalinimą.

2.2. Architektūros ir algoritmai

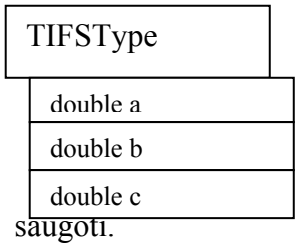
2.2.1. Projekte naudojamos duomenų struktūros

Projekte be *TfrmMain* klasių ir Delphi suteikiamų duomenų struktūrų naudojamos savos struktūros, jos aprašomos modulyje *MyTypes* :

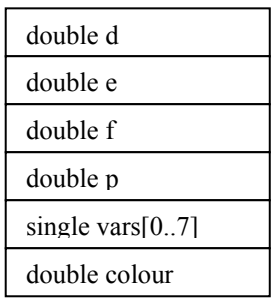
- *TIFSType*
- *TDisplayProp*
- *TRandomSettings*
- *TTriangle*
- *TMapPalette*
- *TBounds*

TIFSType

Struktūra naudojama IFS bylų parametrams saugoti, kurie bus naudojami skaičiavimams.



a – pagalbinis kintamasis absisės pradžios reikšmei saugoti.
b – pagalbinis kintamasis absisės pabaigos reikšmei saugoti.
c – pagalbinis kintamasis ordinatės pradžios reikšmei saugoti.
d – pagalbinis kintamasis ordinatės pabaigos reikšmei saugoti.

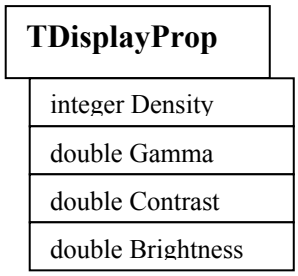


e – pagalbinis kintamasis x reikšmė plokštumoje.
f – pagalbinis kintamasis y reikšmė plokštumoje.
p – pagalbinis kintamasis interacijų skaičius.
vars – pokyčių erdvėje (variacijų masyvas).
colour – spalva.

Kadangi ši struktūra naudojama ne viename modulyje, todėl parametrams naudojamas double tipas.

TDisplayProp

Struktūra skirta vaizdo parametrų nustatymams .



Density – vaizdo parametras tankumo nustatymui.

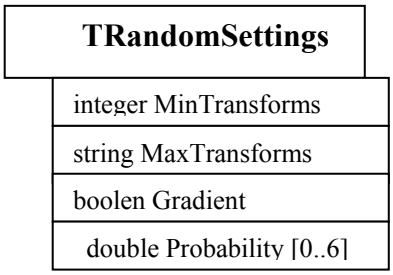
Gamma – vaizdo parametras gamos nustatymui.

Contrast – vaizdo parametras kontrasto nustatymui.

Brightness – vaizdo parametras šviesumo nustatymui.

TRandomSettings

Struktūra skirta atsitiktinių parametrų ir koeficientų skaičiavimui.



MinTransforms – minimalus leistinas transformacijų skaičius.

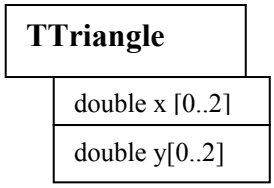
MaxTransforms - maksimalus leistinas transformacijų skaičius.

Gradient – atsitiktinio gradiento koeficientas.

Probability – atsitiktinių galimybių masyvas.

TTriangle

Struktūra skirta trikampio viršūnių taškų nustatymui.

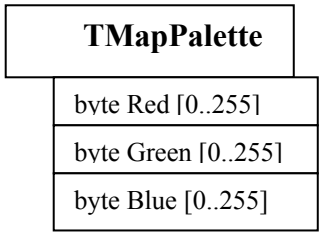


x – trikampio viršūnės x koordinatė

y – trikampio viršūnės y koordinatė

TMapPalette

Struktūra skirta išskaidytos RGB spalvos nustatymui.



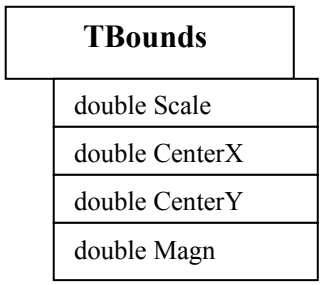
Red – raudonos spalvos reikšmė.

Green – žalios spalvos reikšmė.

Blue – mėlynos spalvos reikšmė.

TBounds

Struktūra skirta vaizdo ribų ir centro nustatymo parametrams saugoti.



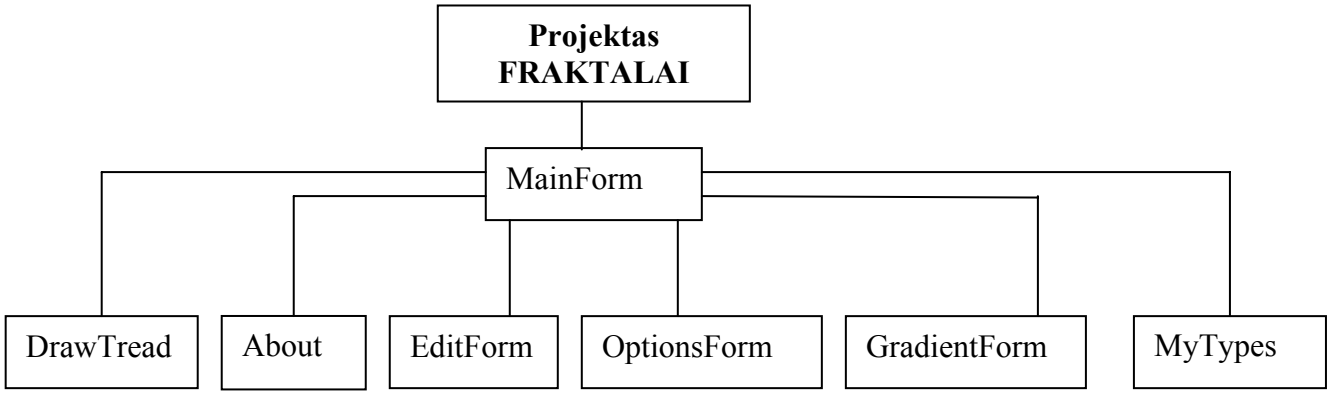
Scale – vaizdavimo skalė.

CenterX – centro x koordinatė.

CenterY – centro y koordinatė.

Magn – vaizdo nustatytam rezultatui saugoti.

2.2.2. Projekto pagrindinių modulių hierarchinė schema



3.1 pav. Projekto pagrindinių modulių hierarchinė schema

2.2.3. Pagrindinių modulių paprogramių aprašai

Modulis MainForm - pagrindinio lango modulis, naudoja klasę *TfrmMain*, paveldima klasė *TForm.* Šioje klasėje ne tik vykdomos langų sukūrimo bei valdymo operacijos, bet ir aptarnaujamas projekto langas. Vykdomos pagrindinės paprogramės.

- procedure *Animate* – pasirenkamas atsitiktinių fraktalų trikampių centras, animacija vykdoma 180 laipsnių apsisukimais, kadangi trikampiai ne lygiakraščiai, sustabdžius animaciją gaunasi naujas fraktalas.
- procedure *ComputeIFS* – po naujos įvykdytos komandos arba korekcijos perskaičiuojamas fraktalas.
- **procedure** *ComputeWeights* - pakeitus trikampių padėtį perskaičiuojamos transformacijos dydis.

- **procedure** *DrawBlankScreen* – jei atliekamas skaičiavimas, pagal lango dydį performuojamas ekrano dydis bei formuojamas naujas vaizdas.
- procedure *IFS_compute* – parinkus fraktalo parametrus, perskaičiuojama trikampių pozicija.
- **procedure** *ListIFS* – iš fraktalinių failų su plėtinių *.fla arba *.ifs skaitomi bei ruošiami duomenys tolesniam darbui.
- **procedure** *ListFlames* – iš failo Frak.fla skaitomi bei ruošiami duomenys.
- procedure *RandomBatch* – atsitiktinai parenka kitą fraktalų, pvz. grupę, bei perrašo juos į failą Frak.fla.
- **procedure** *RandomIFS* – atsitiktinai pertvarko trikampių padėtį bei jų transformacijų skaičių, taip gaunasi kitas atsitiktinis fraktalas*;*
- **procedure** *RandomPalette* – atsitiktinai fraktalui parenkama spalvų gama bei pagal ją spalvina.
- **procedure** *RandomVariation* – atsitiktinai parenkamas fraktalo išsidėstymas erdvėje.
- procedure *RandomWeights* – atsitiktinai parenkamas fraktalo dydis.
- procedure *SaveAs* – pasirinktu formatu *.fla arba *.ifs išsaugomas fraktalas norimu vardu.
- **procedure** *SetVariation* – nustatomas pagal nutylėjimą pasirinktas išdėstymas erdvėje.
- **procedure** *ShowInfo* – būsenos juostoje rodomas pasirinktos variacijos pavadinimas.
- procedure *TrianglesFromIFS* – iš IFS kodo nustatoma trikampių pozicija bei transformacijų kiekis.
- procedure *UpdateUndo* – atšaukiamas paskutinis padarytas pakeitimas.

Modulis Editform – IFS readaktoriaus modulis, naudoja klasę *TfrmEdit*, paveldima klasė *TForm*. Trikampių pagalba transformuojami fraktalai. Piešiami nauji fraktalo elementai.

Nustatomi žemėlapio parametrai. Pasirenkama variacija.

- **procedure** *AutoZoom* – automatinis vaizdo priartinimas.

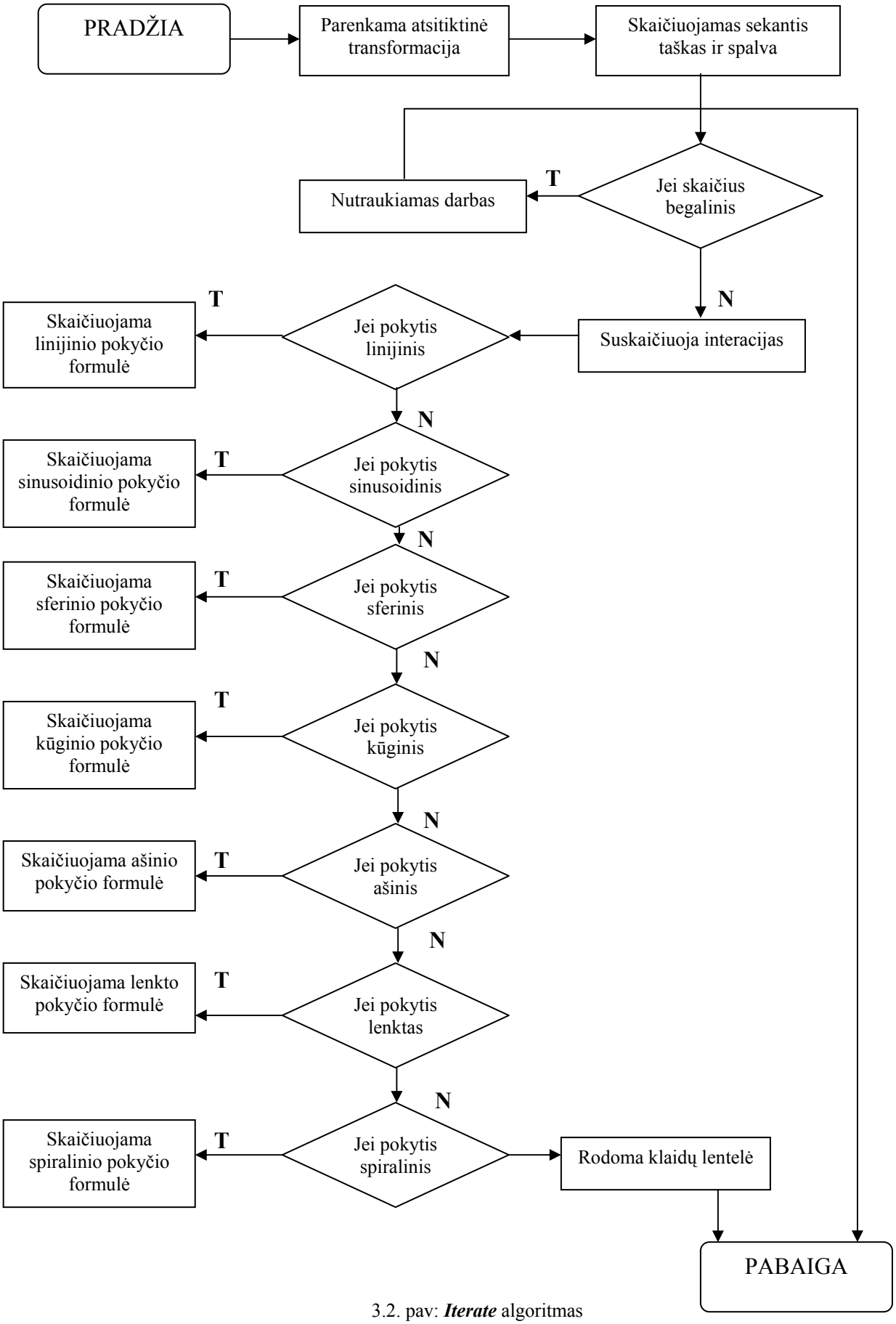
- **procedure** *DeleteTriangle* – pašalina trikampį, negali likti mažiau kaip du trikampiai.
- **procedure** *DrawGraph* – pagal trikampių padarytus pakeitimus piešiamas fraktalo vaizdas, padarius transformaciją matomas rezultatinis fraktalas.
- **procedure** *Scale* – būsenos juostoje, skaičiuojamas pelės taško pozicija pagal dvimatę koordinačių sistemą.
- **procedure** *ShowSelectedInfo* – rodomos darbinio, pažymėto trikampio kiekvienos viršūnės koordinates pagal trikampio viršūnes.
- **procedure** *UpdateIFS* – atnaujinamas vaizdas pagrindiniame lange, matomas pakeistas fraktalas.
- **procedure** *VNormalize* – sutvarkomas ryšis tarp transformuojamų trikampių bei pasirenkant išdėstymą erdvėje.

Modulis GradientForm – spalvų gamos lango modulis, naudoja klasę *TfrmGradien,* paveldima klasė *TForm.*

- procedure *DrawPalette* – pasirinkus atitinkamą spalvų gamą, piešiamas jos pokytis.

Modulis DrawThead – fraktalo piešimo algoritmo modulis, naudoja klasę *TThread,* paveldima klasė *TDrawThrd.*

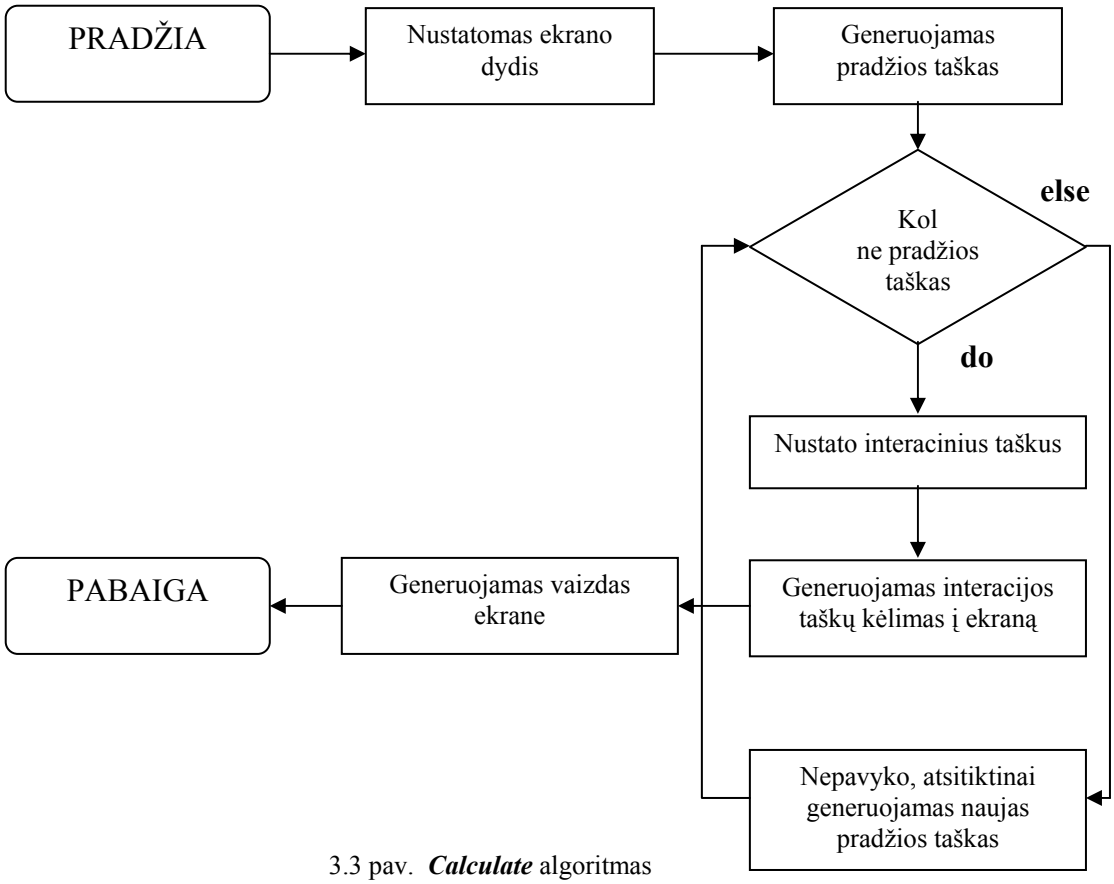
- **function** *Iterate.* Vykdomas procesas, kuris skaičiuoja atsitiktinai pasikartojančias interacijas. Metodo algoritmas pateiktas 3.2 pav.



3.2. pav: ***Iterate*** algoritmas

- **function** *Calculate.* Skaičiuojamo fraktalo ir jo išvedamo į ekraną algoritmas.

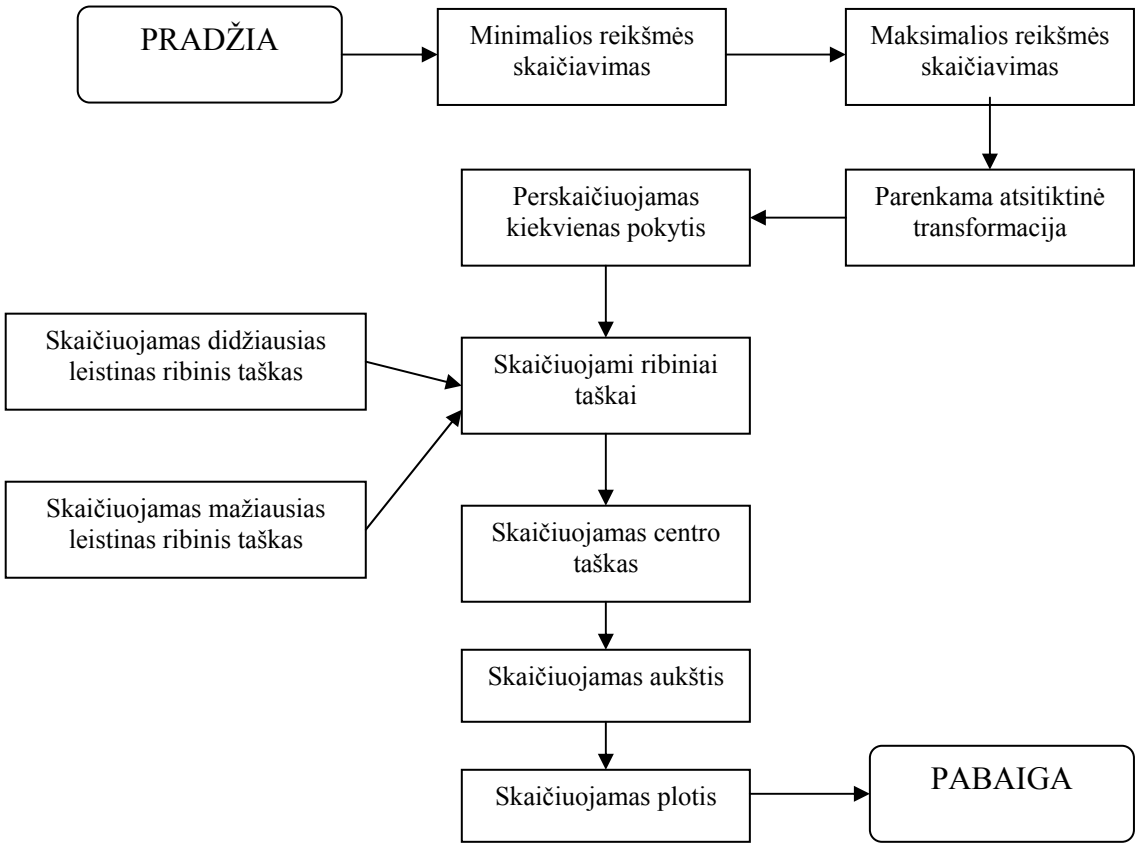
Metodo algoritmas pateiktas 3.3 pav.



3.3 pav. ***Calculate*** algoritmas

- **function** *GetBounds.* Vaizdo centro ir vaizdo ribų nustatymo algoritmas.

Metodo algoritmas pateiktas 3.4. pav.



3.4 pav. ***GetBounds*** algoritmas

- **procedure** *Execute***.** Įvykdo aukščiau minėtas funkcijas.

Modulis OptionsForm – savybių lango modulis, naudoja klasę *TfrmOptions,* paveldima klasė *TForm.* Šiame lange nustatomos:

- ✓ bendros savybės (lango skiriamosios gebos nustatymas, spalvų gamos parametrų nustatymas, pasirenkamas UPR tipas).
- ✓ vaizdo savybės (pagrindinis: tankumas, ryškumas, gama, kontrastas; peržiūros tankūmas gama; animacijos tankumas).
- ✓ atsitiktinės savybės (transformacijų skaičius: maksimalus, minimalus).
- ✓ IFS savybės.
- ✓ fraktalo savybės.

- **procedure** *btnCancelClick.* Uždaro savybių langą.

- **procedure** *FormShow*. Aktyvuoja savybių langą ir užkrauna duomenis, esančius pagal nutylėjimą.
- **procedure** *btnOKClick*. Užsaugo ir patvirtina nustatytus parametrus.
- **procedure** *btnDefGradientClick*. Skirta spalvų gamos parinkimui pagal nutylėjimą iš kietojo disko.
- **procedure** *FormClose*. Uždaro savybių langą.

2.3. Testavimo medžiaga

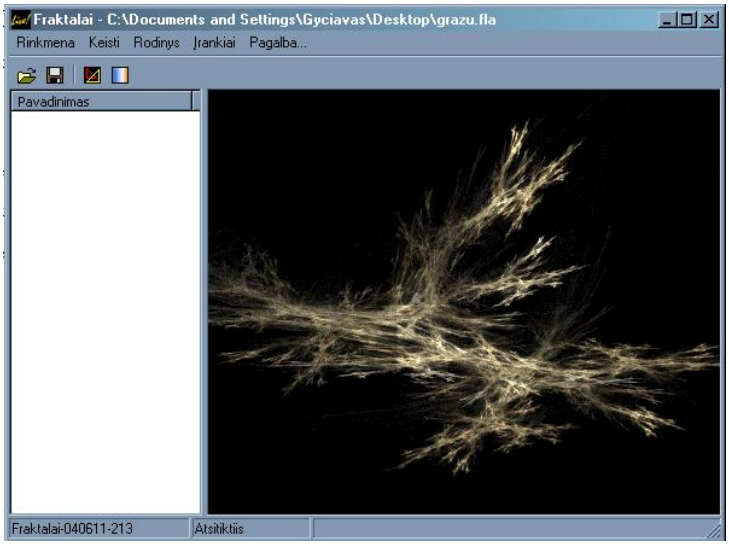
Sistemos testavimą atlieka kūrėjai ir užsakovas. Kadangi sistema buvo kuriama naudojant prototipus ir integruojant modulius, tai kiekviename etape kūrėjų atliekamas integracinis testavimas, patikrinantis modulius pavieniui ir modulių sąveikos korektiškumą. Užsakovo reikalavimų atitikimui patikrinti sudaryti ir žemiau pateikti priėmimo testo planai, kurių visi atlikti punktai turi duoti teigiamą rezultatą, siekiant patvirtinti sistemos teisingumą ir funkcionalumą.

Tikrinami reikalavimai:

- Fraktalo generavimas.
- Fraktalinės interpoliacines fukcijos veikimas.
- Spalvų gamos pritaikymas.
- Fraktalo redagavimas.
- Fraktalo eksportavimas į kitus formatus.

Rezultatai:

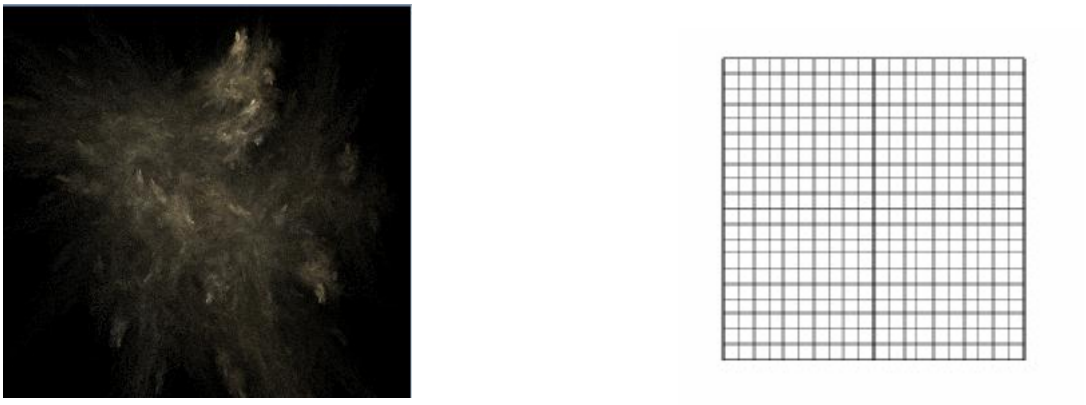
Paleidus programą iš karto gavome sugeneruotą fraktalą:



3.5 pav. Sugeneruotas fraktalas

Fraktalinės interpoliacinės fukcijos:

Interpoliuojant **tiesės** fukcija $v_0(x, y) = (x, y)$, gaunamas rezultatas 3.6 pav.

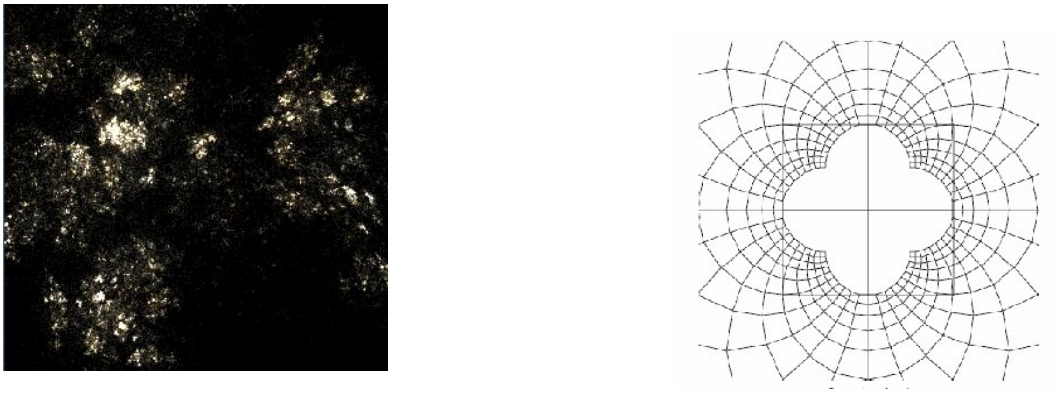


3.6 pav. Linijine interpoliacija

Interpoliuojant **sinuso** funkcija $v_1(x,y)=(\sin x, \sin y)$, gaunamas rezultatas 3.7 pav.

3.7 pav. Sinusoidinė interpoliacija

Interpoliuojant **sferos** funkcija $v_2(x,y)=(x/ r^2, y/ r^2)$, gaunamas rezultatas 3.8 pav.

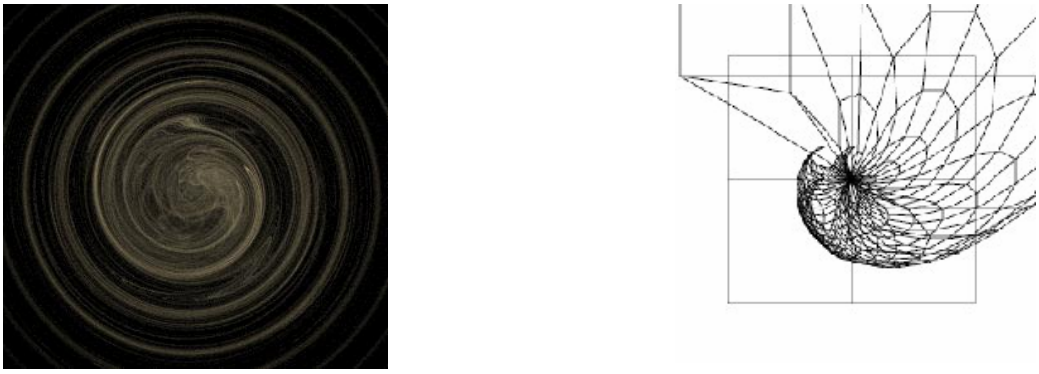


3.8 pav. Sferinė interpoliacija

Interpoliuojant **sukinio** funkcija $v_3(x,y)=(r\cos(\theta +r), r\sin(\theta +r))$, gaunamas rezultatas 3.9 pav.

3.9 pav. Sukinė interpoliacija

Interpoliuojant **spiralinis** funkcija $v_4(x,`y) = ((\cos\theta +\sin r)/r, (\sin\theta - \cos r)/r)$, gaunamas rezultatas 3.10 pav.

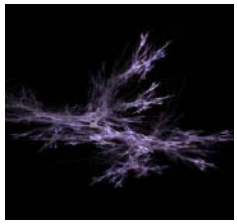


3.10 pav. Spiralinė interpoliacija

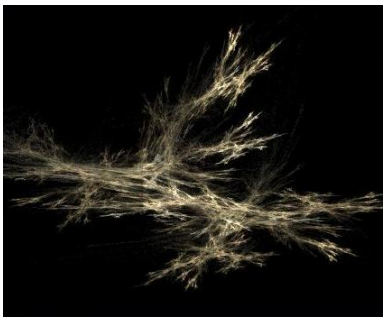
Interpoliuojant **polinė** funkcija $v_5(x,`y) = (\theta/\pi, r - 1)$, gaunamas rezultatas 3.11 pav.

3.11 pav. Polinė interpoliacija

Pritaikę spalvų gamą gavome:

Fraktalo redagavimas, gavome :




Prieš Po

Fraktalą eksportavome į bmp formatą ir rezultatą gavome faile „*Bandymas.bmp“*

2.4. Išvados

- ✓ Išanalizavus fraktalų generavimo bei jų panaudojimo principus, suprojektuota fraktalinio interpoliavimo priemonė.
- ✓ Išanalizavus fraktalinės interpoliacines funkcijas, gauti fraktaliniai tiesės, sinusoidino, sferinio, kūginio, spiralinio bei polinio interpoliavimo vaizdai.
- ✓ Pagrįstas vartojamų programinių priemonių parinkimas.

3. VARTOTOJO DOKUMENTACIJA

3.1. Sistemos funkcinis, instaliavimo, administratoriaus aprašymas

Vartotojiška ir mokomoji fraktalinių vaizdų kūrimo sistema dirba Windows 9x, Me, NT ir XP operacinių sistemų platformose.

Programos pavadinimas: Fraktalai.

Autorius: Gytis Urbonavičius.

Programa užima 1.88 Mb vietos kietajame diske.

Programa sudaro 4 failai: Fraktalai.exe, CMAP.UGR, IFS.UGR, Pagalba.pdf.

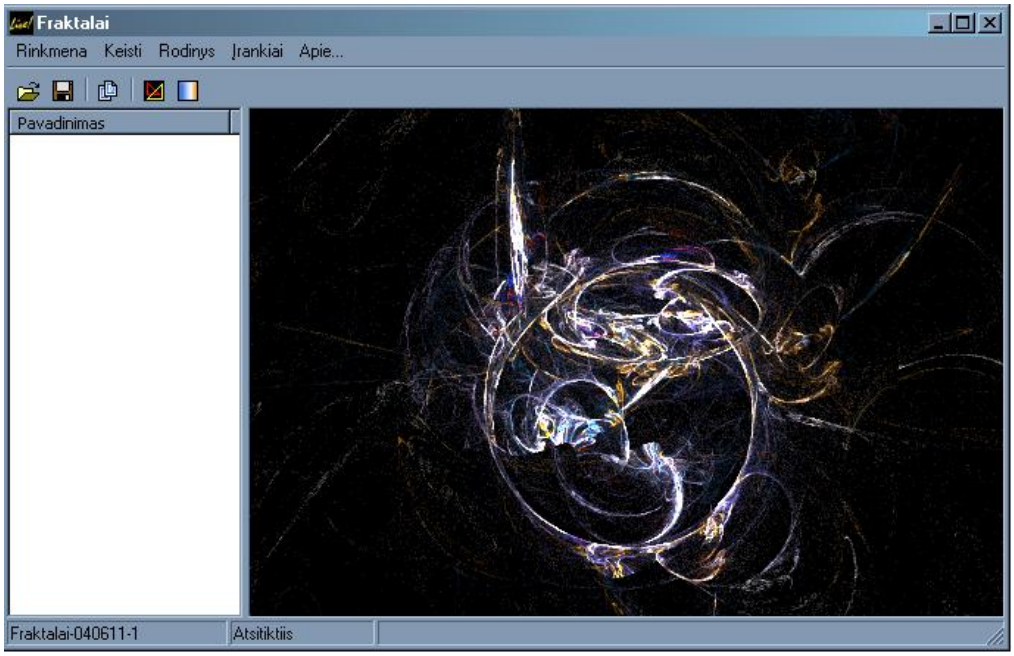
Norint tvarkingai paleisti programa svarbu, kad visi failai būtų tame pačiame kataloge. Programos instaliuoti nereikia, užtenka paleisti „frakatalai.exe“ ir ji jau veikia.

Kadangi sistemoje nėra nei vartotojų grupės, nei administratoriaus teisių, ypatingų priemonių nereikia paleidžiant programą.

3.2. Įžanginis vadovas

3.1.1. Programos paleidimas

Norint paleisti programą reikia: Įsitikinti, kad dirbate Windows 95/98/NT/2000/XP/ operacinėje sistemoje. Surasti programos paleidžiamąjį failą „Fraktalai.exe“ ir jį paleisti. Tai atlikus turi atsirasti pagrindinis programos langas 4.1 pav., jame jau bus matomas atsitiktinai sugeneruotas fraktalas.



4.1 pav. Pagrindinis langas

Kitas būdas – po paleidžiamo failo nurodyti fraktalinio *.IFS arba *.Fla plėtinio failą. Tokiu būdu programa iš karto užsikraus su jūsų pasirinkto fraktalinio failo vaizdu.

3.1.2. Programos valdymas

Programos valdymui realizuotas tiek valdymas pele, tiek mygtukų pagalba, kadangi komandos turi savo klavišų sekas ar atskirus klavišus. Komandai priskirtą klavišą ar klavišų kombinaciją galima sužinoti suradus ieškomos komandos meniu punktą. Šalia kiekvieno meniu punkto antraštės nurodyta jį atitinkanti kombinacija (Pvz.: Crtl+G atveriamas „Spalvų gamos“ pasirinkimo langas).

3.1.3. Išėjimas iš programos

Galimi keli būdai:

- ✓ Paspausti lango kampe esantį mygtuką su ant jo pavaizduotu kryžiumi.
- ✓ Pasirinkti menių punktą „Rinkmena“ → „Uždaryti“.
- ✓ Paspausti klavišų kombinacija Alt+F4.

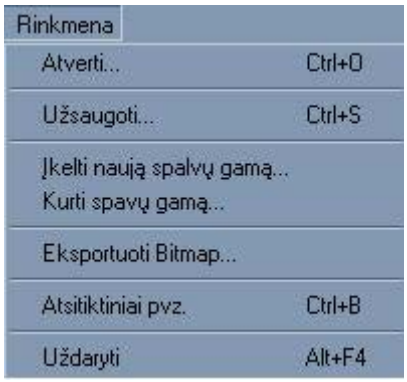
3.3. Sistemos nuorodų vadovas

Programa valdoma meniu arba įrankių juostos pagalba. Šiame skyriuje detaliai aprašomi meniu punktai ir jos atitinkantis įrankių juostuos mygtukai. Bendras meniu ir įrankių juostos vaizdas parodytas 4.2 pav.

4.2 pav. Sistemos nuorodų vadovas

3.3.1. Meniu punktai ir juos atitinkantys įrankių juostos mygtukai

3.3.1.1. Meniu punktas „Rinkmena“.

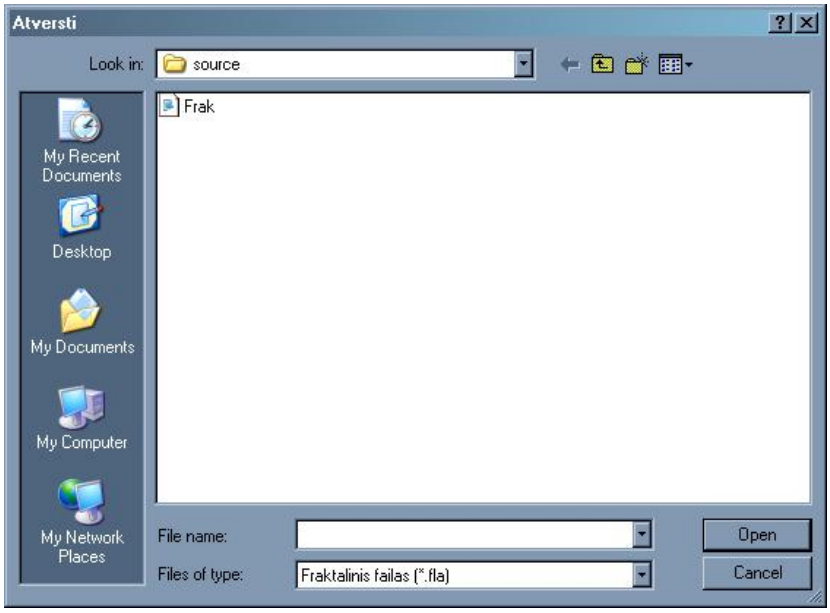


Šis meniu 4.3 pav. skirtas fraktalinių failų atidarymui, užsaugojimui, spalvų gamos įkėlimui, eksportavimui į kitus formatus, fraktalų užkrovimui, animacijos užsaugojimui bei darbo pabaigai.

4.3 pav. 1 Meniu "Rinkmena"

Komanda „*Atverti*“.

 Skirta failo nuskaitymui į atmintį ir jo išvedimui į ekraną. Iškviesti ją galima pasinaudojus meniu arba įrankių juostos pagalba (atitinkamas mygtukas parodytas kairėje). Pasirinkus šia komanda ekrane pasirodys toks dialogo langas:



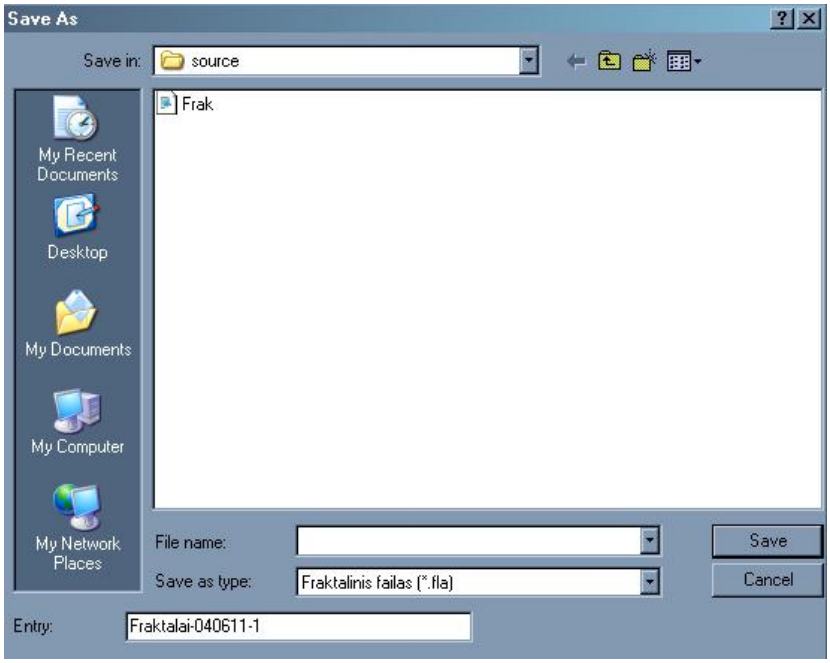
4.4 pav. Langas atversti

Šiame lange (4.4 pav.) galite pasirinkti failą, kurį norite atidaryti. Tam, kad tai padaryti – pažymėkite norimą failą kairio pelės klavišo mygtuką ir paspauskite mygtuką „Open“.

Tuomet, jei failas sėkmingai atidarytas, ekrane pamatysime išvestą fraktalą.

Komanda „*Užsaugoti*“.

 Skirta failo įrašymui į kietą diską. Iškviesti ją galima pasinaudojus meniu arba įrankių juostos pagalba (atitinkamas mygtukas parodytas kairėje). Pasirinkus šią komandą, ekrane pasirodys 4.5 pav. pav. parodytas langas:



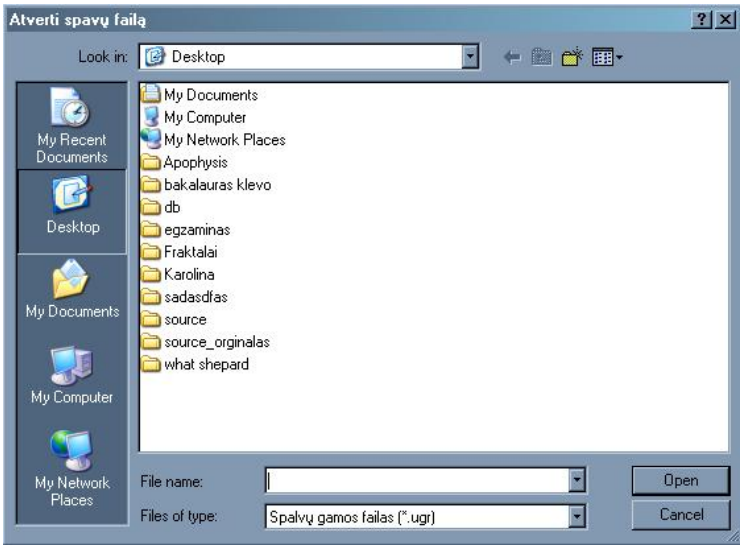
4.5 pav. Langas užsaugoti

Įvedus failo vardą pasirinkite kokiu norite formatu *.fla arba *.ifs užsaugoti ir paspauskite „Save“.

Komanda „*Įkelti naują spalvų gamą*“.

Skirta naujos spalvų gamos įkėlimui į atmintį. Iškviesti ją galima pasinaudojus meniu pagalba.

Pasirinkus šią komandą ekrane pasirodys toks dialogo langas:

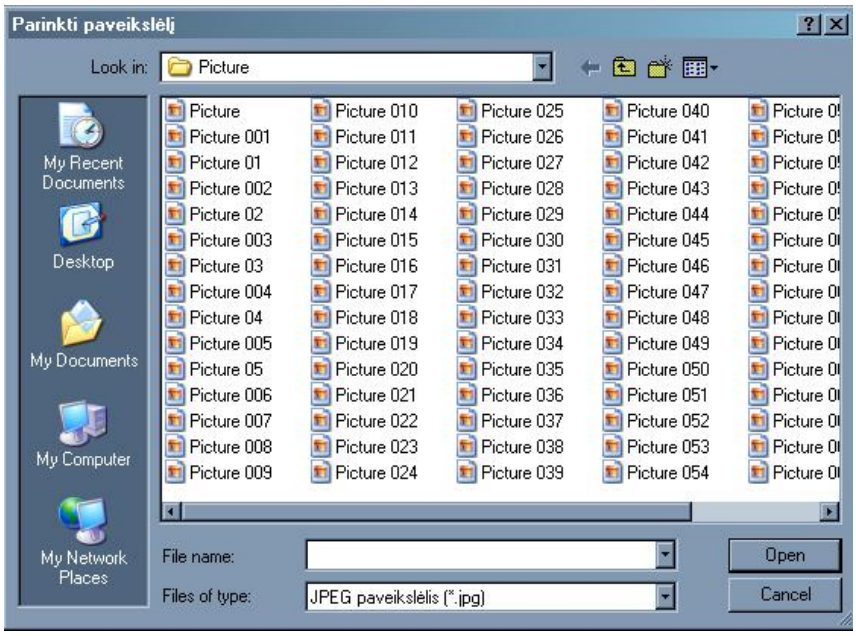


4.6 pav. Atversti spalvų langą

Šiame lange (4.6 pav.) galite pasirinkti spalvų gamos failą su plėtiniu *.ugr , kurį norite atidaryti. Tam, kad tai padaryti – pažymėkite norimą failą kairio pelės klavišo mygtuku ir paspauskite mygtuką „Open“. Tuomet, jei failas sėkmingai atidarytas, pasirinktas fraktalas automatiškai perskaičiuojamas pagal naują spalvų gamą.

Komanda „*Kurti spalvų gamą*“.

Skirta naujai spalvų gamos kūrimui. Iškviesti ją galima pasinaudojus meniu pagalba. Pasirinkus šią komandą ekrane pasirodys toks dialogo langas:



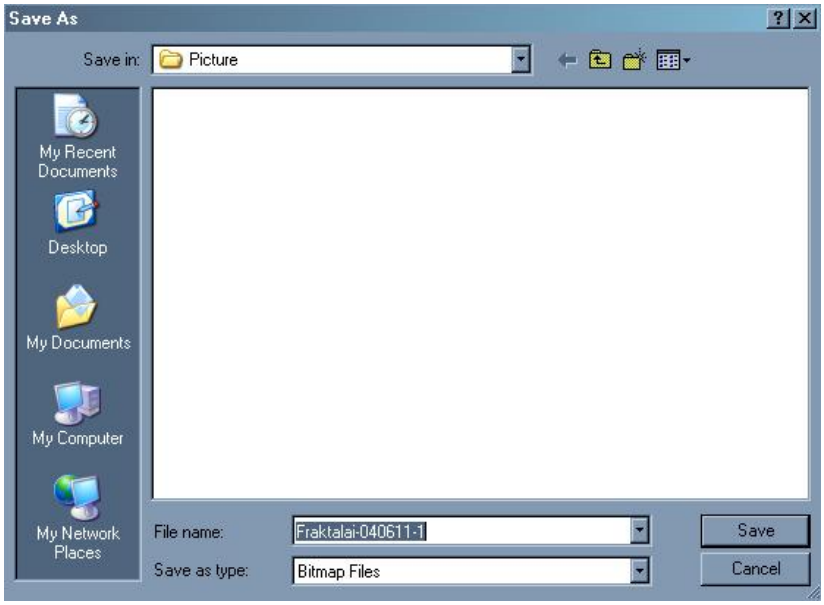
4.7 pav. Parinkti paveikslėlį

Šiame lange (4.7 pav.) galite pasirinkti paveikslėlio failą su plėtiniu *.jpg arba *.bmp , kurį norite atidaryti. Tam, kad tai padaryti – pažymėkite norimą failą kairio pelės klavišo

mygtuku ir paspauskite mygtuką „Open“. Tuomet, jei failas sėkmingai atidarytas, pasirinktas fraktalas automatiškai perskaičiuojamas pagal naujai sukurtą spalvų gamą. Nauja spalvų gama automatiškai bus išsaugota atmintyje.

Komanda „*Eksportuoti Bitmap*“.

Skirta esamo fraktalo eksportavimui į paveikslėlio formatą *.bmp. Iškviesti ją galima pasinaudojus meniu pagalba. Pasirinkus šią komandą ekrane pasirodys toks dialogo langas:

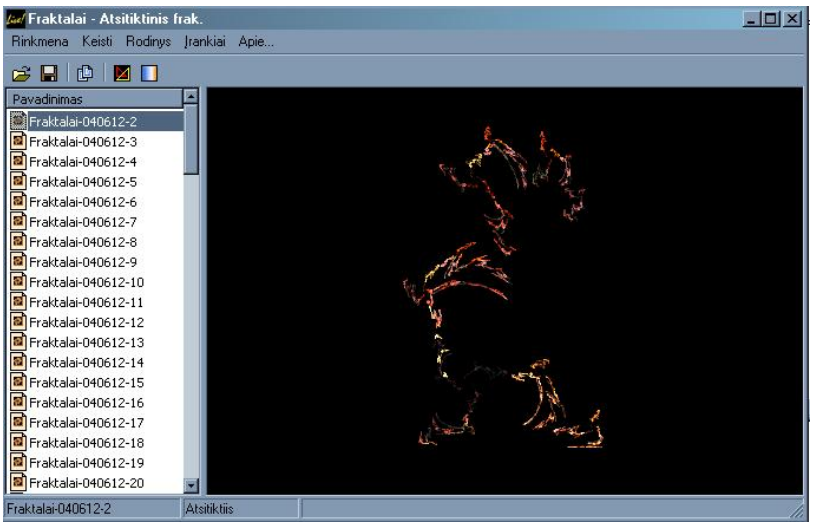


4.8 pav. Eksportavimo langas

Šiame lange (4.8 pav.) įvedus failo vardą paspauskite „Save“ ir vaizdinys bus eksportuotas ir išsaugotas *.bmp formatu. Paveikslėlį galima peržiūrėti bet kokia grafininių vaizdų naršykle, kuri palaiko failus su tokio tipo plėtiniais.

Komanda „*Atsitiktiniai pvz.*“ .

Įvykdžius komandą, į atmintį atsitiktinai sugeneruojama 100 fraktalų pavyzdžių. Pavyzdžiai išsaugomi faile Frak.fla, atlaisvinama atmintis. Pagrindiniame lange matome:



4.9 pav. Atsitiktiniai pvz.

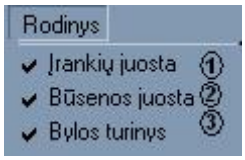
Šiame lange (4.9 pav.) matome atsitiktinai sugeneruotų fraktalų pvz. Paspaudę ant bet kurio pelės pagalba, matysime jo vaizdą. Pakartojus komandą *Atsitiktiniai pvz.*, bus generuojamas kitas 100 fraktalų pavyzdžių. Faile Frak.fla bus užsaugoti paskutinio generavimo parametrai.

3.3.1.2. Meniu punktas „Keisti“

Šis meniu 4.10 pav. skirtas (1) atšaukti paskutinę įvykdytą komandą ar pakeitimą, (2) gražinti atšauktą komandą į pradinę būseną.

4.10 pav. Meniu "Keisti"

3.3.1.3. Meniu punktas „ Rodinys “



Šis meniu 4.11 pav. turi darbo aplinkos nustatymo parametrų keitimo punktus.

4.11 pav. Meniu "Rodinys"

Komanda „*Įrankių juosta*“. (4.12 pav. (1))

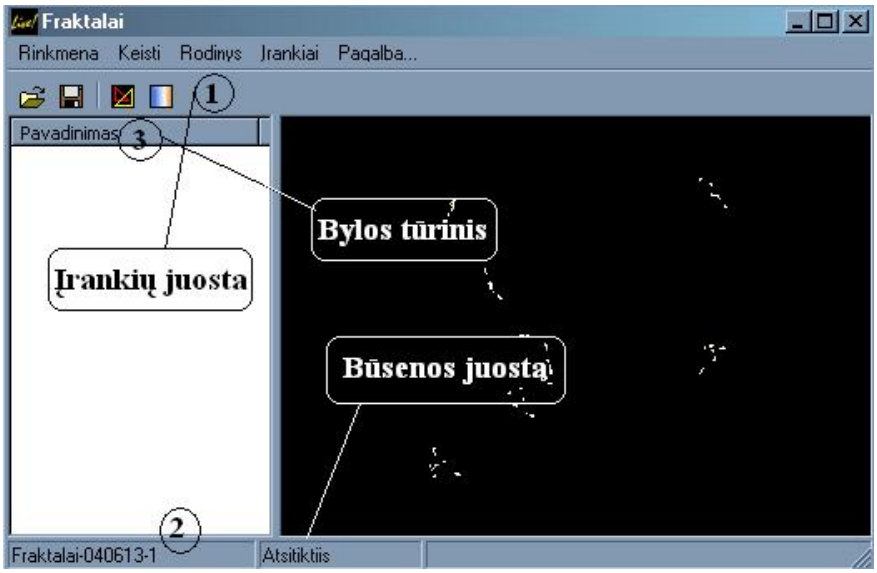
Komanda skirta įrankių juostos rodymo arba paslėpimo režimų aktyvavimui. Šalia šios komandos yra žymeklis („varnelė“), kuris rodo dabartinę informacinės juostos būseną. T. y., jam esant įrankių juosta yra matoma, kitaip – ne.

Komanda *„Būsenos juosta“*. (4.12 pav. (2))

Komanda skirta būsenos juostos rodymo arba paslėpimo režimų aktyvavimui. Šalia šios komandos yra žymeklis („varnelė“), kuris rodo dabartinę būsenos juostos būseną. T. y., jam esant būsenos juosta yra matoma, kitaip – ne.

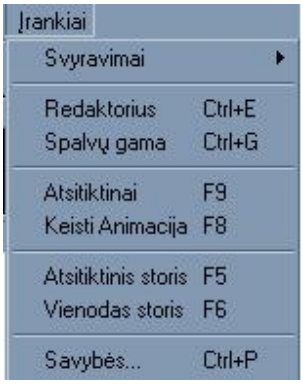
Komanda *„Bylos turinys“*. (4.12 pav. 3)

Komanda skirta bylos turinio rodymo arba paslėpimo režimų aktyvavimui. Šalia šios komandos yra žymeklis („varnelė“), kuris rodo bylos turinio būseną. T. y., jam esant bylos turinys yra matoma, kitaip – ne.



4.12 pav. Pagalbinės juostos

3.3.1.4. Meniu punktas „ Įrankiai“.



Šis meniu 4.13 pav. skirtas svyravimų nustatymui, redaktoriaus langui aktyvuoti, spalvų gamos parinkimui, atsitiktinio fraktalo parinkimui, paleisti ir sustabdyti animaciją bei savybių langui aktyvuoti.

4.13 pav. Meniu "Įrankiai"

Komanda - meniu „*Svyravimai*“.

Šis meniu 4.14 pav. skirtas svyravimų parinkimui. Pasirinkę norimą būseną, ji bus iš karto aktyvuota, fraktalas bus perskaičiuojamas pagal atitinkamą pokytį. Atsitiktiniu atveju kiekvieną kartą pagal nutylėjimą bus parinktas bet kuris iš sekančių pokyčių.

4.14 pav. Meniu "Svyravimai"

Komanda „A*tsitiktinis*“. (4.14 pav. 1)

Komanda skirta atsitiktiniam pokyčiui parinkti. Šalia šios komandos yra žymeklis („taškas“), kuris rodo dabartinę komandos būseną. T. y., jam esant atsitiktinė būsena yra parinkta, kitaip – ne.

Komanda „*Linijinis*“. (4.14 pav 2)

Komanda skirta linijiniam pokyčiui parinkti. Šalia šios komandos yra žymeklis („taškas“), kuris rodo dabartinę komandos būseną. T. y., jam esant linijinė būsena yra parinkta, kitaip – ne.

Komanda „*Sinusoidinis*“. (4.14 pav 3)

Komanda skirta sinusoidiniam pokyčiui parinkti. Šalia šios komandos yra žymeklis („taškas“), kuris rodo dabartinę komandos būseną. T. y., jam esant sinusoidinė būsena yra parinkta, kitaip – ne.

Komanda „*Sferinis*“. (4.14 pav 4)

Komanda skirta sferiniam pokyčiui parinkti. Šalia šios komandos yra žymeklis („taškas“), kuris rodo dabartinę komandos būseną. T. y., jam esant sferinė būsena yra parinkta, kitaip – ne.

Komanda „*Spiralinis*“. (4.14 pav 5)

Komanda skirta spiraliniam pokyčiui parinkti. Šalia šios komandos yra žymeklis („taškas“), kuris rodo dabartinę komandos būseną. T. y., jam esant spiralinė būsena yra parinkta, kitaip – ne.

Komanda „*Kūginis*“. (4.14 pav 6)

Komanda skirta kūginiam pokyčiui parinkti. Šalia šios komandos yra žymeklis („taškas“), kuris rodo dabartinę komandos būseną. T. y., jam esant kūginė būsena yra parinkta, kitaip – ne.

Komanda „*Polinis*“. (4.14 pav. 7)

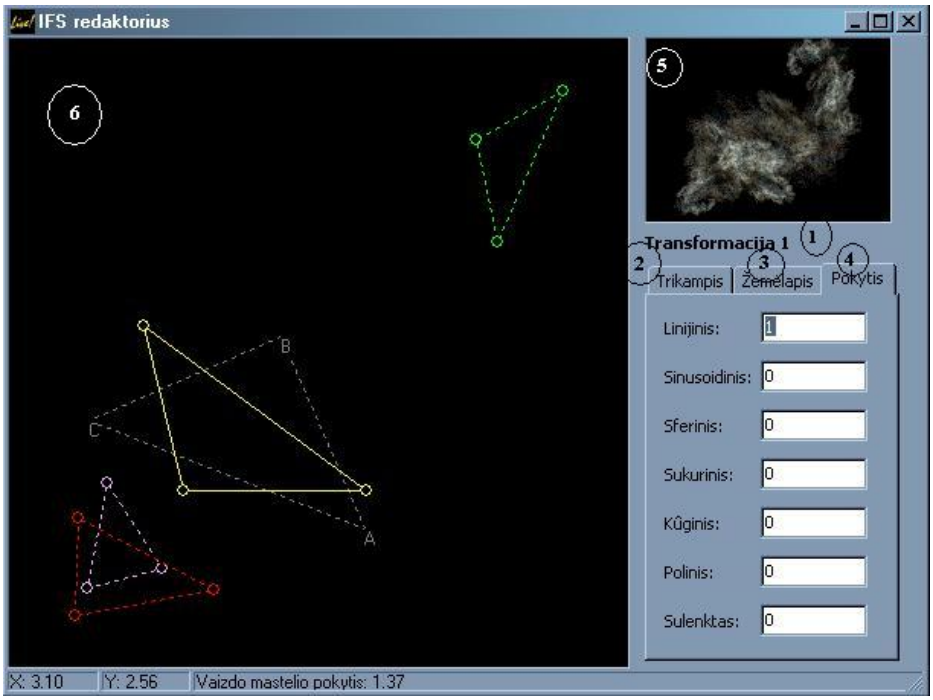
Komanda skirta poliniam(ašiniam) pokyčiui parinkti. Šalia šios komandos yra žymeklis („taškas“), kuris rodo dabartinę komandos būseną. T. y., jam esant ašinė būsena yra parinkta, kitaip – ne.

Komanda „*Lenktas*“. (4.14 pav 8)

Komanda skirta sulenktam pokyčiui parinkti. Šalia šios komandos yra žymeklis („taškas“), kuris rodo dabartinę komandos būseną. T. y., jam esant lenkta būsena yra parinkta, kitaip – ne.

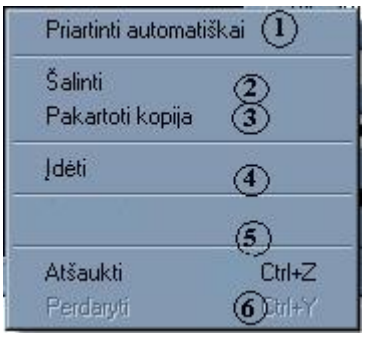
Komanda „*Redaktorius*“.

Skirta fraktalo redaktoriaus lango aktyvavimui. Iškviesti ją galima pasinaudojus meniu arba įrankių juostos pagalba (atitinkamas mygtukas parodytas kairėje). Pasirinkus šią komandą, ekrane pasirodys toks dialogo langas:



4.15 pav. IFS redaktorius

4.15 pav. fraktalą transformacinių trikampių pagalb 6 . Tam, kad tai padaryti – pažymėkite norimą transformaciją – trikampį – ir pelės pagalba stumkite jį į vieną ar kitą pusę. Galima dirbti ir su trikampių viršūnėmis, atitinkama viršūnė pažymėta A, B, C. Keičiant kiekvieną iš jų, keičiasi atitinkamo taško x ir y koordinatės, tuo pačiu ir atitinkamas transformacijos dydis. Ant trikampio lauko paspaudus dešinį pelės klavišą atsidaro komandų laukas:



4.16 pav. Komandų laukas

Šiame lange 4.16 pav. matomų komandų paskirtis:

1. Automatiškai priartinamas bendras vaizdas.
2. Pašalinamas pažymėtas trikampis – transformacija.
3. Padaroma pažymėto trikampio kopija.
4. Įdedamas naujas trikampis – transformacija.
5. Atšaukti paskutinę padarytą komandą.
6. Gražinti atšauktą komandą į pradinę būseną.

Lange 4.16 pav (5) matome realių laikų fraktalo vaizdą, keičiant trikampių padėtis galima matyti kaip keičiasi fraktalo vaizdas, uždarius redaktoriaus langą, toks vaizdas bus matomas pagrindiniame lange.

Laukas 4.16 pav (1) matome su kuria transformacija – trikampiu dabar dirbate.

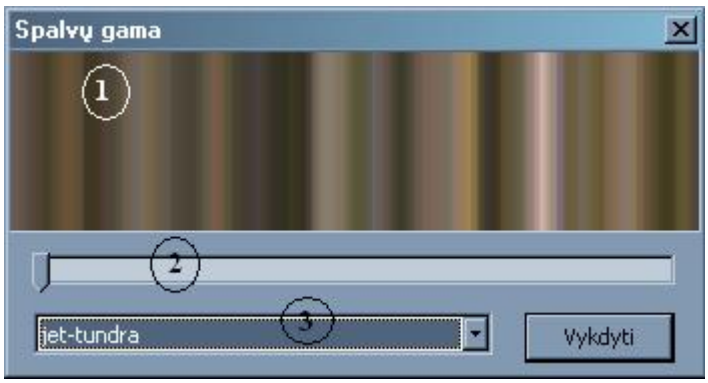
Parametrų laukas „*Pokytis*“ 4.16 pav (4) matome pokyčių skaitines vertes.

Parametrų laukas „*Žemėlapis*“ 4.16 pav. (3) matome fraktalo vaizdo skaitines vertes.

Parametrų laukas „*Trikampis*“ 4.16 pav. (2) matome trikampio skaitines vertes.

Komanda „*Spalvų gama*“.

Skirta fraktalo spalvų gamos lango aktyvavimui. Iškviesti ją galima pasinaudojus meniu arba įrankių juostos pagalba (atitinkamas mygtukas parodytas kairėje). Pasirinkus šią komandą ekrane pasirodys toks dialogo langas:



4.17 pav. Spalvų gamos langas

Šiame lange (4.17 pav.) galite pasirinkti norimą spalvų gamą. Ją aktyvavus pagrindiniame lange fraktalas bus nuspalvintas jūsų pasirinktomis spalvomis:

1. Spalvų gamos vaizdas.
2. Spalvų gamos perėjimas.
3. Pasirinkimo laukas, galima pasirinkti norimą spalvų gamą.

Komanda „*Atsitiktinai*“.

Skirta atsitiktinio fraktalo generavimui . Iškviesti ją galima pasinaudojus meniu pagalba. Pasirinkus šią komandą ekrane bus sugeneruotas atsitiktinai parinktas fraktalas.

Komanda „*Keisti Animacija*“.

Skirta esamo fraktalo animacijai paleisti. Iškviesti ją galima pasinaudojus meniu. Pasirinkus šią komandą ekrane esantis fraktalas pradeda keistis, t.y., pagal kiekvieną transformaciją - trikampis juda aplink savo centras. Pakartotinai iškvietus komandą, animacija sustabdoma.

Komanda „*Atsitiktinis storis*“.

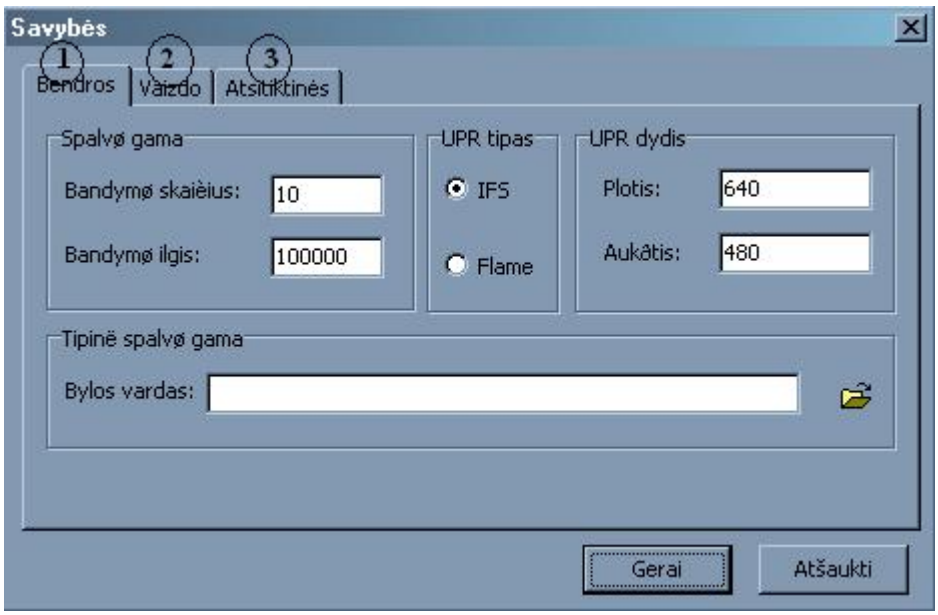
Skirta atsitiktinio storio nustatymui. Iškviesti ją galima pasinaudojus meniu pagalba. Pasirinkus šią komandą ekrane atsitiktinai bus sugeneruotas fraktalo storis.

Komanda „*Vienodas storis*“.

Skirta vienodo storio nustatymui. Iškviesti ją galima pasinaudojus meniu pagalba. Pasirinkus šią komandą ekrane bus sugeneruotas vienodas fraktalo storis.

Komanda „*Savybės*“.

Skirta savybių lango aktyvavimui. Iškviesti ją galima pasinaudojus meniu pagalba. Pasirinkus šią komandą ekrane pasirodys toks dialogo langas:

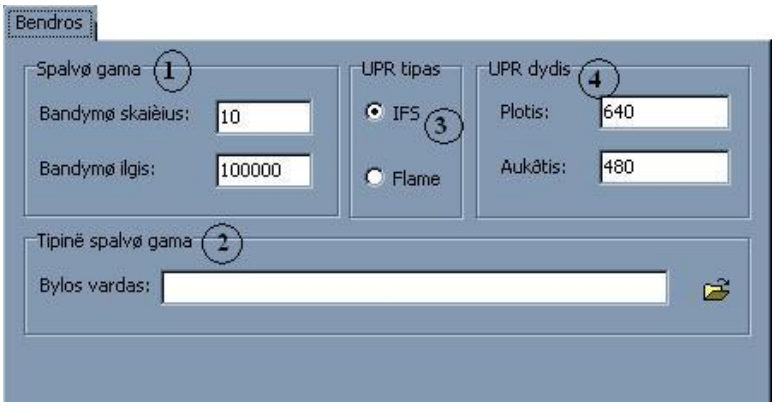


4.18 pav. Savybių langas

Šiame lange (4.18 pav.) galima parinkti tokias savybes:

1. Bendros.
2. Vaizdo.
3. Atsitiktinės.

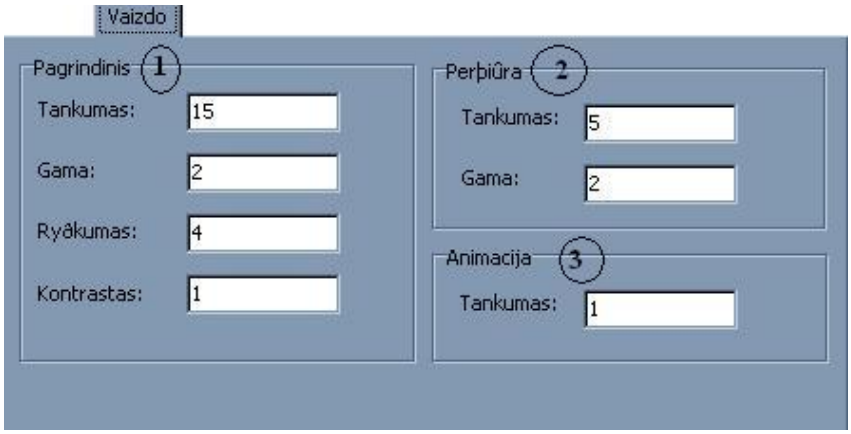
Savybių langas 4.18 pav. (1) – *„Bendros“:*



4.18 pav. Bendros savybės

1. Spalvų gamos parametrų nustatymas.
2. Parenkama tipinė spalvų gama.
3. URP tipas: IFS arba fraktalinis.
4. UPR dydžio nustatymas.

Savybių langas 4.19 pav. (2) – „*Vaizdo“:*



4.19 pav. Vaizdo savybės

1. Pagrindinis: gamos, tankumo, kontrasto bei ryškumo nustatymas.
2. Peržiūra: tankumo, gamos nustatymas.
3. Animacija: tankumo nustatymas.

Savybių langas 4.19 pav. (3) – „*Vaizdo“:*

4.20 pav. Atsitiktinės savybės

Šiame lange 4.20 pav. galima parinkti atsitiktinių maksimalų transformacijų skaičių ir minimalų, bei uždėjus „varnelę“ – bus generuojamas atsitiktinis gradientas.

IŠVADOS

1. Magistriniame darbe buvo realizuota vartotojiška ir mokomoji fraktalinių vaizdų kūrimo sistema. Išanalizavus fraktalų generavimo bei jų panadojimo principus, suprojektuota fraktalinio interpoliavimo priemonė.
2. Sistemos pagalba realizuojamos penkios pagrindinės funkcijos:
    a. Fraktalo generavimas.
    b. Fraktalinės interpoliacines funkcijos veikimas.
    c. Spalvų gamos pritaikymas.
    d. Fraktalo redagavimas.
    e. Fraktalo eksportavimas į kitus formatus.
3. Išanalizavus fraktalinės interpoliacines funkcijas gauti fraktaliniai tiesės, sinusoidinės, sferinio, kūginio, spiralinio bei polinio interpoliavimo vaizdai.
4. Kadangi tiriamojo darbo objektas yra teoriniai ir praktiniai fraktalinių interpoliacinių funkcijų sudarymo aspektai, tai pagrindinis dėmesys algebriniams fraktalams, gaunamiems taikant keruotąsias (afiniųjų) funkcijų sistemas.
5. Panaudotas atsitiktinių iteracijų algoritmas sukurti iteruotųjų funkcijų sistemų atraktorių (fraktalinius vaizdus).
6. Kadangi elementariosios funkcijos (tokios kaip sinusas, kosinusas, daugianariai ir panašiai.) sudaro tradicinio, visuotinai priimto eksperimentinių duomenų analizės metodo pagrindą, todėl buvo naudotos fraktalinės interpoliacinės funkcijos.
7. Vartotojiška ir mokomoji fraktalinių vaizdų kūrimo sistema yra lanksti vartotojo pažiūriu. Nebūtinas programos instaliavimas. Sistemą namų vartotojas gali prisitaikyti pagal savo poreikius (redaktorius, kuris leidžia vartotojui keisti sistemos meniu ir taip pritaikyti vartotojišką ir mokomąją fraktalinių vaizdų kūrimo sistemą įvairioms aplinkoms).
8. Sistema gali naudotis ir kitų populiarių programų sukurtais fraktaliniais formatais *.ifs, turėdamas ir savąjį formatą *.fla. Fraktalus galima panaudoti ir kitose grafikos sferose, paveikslėlį galima užsaugoti *.bmp formatu, kurį palaiko dauguma su grafiniais vaizdais dirbti skirtų programų.
9. Vartotojiška ir mokomoji fraktalinių vaizdų kūrimo sistema dirba Windows 9x, Me, NT ir XP operacinių sistemų platformose. Programa užima 1.88 Mb vietos kietajame diske. Programą sudaro 4 failai: Fraktalai.exe, CMAP.UGR, IFS.UGR, Pagalba.pdf. Norint tvarkingai paleisti programą svarbu, kad visi failai būtų tame pačiame kataloge. Programos instaliuoti nereikia, užtenka paleisti „frakatalai.exe“ ir ji jau veikia.

ANNOTATION IN ENGLISH (ANOTACIJA)

Concumer and educational flame fractal development system

„Fraktalai“ is a program for creating and editing IFS and Flame Fractal parameters. Once a nice fractal has been found the parameters can be exported to IFS for use with my **Iterated Function Systems** coloring algorithm. The IFS files that it creates can also be viewed in other fractal programs. Iterated Function Systems is used in image compression. It also becomes very popular as a way of producing fractal images. „ Fraktalai” only makes a distinction between Iterated Function Systems and Flame Fractals when loading and saving parameters.

The algorithm that generates random fractals is designed to create reasonably interesting parameters but it has no sense of beauty, so you will probably want to make some adjustments.

The supplied gradients were created from image files using a special algorithm (you may have noticed the names of a few famous paintings). This sort of gradient looks very nice with Flame Fractals, and “Fraktalai” has the ability to create similar gradients from BMP or JPEG images.

LITERATŪROS SĄRAŠAS

1. ANTANAS VIDŽIŪNAS. *Delfi 6 programavimas ir vaizdiniai*. Kaunas, 2002.
2. GINTAUTAS BAREIKIS. *Paskaitų konspektas „FRAKTALAI“*. Vilniaus Universitetas, Matematikos fakultetas, 2002.
3. VYTAUTAS BARZDAITIS. *Delfi programavimo pagrindai.* Kaunas, 2004.
4. J. VALANTINAS. Fraktalinė geometrija.— K.: Technologija, 1999.—p.10—69,121— 153.
5. B. B. MANDELBROT. The Fractal Gheometry of Nature. 1977
6. B. VILJAMS Trading Chaos.
7. HTTP://COMMUNITY.BORLAND.COM/DELPHI/
8. HTTP://SPANKY.TRIUMF.CA/WWW/FRACTINT/FRM-TUT/FRM-TUTOR.HTML#FORMULA_STRUCTURE_ANCHOR
9. HTTP://WWW.FRACTALISM.COM/FRACTALS/GEOMETRY.HTM
10. HTTP://SPANKY.TRIUMF.CA/WWW/FRACTINT/FRACTINT.HTML
11. HTTP://WWW.MIF.VU.LT/TTSK/BYLOS/BG/FR.HTML

1 PRIEDAS PAGRINDINIO METODO ALGOTITMAS DELPHI KALBA

(Kiti programos kodai talpinami CD priede)

```
uses SysUtils, Dialogs, Windows, Graphics, MainForm;


{procedure Log(text: string);
var
  lfile: TextFile;
begin
  Assign(lfile, LogFile);
  if fileexists(LogFile) then Append(lfile) else
    ReWrite(lfile);
  Writeln(lfile, text);
  closeFile(lfile);
end;}

function TDrawThrd.GetBounds(Height, Width, MaxIter: integer; lg: boolean):
TBounds;
{ Find scale and center of image (from screensaver code) }
var
  fn: integer;
  a: double;
  tx, ty: double;
  r, r2, c1, c2: double;
  x, y: double;
  minx, maxx: double;
  miny, maxy: double;
  CenterX, CenterY, yLength, xLength: double;
  deltax, deltay: double;
  pixels_per_unit: double;
  i, j: integer;
  cntminx, cntmaxx: integer;
  cntminy, cntmaxy: integer;
  LimitOSPoints: integer;
  Points: array[0..1000, 0..1] of double;
  function min(a, b: double): double;
  begin
    if a < b then
      min := a
    else
      min := b;
  end;

  function max(a, b: double): double;
  begin
    if a > b then
      max := a
    else
      max := b;
  end;

begin
  i := 0;
  randseed := 12346;
  x := (random * 2) - 1;
  y := (random * 2) - 1;
  try
    while i < MaxIter do
    begin
    // Pick a random transform
      r := random;
      for fn := 0 to FTransforms - 1 do
```

```
  begin
    r := r - fIFS[fn].p;
    if (r < 0) then break;
  end;
  tx := fIFS[fn].a * x + fIFS[fn].b * y + fIFS[fn].e;
  ty := fIFS[fn].c * x + fIFS[fn].d * y + fIFS[fn].f;
  x := 0.0;
  y := 0.0;
// Add proportional amounts of each variation
  if (fIFS[fn].vars[0] > 0.0) then
  // Linear
  begin
    x := x + fIFS[fn].vars[0] * tx;
    y := y + fIFS[fn].vars[0] * ty;
  end;
  if (fIFS[fn].vars[1] > 0.0) then
  // Sinusoidal
  begin
    x := x + fIFS[fn].vars[1] * sin(tx);
    y := y + fIFS[fn].vars[1] * sin(ty);
  end;
  if (fIFS[fn].vars[2] > 0.0) then
  // Spherical
  begin
    r2 := tx * tx + ty * ty + 0.000001;
    x := x + fIFS[fn].vars[2] * tx / r2;
    y := y + fIFS[fn].vars[2] * ty / r2;
  end;
  if (fIFS[fn].vars[3] > 0.0) then
  // Swirl
  begin
    r2 := tx * tx + ty * ty;
    c1 := sin(r2);
    c2 := cos(r2);
    x := x + fIFS[fn].vars[3] * (c1 * tx - c2 * ty);
    y := y + fIFS[fn].vars[3] * (c2 * tx + c1 * ty);
  end;
  if (fIFS[fn].vars[4] > 0.0) then
  // Horseshoe
  begin
    a := 0.0;
    if ((tx < -EPS) or (tx > EPS) or (ty < -EPS) or (ty > EPS)) then
      a := arctan2(tx, ty);
    c1 := sin(a);
    c2 := cos(a);
    x := x + fIFS[fn].vars[4] * (c1 * tx - c2 * ty);
    y := y + fIFS[fn].vars[4] * (c2 * tx + c1 * ty);
  end;
  if (fIFS[fn].vars[5] > 0.0) then
  // Polar
  begin
    a := 0.0;
    if ((tx < -EPS) or (tx > EPS) or (ty < -EPS) or (ty > EPS)) then
      a := arctan2(tx, ty) / pi;
    x := x + fIFS[fn].vars[5] * a;
    y := y + fIFS[fn].vars[5] * (sqrt(tx * tx + ty * ty) - 1.0);
  end;
  if (fIFS[fn].vars[6] > 0.0) then
  // Bent
  begin
    c1 := tx;
    c2 := ty;
    if (c1 < 0.0) then
      c1 := 2 * c1;
```

```
    if (c2 < 0.0) then
      c2 := c2 / 2.0;
    x := x + fIFS[fn].vars[6] * c1;
    y := y + fIFS[fn].vars[6] * c2;
  end;
  inc(i);
  if i >= 0 then
  begin
    Points[i, 0] := x;
    Points[i, 1] := y;
  end
end;
LimitOSPoints := Round(0.05 * MaxIter);
minx := 1E10;
maxx := -1E10;
miny := 1E10;
maxy := -1E10;
for i := 0 to MaxIter - 1 do
begin
  minx := min(minx, Points[i, 0]);
  maxx := max(maxx, Points[i, 0]);
  miny := min(miny, Points[i, 1]);
  maxy := max(maxy, Points[i, 1]);
end;
deltax := (maxx - minx) * 0.25;
maxx := (maxx + minx) / 2;
minx := maxx;
deltay := (maxy - miny) * 0.25;
maxy := (maxy + miny) / 2;
miny := maxy;
for j := 0 to 10 do
begin
  cntminx := 0;
  cntmaxx := 0;
  cntminy := 0;
  cntmaxy := 0;
  for i := 0 to MaxIter - 1 do
  begin
    if (Points[i, 0] < minx) then Inc(cntminx);
    if (Points[i, 0] > maxx) then Inc(cntmaxx);
    if (Points[i, 1] < miny) then Inc(cntminy);
    if (Points[i, 1] > maxy) then Inc(cntmaxy);
  end;
  if (cntMinx < LimitOSPoints) then
  begin
    minx := minx + deltax;
  end
  else
  begin
    minx := minx - deltax;
  end;
  if (cntMaxx < LimitOSPoints) then
  begin
    maxx := maxx - deltax;
  end
  else
  begin
    maxx := maxx + deltax;
  end;
  deltax := deltax / 2;
  if (cntMiny < LimitOSPoints) then
  begin
    miny := miny + deltay;
  end
```

```
      else
      begin
        miny := miny - deltay;
      end;
      if (cntMaxy < LimitOSPoints) then
      begin
        maxy := maxy - deltay;
      end
      else
      begin
        maxy := maxy + deltay;
      end;
      deltay := deltay / 2;
    end;
    centerX := (minx + maxx) / 2;
    centerY := (miny + maxy) / 2;
{    if lg then begin
      Log('minx: ' + FloatToStr(minx));
      Log('maxx: ' + FloatToStr(maxx));
      Log('miny: ' + FloatToStr(miny));
      Log('maxy: ' + FloatToStr(maxy));
      Log('CenterX: ' + FloatToStr(CenterX));
      Log('CenterY: ' + FloatToStr(CenterY));
    end;}
    Result.CenterX := CenterX;
    Result.CenterY := CenterY;
    if ((maxx - minx) > 0.001) and ((maxy - miny) > 0.001) then
      pixels_per_unit := 0.7 * Min(Width / (maxx - minx), Height / (maxy -
miny))
    else
      pixels_per_unit := 10;
    Result.scale := pixels_per_unit;
{    if lg then
      Log('pixels_per_unit: ' + FloatToStr(Pixels_per_unit));}
    { Get #magn for UPR }
    xLength := maxx - minx;
    yLength := maxy - miny;
    if xLength >= yLength then
    begin
      Result.Magn := 1 / xLength * 2;
    end
    else
    begin
      Result.Magn := 1 / yLength * 2;
    end;
  except on E: EMathError do
    begin
{      Log('DrawThread.GetBounds: ' + E.Message);}
    end;
  end;
end;



function TDrawThrd.Iterate(var x, y, c: extended): boolean;
var
  fn: integer;
  a: extended;
  tx, ty: extended;
  r, r2, c1, c2: extended;
begin


  Result := True;
```

```
  if terminated then
    Exit;
  try
    // Pick a random transform
    r := random;
    for fn := 0 to FTransforms - 1 do
    begin
      r := r - Fifs[fn].p;
      if (r < 0) then break;
    end;
    // Compute color of next point
    c := (c + Fifs[fn].color) / 2.0;
    tx := Fifs[fn].a * x + Fifs[fn].b * y + Fifs[fn].e;
    ty := Fifs[fn].c * x + Fifs[fn].d * y + Fifs[fn].f;
    // Big numbers cause invalid floating point operations
    if (abs(tx) > 1E200) or (abs(tx) < 1E-200) or (abs(ty) > 1E200) or
(abs(ty) < 1E-200) then
    begin
      Result := False;
      Exit
    end;
    x := 0.0;
    y := 0.0;
    // Add proportional amounts of each variation
    if (Fifs[fn].vars[0] > 0.0) then
      // Linear
    begin
      x := x + Fifs[fn].vars[0] * tx;
      y := y + Fifs[fn].vars[0] * ty;
    end;
    if (Fifs[fn].vars[1] > 0.0) then
      // Sinusoidal
    begin
      x := x + Fifs[fn].vars[1] * sin(tx);
      y := y + Fifs[fn].vars[1] * sin(ty);
    end;
    if (Fifs[fn].vars[2] > 0.0) then
      // Spherical
    begin
      r2 := tx * tx + ty * ty + 0.000001;
      x := x + Fifs[fn].vars[2] * tx / r2;
      y := y + Fifs[fn].vars[2] * ty / r2;
    end;
    if (Fifs[fn].vars[3] > 0.0) then
      // Swirl
    begin
      r2 := tx * tx + ty * ty;
      c1 := sin(r2);
      c2 := cos(r2);
      x := x + Fifs[fn].vars[3] * (c1 * tx - c2 * ty);
      y := y + Fifs[fn].vars[3] * (c2 * tx + c1 * ty);
    end;
    if (Fifs[fn].vars[4] > 0.0) then
      // Horseshoe
    begin
      a := 0.0;
      if ((tx < -EPS) or (tx > EPS) or (ty < -EPS) or (ty > EPS)) then
        a := arctan2(tx, ty);
      c1 := sin(a);
      c2 := cos(a);
      x := x + Fifs[fn].vars[4] * (c1 * tx - c2 * ty);
      y := y + Fifs[fn].vars[4] * (c2 * tx + c1 * ty);
    end;
    if (Fifs[fn].vars[5] > 0.0) then
```

```
      // Polar
    begin
      a := 0.0;
      if ((tx < -EPS) or (tx > EPS) or (ty < -EPS) or (ty > EPS)) then
        a := arctan2(tx, ty) / pi;
      x := x + Fifs[fn].vars[5] * a;
      y := y + Fifs[fn].vars[5] * (sqrt(tx * tx + ty * ty) - 1.0);
    end;
    if (Fifs[fn].vars[6] > 0.0) then
      // Bent
    begin
      c1 := tx;
      c2 := ty;
      if (c1 < 0.0) then
        c1 := 2 * c1;
      if (c2 < 0.0) then
        c2 := c2 / 2.0;
      x := x + Fifs[fn].vars[6] * c1;
      y := y + Fifs[fn].vars[6] * c2;
    end;
  except on E: EMathError do
{    begin
      Log('DrawThread.Iterate: ' + E.Message);
      log(FloatToStr(x));
      log(FloatToStr(y));
      Result := False;
    end;}
  end;
end;

function TDrawThrd.Calculate: boolean;
var
  it, sx, sy, sz, nsamples, clr_index, cmidx
    : integer;
  ls, area, k1, k2: extended;
  lastpt: array[0..2] of extended; newpt: array[0..2] of extended;
  scl_cmap: array[0..255, 0..3] of integer; lc: array[0..3] of extended;
begin
  it := 0; nsamples := 0;
  FCompleted := False;
  if terminated then
  begin
    Result := False;
    Exit;
  end;
  try
    { Ugly -- needed because I use a fixed array size for virtual screen }
    if (FWidth > 1280) or (FHeight > 1024) then
    begin
      Result := False;
      Exit;
    end;
    { Scale colors down to the 0 - White Level range }
    cmidx := 0;
    while (cmidx <= 255) do
    begin
      scl_cmap[cmidx, 0] := round(FPal.Red[cmidx] * (White_level / 255));
      scl_cmap[cmidx, 1] := round(FPal.Green[cmidx] * (White_level / 255));
      scl_cmap[cmidx, 2] := round(FPal.Blue[cmidx] * (White_level / 255));
      scl_cmap[cmidx, 3] := White_level;
      cmidx := cmidx + 1;
    end;
    { Intitialise screen array }
    for sx := 0 to FWidth - 1 do
```

```
      for sy := 0 to FHeight - 1 do
        for sz := 0 to 3 do
        begin
          FPix[sx, sy, sz] := 0;
        end;
    Result := True;
    //    FCompleted := False;
    nsamples := FWidth * FHeight * FDensity;
    { Generate the starting point, (-1: 1, -1: 1, 0: 1) }
    randseed := 12346;
    newpt[0] := (random * 2) - 1;
    newpt[1] := (random * 2) - 1;
    newpt[2] := random;
    it := -15; // sx := 0; sy := 0;
    while (it < nsamples) and (not Terminated) do
    begin
      { Copy new to old }
      lastpt[0] := newpt[0];
      lastpt[1] := newpt[1];
      lastpt[2] := newpt[2];
      if iterate(newpt[0], newpt[1], newpt[2]) then
      begin
        if it > 0 then // Past the fuse
        begin
          { Scale to screen }
          sx := round((newpt[0] - FBounds.CenterX) * (FBounds.scale *
FZoom) + FWidth / 2);
          sy := round((-newpt[1] - (-FBounds.CenterY)) * (FBounds.scale *
FZoom) + FHeight / 2);
          if (sx >= 0) and (sx < FWidth) and (sy >= 0) and (sy < FHeight)
then
          begin
            { try clr_index as byte }
            clr_index := round(newpt[2] * 256.0);
            if (clr_index < 0) then
              clr_index := 0
            else if (clr_index > 255) then
              clr_index := 255;
            { Add the color to the accumulator }
            FPix[sx, sy, 0] := FPix[sx, sy, 0] + scl_cmap[clr_index, 0];
            FPix[sx, sy, 1] := FPix[sx, sy, 1] + scl_cmap[clr_index, 1];
            FPix[sx, sy, 2] := FPix[sx, sy, 2] + scl_cmap[clr_index, 2];
            FPix[sx, sy, 3] := FPix[sx, sy, 3] + scl_cmap[clr_index, 3];
          end;
        end;
      end
      else
      begin
        { The iteration failed so we pick a new random point and start
again }
        newpt[0] := (random * 2) - 1;
        newpt[1] := (random * 2) - 1;
        newpt[2] := random;
      end;
      inc(it);
    end; // Main Loop
    k1 := FContrast * FBrightness * prefilter_white * 268.0 / 256;
    area := FHeight * FWidth / (FBounds.scale * FBounds.scale);
    k2 := 1 / (FContrast * area * White_level * FDensity);
    //  Loop through each pixel and perform log calculation
    sx := 0;
    while (sx < FWidth) do
    begin
      sy := 0;
```

```
      while (sy < Fheight) do
      begin
        if (FPix[sx, sy, 3] > 0) then
        begin
          lc[0] := FPix[sx, sy, 0];
          lc[1] := FPix[sx, sy, 1];
          lc[2] := FPix[sx, sy, 2];
          lc[3] := FPix[sx, sy, 3];
          // Scale by the log of the number of occurrences
          ls := k1 * log2(1.0 + lc[3] * k2) / lc[3];
          lc[0] := lc[0] * ls;
          lc[1] := lc[1] * ls;
          lc[2] := lc[2] * ls;
          lc[3] := lc[3] * ls;
          FPix[sx, sy, 0] := round(lc[0] + 0.5);
          FPix[sx, sy, 1] := round(lc[1] + 0.5);
          FPix[sx, sy, 2] := round(lc[2] + 0.5);
          FPix[sx, sy, 3] := round(lc[3] + 0.5);
        end;
        sy := sy + 1;
      end;
      sx := sx + 1
    end;
    Fg := 1 / FGamma;
  except on E: EMathError do
    begin
{      Log('DrawThread.Calculate: ' + E.Message);}
      Result := False;
    end;
  end;
  // Reached the end so it's a good draw
  if it >= nsamples then
  begin
    FCompleted := True;
  end;
end;

procedure TDrawThrd.Execute;
begin
  if FNewBounds then FBounds := GetBounds(FHeight, FWidth, 1000, false);
  Calculate;
  PostMessage(frmMain.Handle, WM_THREAD_COMPLETE, 0, 0);
end;

end.
```

2 PRIEDAS MOKOMOJI MEDŽIAGA KOMPAKTINĖ PLOKŠTELĖ