Fraktálok és számítógépes grafika

- Szakdolgozat -

Készítette: **Mansaré Anna Manty**
(Matematika BSc, Elemző szakirány)

Témavezető: **Buczolich Zoltán**
(Analízis Tanszék, Matematikai Intézet)

Eötvös Loránd Tudományegyetem
Természettudományi Kar
Budapest, 2014

Tartalomjegyzék

1. Előszó **3**

2. Matematikai háttér **4**
2.1. Iterált függvényrendszerek 4
2.1.1. Metrikus tér 4
2.1.2. Iterált Függvényrendszerek 5
2.2. Fixpont, attraktor, Hausdorff dimenzió 6
2.2.1. Fixpont-tétel 6
2.2.2. Attraktor 6
2.2.3. Önhasonlóság 7
2.2.4. Hausdorff mérték és dimenzió 7

3. A Fraktálok **9**
3.1. Mi a fraktál? 9
3.2. Egyszerűen generálható példák 10
3.2.1. Cantor halmaz 10
3.2.2. Mandelbrot és Julia halmazok 12
3.2.3. Koch görbe 13
3.2.4. Sierpinski háromszög és szőnyeg 14
3.2.5. Menger szivacs 16
3.3. Fraktálok a természetben 18
3.3.1. Vízesés 19

4. Kik használják és hogyan? **21**
4.1. Számítógépes grafika 21
4.1.1. Tájképek 21
4.1.2. Véletlenszerű középpontáthelyezés algoritmus 22
4.1.3. Gyémánt-négyzet (Diamond-Square) algoritmus 24
4.2. Képtömörítés 27
4.2.1. Bevezetés 27
4.2.2. Sárkány elkódolása (IFS) 27
4.2.3. Elkódolás és visszafejtés 28
4.3. Saját program 30
4.3.1. Az alapötlet 31
4.3.2. A program működésének elmélete 31
4.3.3. Méret és felbontás 32
4.3.4. A megfelelő adatszerkezet 33
4.3.5. A gyermekek kiszámítása 34
4.3.6. A bal alsó hiba 34
4.3.7. Animációk 36
4.3.8. Véletlen szőnyeg játék 36
4.3.9. Véletlen Sierpinski háromszög 37

5. CD Melléklet **39**
5.1. Waterfall program 39
5.2. Fractal program 39
5.2.1. Megjegyzés 39
5.3. Szakdolgozat 39

6. Konklúzió **40**

1. Előszó

Szakdolgozatom témájául a fraktálokat választottam. Ez egy viszonylag új terület a matematikában, vizsgálatukat a számítógépek megjelenése forradalmasította, bár már a XX. század előtt is fogalkoztatták a matematikusokat ezek a különös geometriai alakzatok.
Amint sikerült őket jobban megismerni, kiderült hogy a természetben is léptennyomon megtalálhatók és egyre több területen kezdték alkalmazni őket, mint például: szívritmus vizsgálata, egyensúly vizsgálata, rákkutatás, meteorológia, számítógépes grafika, digitális képfeldolgozás.
Ebben a szakdolgozatban az utóbbi két alkalmazással foglalkoztam:

- hogyan lehet gyorsan, egyszerűen élethűnek tűnő tájképeket rajzolni számítógép segítségével és
- hogyan lehet a képeket hatékonyabban tömöríteni és felbontásukat növelni

a fraktálok és tulajdonságaik felhasználásával.
Néhány matematikai fogalom tisztázása után - pontos definíció híján - tulajdonságaikon és néhány szemléletes példán keresztül próbálom megmutatni mik a fraktálok. Ehhez egy számítógépes programot is készítettem, mely egy természet által létrehozott fraktált próbál leegyszerűsítve modellezni. Ezután rátérve az alkalmazásokra, a számítógépes grafikában egy a hegyek (domborzat) képének generálására használt algoritmust mutatok be, majd a képtömörítés következik. A képek fraktál-tömörítésének (fractal image compression) elkódolásával (encoding), annak alapötletével, elméletével és visszafejtésével (decoding) fogok foglalkozni, valamint az általam szemléltetés gyanánt írt másik program készítésének mérföldköveit írom le.

2. Matematikai háttér

2.1. Iterált függvényrendszerek

2.1.1. Metrikus tér

Mielőtt rátérnénk a fraktálokra, szükség van néhány alapfogalom és tulajdonság felelevenítésére, definiálására.
Meg kell ismerkednünk az **Iterált Függvényrendszerek**kel - ami fraktálok generálásának egy módszere (Iterated Function Systems, **IFS**) - és alkalmazásának eredményével. Ám előtte még emlékezzünk vissza a **metrikus tér** definíciójára is.

2.1.1.1. Emlékeztető. *$M = (X, d)$ **metrikus tér**, ahol X egy nemüres halmaz és $d : X \times X \mapsto [0, \infty)$ leképezés a távolság. A távolságnak teljesítenie kell a következő feltételeket minden $x, y, z \in X$-re:*

- $d(x, y) \geq 0$,
- $d(x, y) = 0 \Longleftrightarrow x = y$,
- $d(x, y) = d(y, x)$,
- $d(x, y) \leq d(x, z) + d(z, y)$.

2.1.1.2. Emlékeztető. *Az (X, d) metrikus tér **teljes**, ha minden Cauchy-konvergens sorozat konvergens, vagyis ha az $x_0, x_1, \ldots$ sorozathoz minden $\epsilon > 0$ esetén létezik olyan $N \in \mathbb{N}$ küszöb, hogy minden $m, n > N$-re $d(x_n, x_m) < \epsilon$, akkor létezik $x \in X$ pont, amire $x_n \longrightarrow x$.*

2.1.2. Iterált Függvényrendszerek

Esetünkben X a hétköznapi értelemben vett háromdimenziós tér, illetve a kétdimenziós sík pontjainak halmaza lesz.
Most nézzük meg, mi is egy iterált függvényrendszer.

2.1.2.1. Definíció. *Legyen (X, d) egy teljes metrikus tér és*

$$\mathcal{F} = \{ \ f_i \ \}, \ (i = 1 \dots N) \, , \ N \leq \infty$$

X-en folytonos leképezések egy legalább kételemű, véges halmaza.
*Ekkor $(X, \mathcal{F})$-et **iterált függvényrendszer**nek nevezzük.*
*(Továbbiakban **IFS**.)*

2.1.2.2. Definíció. *Legyen (X, d) egy teljes metrikus tér és $\mathcal{F} = \{ \ f_i \ \}, (i = 1 \dots N)$ egy IFS. Tegyük fel, hogy minden i-re f_i **kontrakció**, azaz minden $1 \leq j \leq N$ és minden $x, y \in X$ esetén*

$$d(f_j(x), f_j(y)) \leq \lambda_j d(x, y),$$

ahol $0 < \lambda_j < 1$.
*Ekkor azt mondjuk, hogy $(X, \mathcal{F})$ **Hiperbolikus (kontraktív) IFS**,*
és $\max \mid \lambda_j \mid$ a kontrakciós tényező.

A jelölések egyszerűsítése érdekében egy IFS leírására vezessük be a következő operátort.

2.1.2.3. Definíció. *Legyen $(X, \mathcal{F})$ egy hiperbolikus IFS és*

$$H : Y \mapsto \bigcup_{i=1}^{N} f_i(Y),$$

ahol $Y \subseteq X$.
*Ekkor a H operátort **Hutchinson (vagy Barnsley) operátor**nak nevezzük.*

Mivel az operátort kompakt halmazokon alkalmazzuk, a halmazok Hausdorff távolságának definiálása után beszélhetünk a Hutchinson-operátorról, mint egy leképzésről, mely kompakt halmazokhoz rendeli kompakt halmazok véges unióját, ami szintén kompakt lesz.

2.1.2.4. Definíció. *Legyen (X, d) metrikus tér és $K, L \subseteq X$ nemüres halmazok. Ekkor K és L **Hausdorff távolsága***

$$d_H(K, L) = \max\{\sup_{k \in K} \inf_{l \in L} d(k, l), \sup_{l \in L} \inf_{k \in K} d(k, l)\}.$$

2.2. Fixpont, attraktor, Hausdorff dimenzió

2.2.1. Fixpont-tétel

A fentiek definiálása után elevenítsük fel a Banach-féle fixpont-tételt is. Most, hogy már rendelkezésünkre áll a Hutchinson operátor, azt kell megvizsgálni, mit kapunk eredményül iterált alkalmazásából.

2.2.1.1. Emlékeztető. *Ha egy $x \in X$-re $f(x) = x$, akkor x-et f* ***fixpont****jának nevezzük.*

2.2.1.1. Tétel. *Legyen X teljes metrikus tér és $f : X \mapsto X$ kontrakció (λ kontrakciós tényezővel). Ekkor*

- *f-nek létezik pontosan egy fixpontja,*
- *valamint tetszőleges x_0 kezdőpont esetén az $x_n := f(x_{n-1})$ iteráció konvergens, és $\lim_{n\to\infty} x_n = x^*$.*
  Továbbá, érvényes a $d(x^*, x_m) \leq M \cdot d(x_1, x_0) \cdot \lambda^m$ becslés, ahol $M \geq 0$ konstans.

2.2.2. Attraktor

Az $\mathbb{R}^n$ metrikus téren (számunkra $n = 2, 3$ lényeges) hiperbolikus IFS-eknek létezik egyértelmű kompakt (korlátos és zárt) attraktora, invariáns halmaza. Az attraktor megtalálásának egyszerű módja a fixpont-tétel mintájára, egy iteráció meghatározása.
Induljunk ki egy tetszőleges $A_0 \subseteq X$ pontból vagy halmazból és legyen

$$A_{n+1} = H(A_n),$$

az A attraktor pedig az a halmaz, melyre teljesül az $A = H(A)$ egyenlőség.
A fixpont-tétel következményeként láthatjuk, hogy ha H a függvényrendszer Hutchinson operátora (leképzése), akkor

$$\lim_{n\to\infty} H^n(Y) = A$$

bármely nemüres, kompakt $Y \subseteq X$ halmazra.

2.2.3. Önhasonlóság

Az önhasonlóság ezeknek az invariáns halmazoknak az a tulajdonsága, amely a leglényegesebb a későbbi alkalmazások szempontjából. A jelentése pedig az, hogy a halmaz, az alakzat egy részlete, illetve részletei felnagyítva visszaadják az eredeti egész struktúráját.

2.2.3.1. Definíció. *Azt mondjuk, hogy egy A halmaz* ***önhasonló****, ha létezik nem triviális (nem csak az identitásból álló) hasonlóságok egy olyan $S = \{S_k\}_{k=1}^N$ halmaza, melyre*

$$A = \bigcup_{k=1}^{N} S_k(A).$$

Most, hogy a legfontosabb fogalmakat tisztáztuk, megpróbálom körülírni, bemutatni a fraktálokat.

2.2.4. Hausdorff mérték és dimenzió

2.2.4.1. Emlékeztető. *Legyen U az n-dimenziós euklideszi tér egy nemüres részhalmaza. Ekkor U* ***átmérője***

$$| U |= \sup\{d(x, y) : x, y \in U\},$$

azaz az összes U-beli pontpár távolsága közül a "legnagyobb".

2.2.4.1. Definíció. *Ha U_i egy megszámlálható (vagy véges), legfeljebb δ átmérőjű halmazokból álló halmazrendszer, amely lefedi $F \subset \mathbb{R}^n$-et, vagyis*

$$F \subset \bigcup_{i=1}^{\infty} U_i \quad \text{és} \quad 0 <| U_i |\leq \delta \text{ minden } i\text{-re},$$

akkor azt mondjuk hogy $\{U_i\}$, F egy δ-fedése.
Legyen s egy nemnegatív szám. Minden $\delta > 0$ esetén a következőképpen definiálhatjuk F ***δ-fedésű, s-dimenziós Hausdorff-előmértékét****:*

$$H_\delta^s(F) = \inf\{\sum_{i=1}^{\infty} | U_i |^s \colon \{U_i\} \ F \text{ egy } \delta\text{-fedése}\}.$$

Innen a Hausdorff-mérték meghatározásához már csak egyetlen lépés hiányzik. Tartsunk δ-val nullához.

2.2.4.2. Definíció. *A* $H^s(F) = \lim_{\delta \to 0+0} H^s_\delta(F)$ *határértéket* F*,* s***-dimenziós Hausdorff mértékének*** *nevezzük.*

Minden $F \subset \mathbb{R}^n$ esetén létezik ugyan ez a határérték, ám értéke lehet 0 vagy ∞ (és gyakran annyi is).
Figyeljük meg a Hausdorff mérték változását egy λ-szoros nagyítás során. Tudjuk, hogy egy kocka oldalhosszát λ-szorosára növelve (vagyis a kockát λ-szorosára nagyítva), a nagyított kocka oldalainak területe λ^2-szeresére, a test térfogata pedig λ^3-szörösére nő. Ezek az összefüggések általánosabban is megfogalmazhatók. Ha $F \subset \mathbb{R}^n$ és $\lambda > 0$, akkor

$$H^s(\lambda F) = \lambda^s H^s(F),$$

ahol $\lambda F = \{\lambda x : x \in F\}$, azaz az F halmaz λ-szoros nagyítása. Ugyanis ha $\{U_i\}$ F egy δ-fedése, akkor $\{\lambda U_i\}$ egy $\lambda\delta$-fedése λF-nek, ezért

$$H^s_{\lambda\delta}(\lambda F) \leq \sum \mid \lambda U_i \mid^s = \lambda^s \sum \mid U_i \mid^s \leq \lambda^s H^s_\delta(F).$$

Mivel ez minden U_i δ-fedés esetén fennáll, λ-val 0-hoz tartva, a $H^s(\lambda F) \leq \lambda^s H^s(F)$ egyenlőtlenséget kapjuk. λ-t $\frac{1}{\lambda}$-val, F-et pedig λF-fel helyettesítve, az ellenkező irányú egyenlőtlenséget is megkaphatjuk.

Megfigyelhetjük továbbá azt is, hogy egy adott F halmaz és $\delta < 1$ esetén, s-et növelve, $H^s_\delta(F)$ értéke, sőt $H^s(F)$ sem növekszik. Ennél azonban többet is mondhatunk: ha $t > s$ és $\{U_i\}$ F egy δ-fedése, akkor

$$\sum_i \mid U_i \mid^t \leq \delta^{t-s} \sum \mid U_i \mid^s .$$

Az infimumokat véve $H^t_\delta(F) \leq \delta^{t-s} H^s_\delta(F)$. δ-val 0-hoz tartva pedig láthatjuk, hogy ha $H^s(F) < \infty$, akkor $t > s$-re $H^t = 0$. Így látszik, hogy $(s, H^s(F))$ "grafikonja" egy bizonyos s értéknél a végtelenből 0-ra ugrik. Ezt az s értéket nevezzük az F halmaz **Hausdorff dimenziójának**. Formálisan:

$$\dim_H F = \inf\{s : H^s(F) = 0\} = \sup\{s : H^s(F) = \infty\},$$

amelyre

$$H^s(F) = \begin{cases} \infty & \text{ha} \quad s < \dim_H F \\ 0 & \text{ha} \quad s > \dim_H F. \end{cases}$$

3. A Fraktálok

3.1. Mi a fraktál?

A **fraktál**ok definíciója nem teljesen egyértelmű. Az elnevezést a hetvenes évek elején *Benoit Mandelbrot* adta, akinek *The Fractal Geometry of Nature (A Természet Fraktálgeometriája)* című munkája mutatta be, és magyarázta el először a fogalmakat, melyek alapjául szolgálnak ennek az új területnek. Habár előtte már többen is (mint például: *Cantor, Hausdorff, Julia, Koch, Peano, Poincaré, Richardson, Sierpinski, Weierstrass*) nyertek szűk bepillantást a fraktálok rejtelmeibe egy-egy alakzat vagy rendhagyó tulajdonság vizsgálatával, ezeknek az eredményeknek nem szenteltek figyelmet. *Mandelbrot* elméje volt, amely összekovácsolta őket egyetlen koherens rendszerbe.

Maga a név a latin **fractus** szóból származik, melynek jelentése **törött**, **töredezett**. Tulajdonképpen fraktálnak nevezhető minden olyan motívum, melyről a részleteket felnagyítva összetettebb, bonyolultabb képet kapunk, amely sok esetben az eredeti alakzatra emlékeztet; és nagyítása a végtelenségig folytatható. "Minél közelebbről nézzük, annál több részletet látunk." Ebben térnek el igazán az euklideszi formáktól, hiszen azok közelebbről nézve egyre egyszerűbbnek, simábbnak tűnnek, görbületeik egyenes vonalnak vagy síknak látszódnak. Az analízis nyelvére fordítva: simák. Mondhatjuk azt is, hogy az euklideszi inkább az ember alkotta, a fraktál geometria pedig a természet alkotta objektumok leírására szolgál. Nemsokára erre is láthatunk példát.

Tehát a fraktálok különös, igen komplikált összetételű alakzatok, ponthalmazok, mindezek ellenére gyakran kifejezetten kevés adatból generálhatók, kinyerhetők, ahogy ezt az elkövetkezendőkben látni is fogjuk. Mint ahogy az előbb szó volt róla, a végtelenségig felnagyíthatók és sok esetben önhasonlóságot mutatnak. Ami szintén rendhagyó tulajdonság - de csak érintőlegesen fogunk foglalkozni vele - hogy dimenziójuk általában tört, nem egész és általában az úgynevezett topológiai és mértékelméleti dimenziójuk eltér. [7]

3.2. Egyszerűen generálható példák

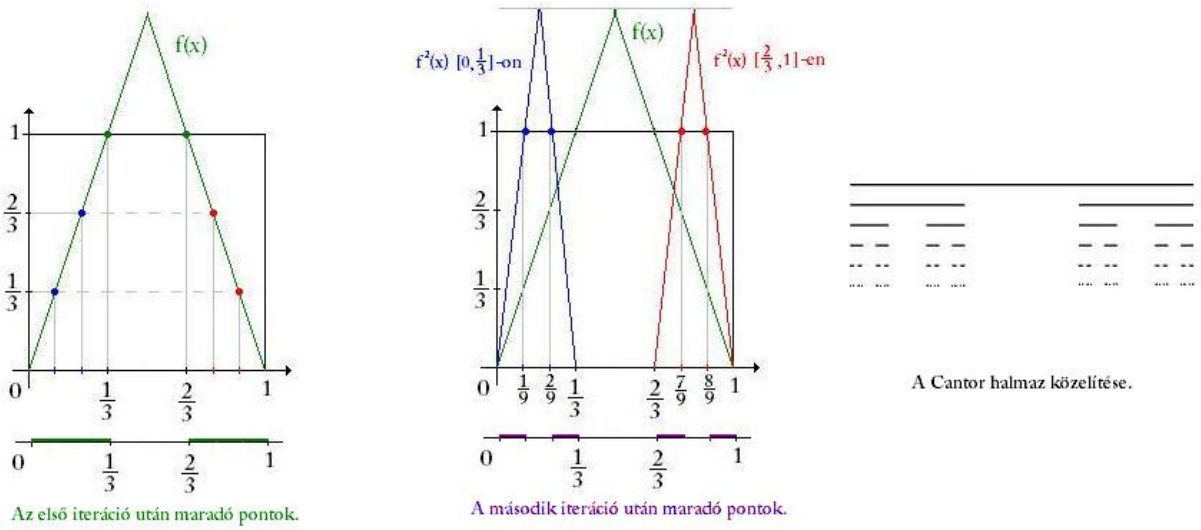
A körülírás után, most figyeljünk meg néhányat a legismertebb, egyszerűen generálható fraktálok közül, majd egy-két a természet alkotta különleges képet. A következő példák nagy részét már jóval a fraktálok fogalmának megalkotása előtt felfedezték és vizsgálták. A **Cantor halmazt** 1883-ban *Georg Cantor*, német matematikus mutatta be, bár felfedezni már 1874-ben felfedezte *Henry John Stephen Smith*. A **Koch-görbét**, más néven **Koch-hópelyhet** 1904-ben *Helge von Koch* írta le, a **Sierpinski háromszöget** pedig *Waclaw Sierpinski* 1915-ben. Hasonlóképpen a **Menger szivacs** is - amely a Cantor halmaz hármondimenziós kiterjesztése - már 1926-ban foglalkoztatta *Karl Mengert* a topologikus dimenzió fogalmának kutatása során, valamint a **Heighway sárkányról** is, később ugyan, de még mindig az összefogó fraktál fogalom létezése előtt, 1967-ben írt *Martin Gardner* egy tudományos cikkében.

3.2.1. Cantor halmaz

A Cantor halmaz definiálásához először tekintsük a következő függvényt a $[0,1]$ intervallumon.

$$f(x) = \begin{cases} 3x & \text{ha} \quad 0 \le x \le \frac{1}{2} \\ -3(x-\frac{1}{2})+\frac{3}{2} & \text{ha} \quad \frac{1}{2} \le x \le 1. \end{cases}$$

1. ábra. A Cantor halmaz és a sátor leképezés.



Láthatjuk, hogy az $x \in (\frac{1}{3}, \frac{2}{3})$ pontok képei $[0,1]$-en kívülre esnek, más szó-

val az $(x, f(x))$ pontok kilépnek a $[0,1] \times [0,1]$ négyzetből, míg a $[0, \frac{1}{3}]$ és az $[\frac{2}{3}, 1]$ intervallumok pontjai egyaránt $[0,1]$-be képződnek. Alkalmazzuk a bent maradt pontokra f-et még egyszer. Megállapíthatjuk, hogy f^2-et $[0, \frac{1}{3}] \cup [\frac{2}{3}, 1]$-re alkalmazva $(\frac{1}{9}, \frac{2}{9})$ és $(\frac{7}{9}, \frac{8}{9})$ pontjai kilépők lesznek, a $[0, \frac{1}{9}], [\frac{2}{9}, \frac{3}{9}], [\frac{6}{9}, \frac{7}{9}], [\frac{8}{9}, 1]$ intervallumok pedig mind $[0,1]$-re képződnek. Ezt látva joggal merül fel a kérdés, hogy léteznek-e olyan pontok, melyek "örökre", f tetszőlegesen sokszori alkalmazása után is bent maradnak $[0,1]$-ben, és ha igen, melyek ezek.
Vegyük észre, hogy minden iterációs lépés után, az addig bentmaradt pontok halmaza zárt, egymást nem metsző intervallumokból áll, és a következő lépésben minden ilyen intervallum középső harmada kilépő lesz, azaz

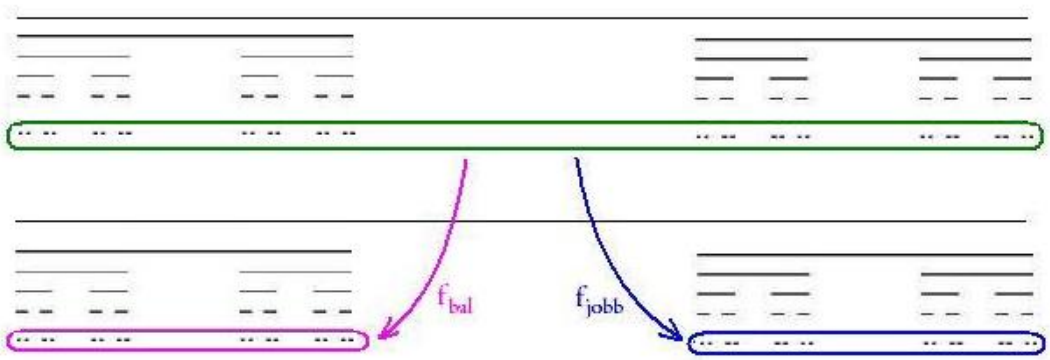
$$C_n = \frac{C_{n-1}}{3} \cup \frac{2}{3} + \frac{C_{n-1}}{3}.$$

Így az "örökre maradó" pontok halmazát felírhatjuk a fenti sorozat határértékének formájában:

$$C_0 = [0,1]\,, \quad C_n = \frac{C_{n-1}}{3} \cup \frac{2}{3} + \frac{C_{n-1}}{3}, \quad C = \lim_{n\to\infty} C_n.$$

A fenti C halmazt nevezzük **Cantor halmaz**nak, melynek egyik igen érdekes tulajdonsága, hogy nem tartalmaz itervallumot, azaz teljesen széteső.
Emellett önhasonlóságot is megfigyelhetünk a halmazon, hiszen látszik, a megmaradó halmaz bal, illetve jobboldali részhalmazán, hogy az eredetihez hasonlóan viselkednek f iterált alkalmazása esetén, és az $f_{jobb} = \frac{x}{3}$, $f_{bal} = \frac{x}{3} + \frac{2}{3}$ hasonlóságokkal megkaphatók. Sőt, az $\{f_{jobb}, f_{bal}\}$ iterált függvényrendszer alkalmas a Cantor halmaz, mint fraktál leírására.

2. ábra. A Cantor halmaz hasonlóságai.



Így könnyen látható, hogy az első szétvágás után megmaradt bal és jobb oldali részhalmazok (C_{bal}, C_{jobb}) az eredeti halmaz $\frac{1}{3}$-aránnyal kicsinyített képei. Ekkor

$$H^s(C) \quad = \quad H^s(C_{bal}) + H^s(C_{jobb}) \quad = \quad \left(\frac{1}{3}\right)^s H^s(C) + \left(\frac{1}{3}\right)^s H^s(C),$$

ahol $s = \dim_H C$, azaz s a Cantor halmaz Hausdorff-dimenziója. Feltéve, hogy $0 < H^s(C) < \infty$, ebből $H^s(C)$-vel egyszerűsítve kaphatjuk, hogy

$$1 = 2 \cdot \left(\frac{1}{3}\right)^s, \text{ vagyis } s = \frac{\log 2}{\log 3}.$$

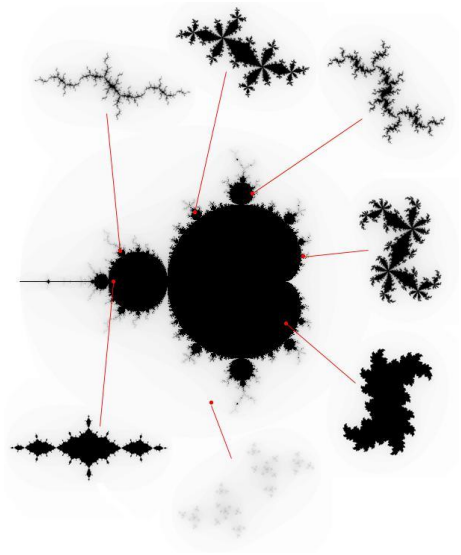
Tehát a Cantor halmaz Hausdorff-dimenziója $\approx 0,631$. Érdekes, hogy egy egység hosszú szakaszból indulva, kevesebb mint egydimenziós halmazt kapunk. Viszont belegondolva, hogy a halmaz teljesen széteső, azaz nem tartalmaz intervallumot, melynek lenne egydimenzióban mérhető a hossza, nem meglepő az eredmény. [8]

3.2.2. Mandelbrot és Julia halmazok

A **Julia halmazokat** egy francia matematikusról, *Gaston Julia*-ról nevezték el, aki vizsgálni kezdte tulajdonságaikat. Julia halmaz alatt azon $z \in \mathbb{C}$ komplex számok halmazát értjük, melyek esetén a $z_0 = z$, $z_{n+1} = F(z_n)$ sorozatok korlátosak, ahol $F(z) = \frac{P(z)}{Q(z)}$, közös osztóval nem rendelkező komplex polinomok hányadosa.
Valójában az $F(z) = z^2 + c$ függvény használata a legelterjedtebb, itt c egy komplex konstans, melynek különböző értékei esetén, más-más halmazokat kapunk eredményül. Általánosítva a formulát, más függvényeket is választhatunk az iterációra, érdemes nem-lineáris függvényekkel is próbát tenni.
A Julia halmazok számítógépes ábrázolása is viszonylag egyszerű. Minden pixelt megfeleltetve a komplex számsík egy pontjának, beállítjuk azt z_0 kezdőértéknek, és ha a rá illeszkedő (z_n) sorozat divergens, akkor a pixel fehér, ha nem, akkor fekete színt kap. Ahhoz viszont, hogy a sorozat konvergenciájáról dönteni tudjunk, általában sok iterációra van szükség.

A **Mandelbrot halmaz** nevét *Benoit Mandelbrot*-ról kapta. Igen közel áll a Julia halmazokhoz, generálása szintén a sorozathoz kötődik. A halmaz pontjait azok a c komplex számok alkotják, melyek esetén a $z_0 = 0$, $z_{n+1} = z_n^2 + c$ sorozat korlátos.
Érdekes összefüggés közöttük, hogy a Mandelbrot halmaz a Julia halmazok egyfajta "tárgymutatója", hiszen minden Mandelbrot halmazon belülről vett c érték esetén, a c-hez tartozó Julia halmaz összefügő lesz, és minden külső c értékre pedig teljesen széteső.[9] [10]

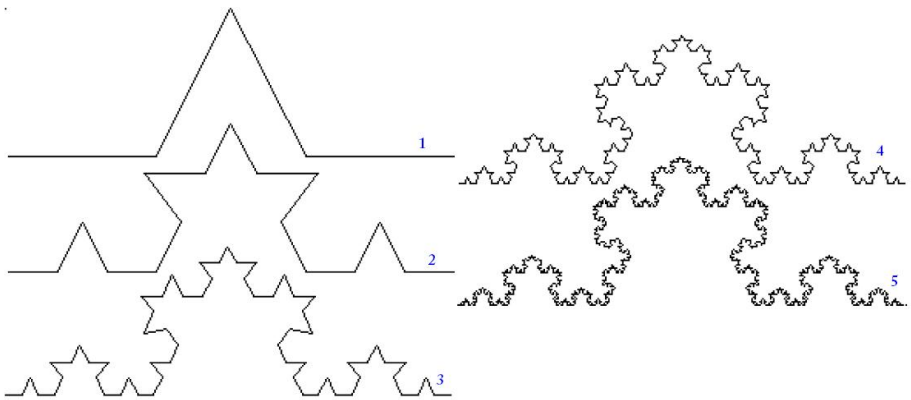
3. ábra. Különböző Julia halmazok c értékei a Mandelbrot halmazhoz képest.[10]



3.2.3. Koch görbe

A **Koch-görbe** az egyik legelső leírt fraktál. Generálása igen egyszerű. Egység hosszú szakaszból indulva, az első lépésben felosztjuk a szakaszt három egyenlő részre, majd a középsőt helyettesítjük két darab harmadhosszúságú szakasszal, mint egyenlő oldalú háromszög száraival. Ezt a kiegészítést tesszük meg minden iterációs lépésben, minden egyenes szakaszon.

4. ábra. Az Koch görbe generálása.



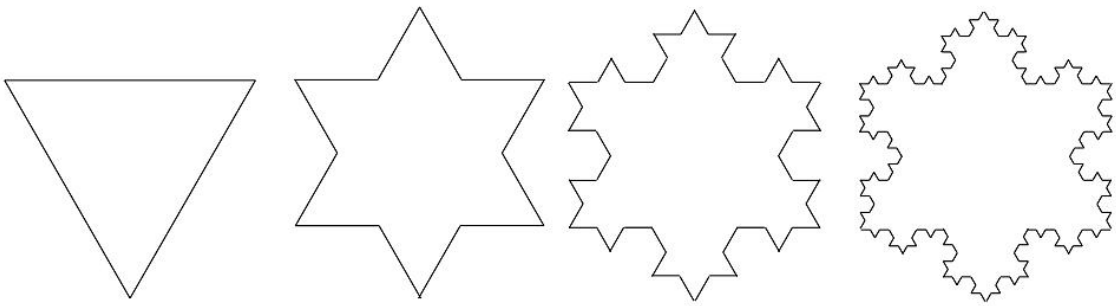
A görbe hosszának dimenziója már ránézésre is könnyen felírható, hiszen

az egész görbe négy darab $\frac{1}{3}$ aránnyal kicsinyített görbére osztható, azaz

$$H^s(K) \quad = \quad 4 \cdot \left(\frac{1}{3}\right)^s H^s(K), \text{ így } s = \frac{\log 4}{\log 3},$$

feltéve hogy $0 < H^s(K) < \infty$. Eszerint a Koch görbe Hausdorff-dimenziója $\approx 1,262$.
Érdekessége továbbá, hogy a görbe végtelen hosszúságú, hiszen az n-edik lépés után a hossza $\left(\frac{4}{3}\right)^n$ és $\lim_{n\to\infty}\left(\frac{4}{3}\right)^n = \infty$. Ez azt jelenti, hogy a görbe véges térrészen, önmagát nem metszve tetszőlegesen hosszú lesz, ha az iterálást elég sokáig folytatjuk.
Ha pedig egy szakasz helyett egy szabályos háromszög oldalaiból, három szakaszból indulunk, akkor kaphatjuk meg az úgynevezett **Koch-hópelyhet**. [11]

5. ábra. Az első néhány lépés.



3.2.4. Sierpinski háromszög és szőnyeg

A Sierpinski háromszög, mint azt már említettem, *Waclaw Sierpinski*, lengyel matematikus nevéhez fűződik. Az előző példához hasonlóan ez is önhasonló halmaz, így generálása is egyszerű. Egy szabályos háromszöglapból indulunk, melyet felosztunk négy darab, fele oldalhosszúságú kisebb háromszögre, melyek közül a középsőt elhagyjuk. A következő lépésben az így kapott "lyukas háromszöget" kicsinyítjük felére, és azt illesztjük be a három kisháromszög pozíciójába. Ezt az átalakítást elég (végtelen) sokszor ismételve kapjuk a Sierpinski háromszöget.

A művelet leírása formálisabban, iterált függvényrendszerrel, a következő $f = \{f_1, f_2, f_3\}$ lesz, ahol a, a kiinduló háromszög oldalhossza, bal alsó csúcsa pedig az origó:

6. ábra. A Sierpinski háromszög első néhány iterációja.

- $f_1(x, y) = (\frac{x}{2}, \frac{y}{2})$,
- $f_2(x, y) = (\frac{x}{2}, \frac{y}{2}) + (\frac{a}{2}, 0)$
- $f_3(x, y) = (\frac{x}{2}, \frac{y}{2}) + (\frac{a}{4}, \frac{\sqrt{3}}{4}a)$.

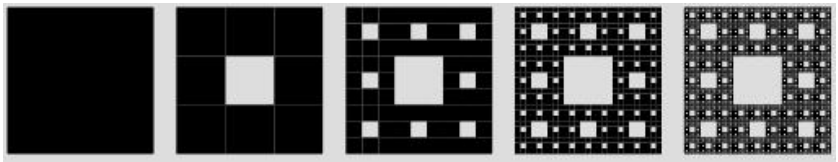
A Sierpinski háromszög önhasonlóságát kihasználva a Hausdorff-dimenziója is könnyen kiszámítható, hasonlóképpen, mint a Cantor-halmaz és a Koch görbe esetén:

$$H^s(SH) \quad = \quad 3 \cdot \left(\frac{1}{2}\right)^s H^s(SH), \quad \text{azaz } s = \frac{\log 3}{\log 2},$$

Ám a $0 < H^s(SH) < \infty$ feltételről sem szabad megfeledkeznünk. A Sierpinski háromszög Hausdorff-dimenziója tehát $\approx 1,585$. [12]

Sierpinski nevének kapcsán beszélhetünk még egy nevezetes fraktálról, a **Sierpinski szőnyegről**, melynek struktúrája igen hasonló. Ezesetben egy négyzetlapból indulunk ki, amit kilenc darab $\frac{1}{3}$ arányú kisnégyzetre osztunk fel, melyek közül a középsőt elhagyjuk. A következő lépésekben is mindig a kapott ábrát kicsinyítjük harmadára, majd illesztjük be a nyolc külső kisnégyzet pozíciójába. Az őt leíró függvényrendszer hasonló a háromszögéhez, $f = \{f_1, f_2, \ldots, f_8\}$, ahol $f_i(x, y) = \frac{1}{3}(x, y) + (c_i, d_i)$.

7. ábra. A Sierpinski szőnyeg első néhány iterációja.



Egy a oldalú, kiindulási négyzet esetén, melynek bal alsó csúcsa az origó, a következő (c_i, d_i) eltolásvektorok lesznek:

$$(0, \frac{2}{3}a),\ (\frac{1}{3}a, \frac{2}{3}a),\ (\frac{2}{3}a, \frac{2}{3}a),\ (0, \frac{1}{3}a),\ (\frac{2}{3}a, \frac{1}{3}a),\ (0, 0),\ (\frac{1}{3}a, 0),\ (\frac{2}{3}a, 0).$$

Dimenziójára pedig ugyanúgy felírhatjuk az egyenletet:

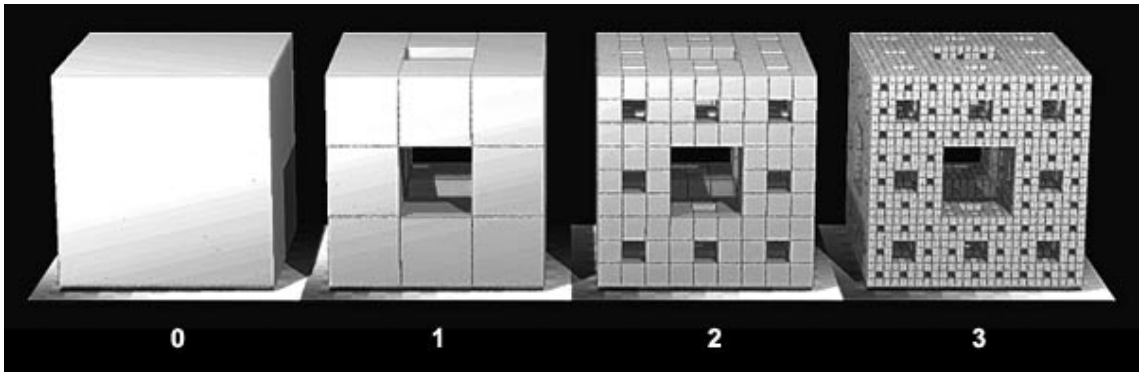
$$H^s(SSz) \quad = \quad 8 \cdot \left(\frac{1}{3}\right)^s H^s(SSz), \quad \text{melyből } s = \frac{\log 8}{\log 3},$$

és természetesen feltesszük, hogy $0 < H^s(SSz) < \infty$. Így a Sierpinski szőnyeg dimenziója $\approx 1,893$. [13]

3.2.5. Menger szivacs

A **Menger szivacs** a Sierpinski szőnyeghez, illetve a Cantor halmazhoz hasonló struktúrájú alakzat, mondhatjuk, hogy ezek háromdimenzióra való kiterjesztése. Első lépésként egységkocka éleit harmadolva, 27 egyforma kisebb kockára osztjuk fel, melyek közül elhagyjuk minden lap középső kockáját, illetve a legbelsőt, azaz összesen hetet. A második iterációban hasonlóképp járunk el a megmaradt 20 kiskockával, majd ezt a darabolási módszert folytatjuk a végtelenségig.

8. ábra. Az első néhány lépés.



Hausdorff-dimenzióját tekintve azt figyelhetjük meg, hogy az első felosztásban vett kiskockák élei $\frac{1}{3}$-szorosai az egységkockának. A megfelelő hét kiskocka elhagyása után húsz marad belőlük, így a következő egyenlőséget írhatjuk fel,

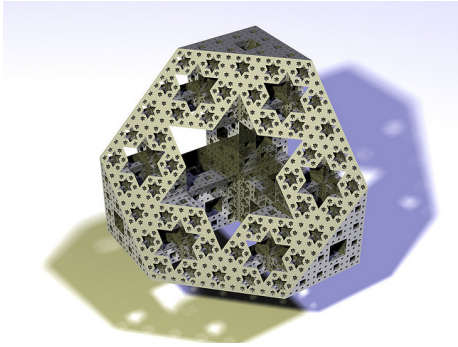
$$H^s(M) \quad = 20 \cdot \left(\frac{1}{3}\right)^s H^s(M), \text{ amelyből } s = \frac{\log 20}{\log 3},$$

feltéve, hogy $0 < H^s(M) < \infty$. Így Hausdorff-dimenziója $\approx 2,727$. [14]

Vegyük észre, hogy a szivacs minden lapja Sierpinski szőnyeg, illetve minden lapátlója, testátlója, valamint a lapok oldalfelező szimmetriatengelyei is

Cantor halmazok. Ez szemlélteti és megerősíti a gondolatot, miszerint a Menger szivacs ezek háromdimenzióra való kiterjesztése.
Ezeken kívül létezik még egy érdekes és látványos részhalmaza a szivacsnak, a "Menger szivacs meglepő szelete". [15]

9. ábra. A szelet (The Surprising Slice)[15].

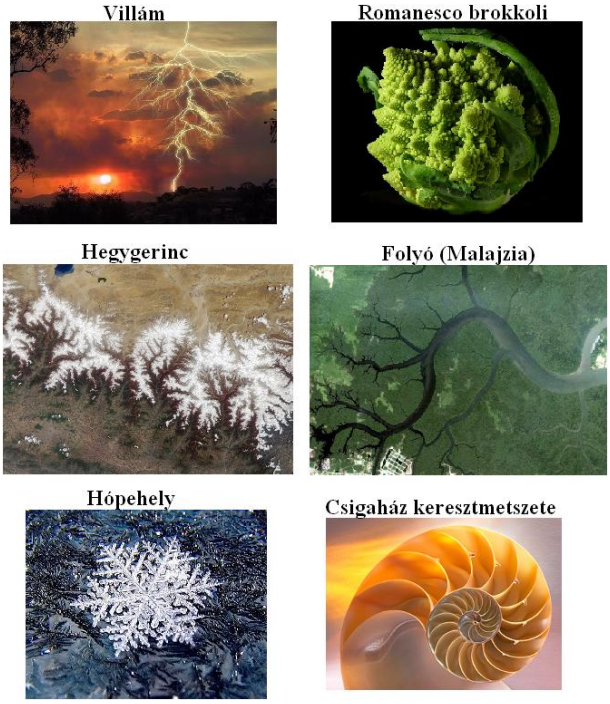


3.3. Fraktálok a természetben

A természetben található tárgyak geometriai leírása olyan régi, mint maga a tudomány. Ezen leíráshoz a mindenki által jól ismert euklideszi vonalakat, téglalapokat, kockákat, gömböket, stb. használják. De a természetben nemcsak euklideszi idomok vannak, sok fraktálszerű alakzatot találunk. A hetvenes évek elején jelentette ki *Benoit Mandelbrot*, hogy

"A felhők nem gömbök, a hegyek nem kúpok,
a partvonalak nem körívek,
a fakéreg nem sima,
és a villám sem terjed egyenes vonalban."

10. ábra.



A legtöbb természeti objektum olyan bonyolult alakú, hogy megérdemlik, hogy geometriailag kaotikusnak hívjuk őket. Lehetetlennek tűnt a matematikai leírásuk, ezért a "matematika szörnyetegeinek" nevezték őket. A fraktál fogalom, a számszerű leíráson kívül az ezekben az objektumokban rejlő szabályosság felismerésében is segít bennünket. Ezek az alakzatok ugyan nem tökéletesen pontosak és struktúrájuk nem tartalmaz 5-6 lépcsőnél többet, mégis megfigyelhető bennük egyfajta önhasonlóság. Az egyik klasszikus példa az elektomos kisülés, például a villámlás, ahol a hasonlóság a terjedésben nyilvánul meg. Növényekben is gyakran találhatunk fraktálszerű alakzatokat, pél-

dául a napraforgók, kövirózsák, karfiolok, fák levelei, vagy a páfránylevél. Továbbá a föld felszínén pedig a folyók és gleccserek által szabdalt síkságok és hegygerincek felülnézete. Az egészen aprók közül pedig igen érdekes a hópelyhek, különböző csigaházak, kristályok szerkezetét megfigyelni.[16]

Mindezek között a legizgalmasabb talán a Romanesco brokkoli spirálisan csavarodó formája. Ezek láttán jogos lehet a kérdés, hogyan képes a természet ilyen bonyolult formák megalkotására. A megoldás kulcsa a szimmetria, illetve az egyszerű generálhatóság, azaz egy bizonyos lépés sokszori ismétlődése. A legegyszerűbb a fák ágainak növekedésébe belegondolni: minden ág újabb ágat növeszt. A másik fontos szerepet játszó tényező pedig a véletlen. Gondolva itt a partvonalak egyenetlen eróziójára, lecsiszolódására folyók deltájának elágazásaira, melyeket az egyenetlen domborzat határoz meg, vagy egy sziklafalon lezúduló vízesés, melyet a kiálló kövek megszabdalnak. Nézzünk példát ez utóbbi modellezésére.

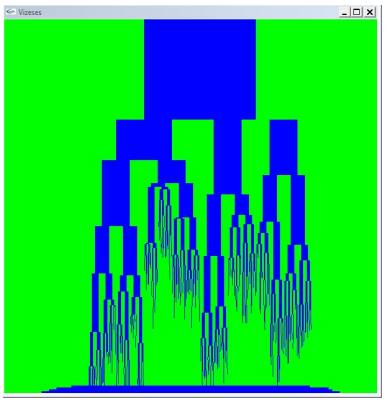
3.3.1. Vízesés

11. ábra. Sziklák által szabdalt zuhatag.

Próbáljuk meg a véletlen szerepét olyan korlátok közé szorítani, melyek segítségével hasonló ábrákat kaphatunk.
Induljunk ki egy d *szélességű* vízoszlopból, amely h *magasságból* esik. Az egyszerűbb kivitelezés érdekében feltételezzük, hogy minden kiálló szikla a vele ütköző vizsugarat egyenlő arányban ketté osztja, így d értékét a fa minden szintjén felezzük. Ebben az esetben egyszerűen egy fáról van szó, melynek ágai különböző hosszúságúak, csak a csúcsok, azaz osztópontok helyeit kell meghatároznunk. Ehhez pedig szükség van egy $v \in [0, 1]$ véletlen számra minden elágazás esetén, illetve h értékét figyelembe véve egy l változóra is, ami az azonos szinten lévő vízsugarak *alaphossza*. A $v \cdot l$ szorzatok adják meg a töretlen vízsugarak hosszát. Alkalmazunk még ezek mellett egy sz szorzót, amely a fa következő szintjére lépéskor növeli vagy csökkenti l-t, sz értékétől

függően.
Tehát a fal tetejéről induló vízoszlopot véletlenszerűen választott hosszúság után ketté osztjuk, majd ugyanezt tesszük az így keletkezett újabb vízsugarakkal is. Ennek a módszernek egy kezdetleges megvalósítása a mellékelt waterfall program.

12. ábra. A program által készített ábra.



4. Kik használják és hogyan?

4.1. Számítógépes grafika

A számítógépes grafika egyike volt a fraktálok legelső alkalmazásainak. Valóban, a fraktálok segítségével valósághű képek készíthetők, melyek ráadásul igen kevés tárhelyet igényelnek, hiszen könnyen generálhatók és hatékonyan tömöríthetők. Már *Mandelbrot* fent említett könyvében is megjelentek fraktál tájképek.
Habár az első algoritmusok és az ötletek magához a "fraktálok felfedezőjéhez" tartoztak, alkalmazásukat a művészet mezején *Richard Voss* kezdte, aki Mandelbrot könyvének tájképeit megalkotta. Ez indította meg számos művész és producer fantáziáját a science fiction filmek világában. Nem sokkal később *Loren Carpenter* készített egy animációs filmet egy repülőútról egy "fraktál táj" felett, melyre azonnal felfigyeltek a szakmában.
Az újonan felfedezett technikát használták a *Star Trek II: The Wrath of Khan* c. filmben a *Genesis* bolygó, és a *Return of the Jedi* c. filmben a holdak domborzatának generálásához is. Ezen speciális effektek sikere igen népszerűvé tette a módszert. Mára már számos szoftver létezik, melyek segítségével bárki, aki csak egy keveset ért a fraktálokhoz és a számítógépes grafikához, létrehozhat efféle alkotást. [17]

4.1.1. Tájképek

Az általunk tárgyalt fraktál tájkép olyan felület, melyet egy sztochasztikus algoritmus generál, amit arra terveztek, hogy eredménye fraktálszerű viselkedést mutasson, ezzel természetes domborzat látszatát keltve. Vagyis a művelet végén nem egy előre meghatározott fraktál-felszínt kapunk, sokkal inkább egy véletlenszerű, fraktál tulajdonságokkal rendelkező alakzatot.

A természetben nagyon sok jelenség mutat statisztikus önhasonlóságot, ezért lehet igazán természetes hatást kelteni "majdnem önhasonló" motívumok használatával.

A valóban természetesnek látszó felszínek és tájképek generálása fordulópontot hozott a művészet történetébe, hiszen a határ, a megkülönböztethetőség a számítógépek által létrehozott és a természetes, ember alkotta tájak között ezzel elmosódott. [18]

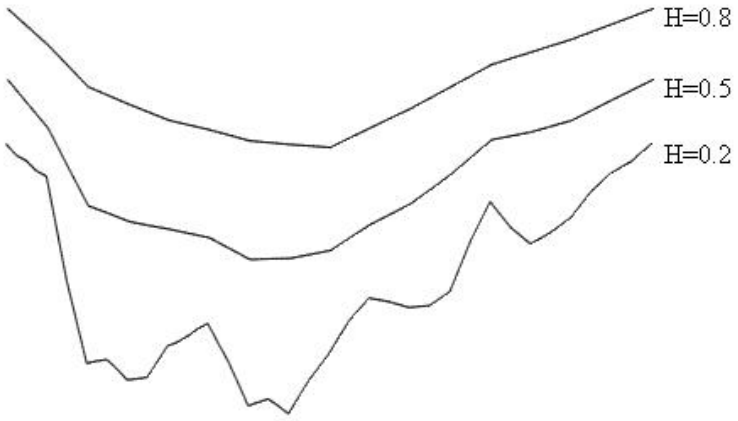
4.1.2. Véletlenszerű középpontáthelyezés algoritmus

Mielőtt felszínekkel foglalkoznánk, nézzük meg először a középpontáthelyezést egydimenzióban. A módszer ilyen formában kiválóan alkalmas egy távol, a horizonton álló hegygerinc törött vonalainak megrajzolására. A művelet lépései tehát:

- Induljunk ki egyetlen vízszintes egyenes szakaszból.
- Ismételjük az alábbi lépéseket (elég sokszor):
  - A töröttvonal minden szakaszán keressük meg a szakasz középpontját és mozdítsuk el az Y-tengelyen egy-egy az I intervallumból véletlenszerűen vett értékkel. Kössük össze az áthelyezett középpontot a szakasz végpontjaival.
  - Szűkítsük a véletlen számok I intervallumát.

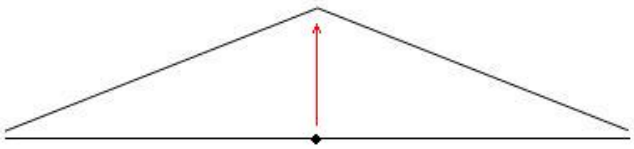
Az, hogy mennyivel kell rövidíteni a véletlen számok intervallumát, attól függ, hogy mennyire durva felszínt szeretnénk kapni. Minél jobban leszűkítjük, annál simább; minél kevesebbet változtatunk, annál egyenetlenebb lesz a végeredmény. Akár konstanst is rendelhetünk az egyenetlenséghez, a következő egyszerű technikával:
legyen $H \in [0, 1]$, ekkor $2^{-H} \in [\frac{1}{2}, 1]$. Ezzel szorozva az intervallumot minden ciklusban, láthatjuk, hogy a $H = 1$ esetben az intervallum feleződik, így az eredményünk sima lesz, illetve minél kisebb H értéket, esetleg nullát véve, az intervallum (alig) változik, s ezzel egészen durva felületet nyerünk.

13. ábra. A vonal simasága, töredezettsége H függvényében.[19]



Nézzünk egy példát, $H = 1$-es állandóval. Elindulunk egyenes vonallal $[-1, 1]$-en az X-tengelyen, Y értéke mindkét végpontban nulla. Kezdetben a

14. ábra. "A ciklus első iterációja és a kezdeti szakasz."[19]



véletlen számok intervallumát $I = [-1, 1]$-nek választjuk (ez tetszőleges), majd $\delta \in I$ -vel elmozdítjuk a középpontot függőleges irányban.

Ezután kettő, fele hosszúságú szakasz áll rendelkezésünkre, és a beszorzás után $I = [-\frac{1}{2}, \frac{1}{2}]$. Mindkét középponthoz egy véletlen számot rendelve és az eltolást elvégezve a következőt kapjuk.

15. ábra. "A ciklus második iterációja."[19]

Újra szűkítünk az intervallumon, ekkor $I = [-\frac{1}{4}, \frac{1}{4}]$, az áthelyezések után a kép:

16. ábra. "A ciklus harmadik iterációja."[19]

A műveletet megfelelően sokszor elvégezve, a kívánt eredményt egy rajzolóprogramba beolvashatjuk, és egy kevés színezéssel máris valami különlegeset kapunk.

Ez a felismerés, hogy néhány egyszerű lépés segítségével ilyen komplex képek készíthetők, vezetett ahhoz, hogy elkezdjenek érdeklődni, foglalkozni a fraktál képtömörítéssel, a képfeldolgozás egy még meghódításra váró, új területével. [19]

17. ábra. "A színezett végeredmény."[19]

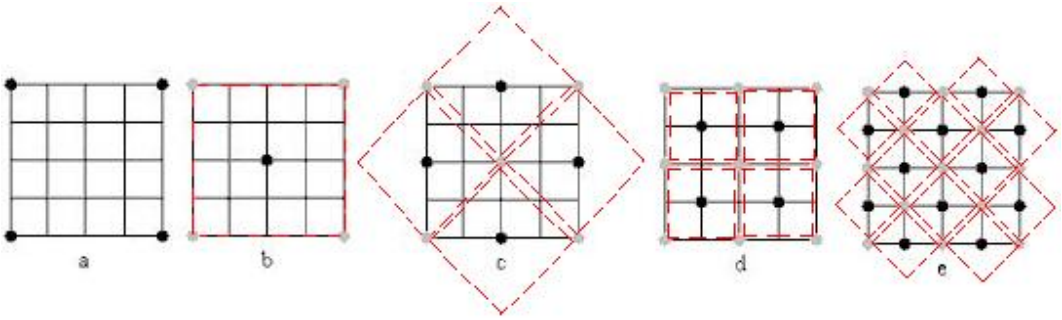
4.1.3. Gyémánt-négyzet (Diamond-Square) algoritmus

A **gyémánt-négyzet algoritmus** használatával hegycsúcsok felszínét hozhatjuk létre, a kiinduló sík pontjainak emelésével. Ez a módszer, melyet eredetileg *Alain Fournier, Don Fussel* és *Loren Carpenter* írt le, tulajdonképpen az előző részben tárgyaltak kétdimenzióban alkalmazható változata.

Íme az ötlet: kezdetnek tekintsünk egy kétdimenziós tömböt, melynek elemei pontok. Mekkora méretűre van szükség? Az egyszerűség kedvéért vegyünk egy négyzetet, oldalhossza pedig legyen 2^n+1. Helyezzük a négy csúcsnak megfelelő pontokat azonos magasságra, így négyzetet kapunk (*18/a* ábra).

Az egyszerűség és a jól láthatóság kedvéért nézzünk példának egy 5×5-ös tömböt. A *18/a* ábrán a négy sarokpont feketével kiemelt. Ez a kiindulási helyzet az iterált felosztási módszerhez, mely a következő két lépés ismétléséből áll:

18. ábra. "...nézzünk példának egy 5×5-ös tömböt..."[19]



- A **gyémánt-lépés**: Vegyünk, egy az előző lépésben kijelölt pontok kö-

zül négyet, melyek meghatároznak egy négyzetet, ahogy a *18/b* és *18/d* ábrákon látható.
Középpontjának magassága legyen a csúcsok magasságának átlaga és egy véletlenszerű hiba összege:

$$\frac{A+B+C+D}{4} + r,\ r \in I.$$

Ezt az összes négyzet középpontján végezzük el. A középpontok által határolt rombuszokat (a francia kártyákról ismerős károkat) kapunk. Ezeket nevezik angolul *diamond*-oknak, azaz gyémántoknak.

- A **négyzet-lépés**: Vegyünk egyet az előző lépésben létrejött rombuszok közül. Azokat a kilógórombuszokat is figyelembe vesszük, melyeknek csak három csúcsát tartalmazza a négyzetrács, ahogy a *18/c* és a *18/e* ábra mutatja. Láthatjuk, hogy az első négyzet lépés kezdetekor még csak ilyen csonka rombuszaink vannak.
A középpontjához az előzőhöz hasonlóan számolt értéket rendeljünk:

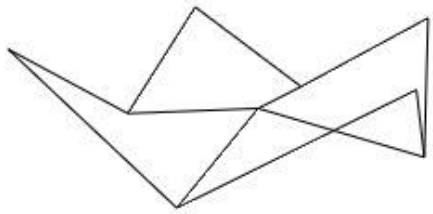
$$\frac{A+B+C+D}{4} + r,\ r \in I.$$

Ezt minden gyémánttal elvégezve, újra négyzeteket kapunk.

- Mielőtt újra megtennénk a gyémánt-lépést, csökkentsük a véletlen számok I intervallumát.

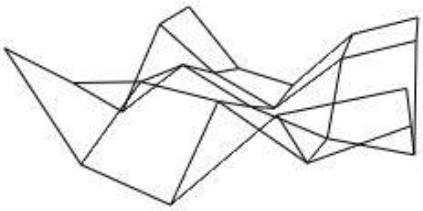
Látható, hogy minden elvégzett gyémánt-négyzet lépéspár után a négyzeteink száma megkétszereződik, így az algoritmus igen gyors.
Az előbbi 5×5-ös példán, ha kezdetben $I = [-1, 1]$, elvégezzük az első lépéspárt, majd a *18/c* ábrán megjelölt kilenc pontot összekötjük, a kapott felszín valahogy így néz ki:

19. ábra. "A *18/c* ábrán megjelölt 9 pontot összekötjük."[19]



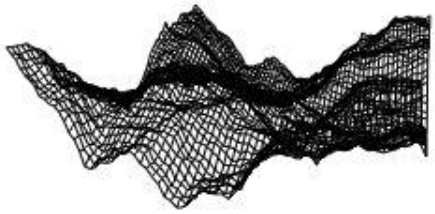
A következő lépések előtt szűkítsük az intervallumot $I = [-\frac{1}{2}, \frac{1}{2}]$-re. Azaz ekkor $H = 1$-es töredezettségállandóval dolgozunk.
A gyémánt lépéssel meghatározva a *18/d* ábrán kiemelt négy, a négyzet lépéssel pedig az *18/e* ábra tizenkét pontjának magasságát, huszonöt elhelyezett pontunk lesz. A megfelelőket összekötve, az eredmény:

20. ábra. "Az *18/e* ábrán megjelölt 25 pontot összekötjük."[19]



Természetesen nagyobb tömbből kiindulva, és elég lépéspárt elvégezve, finomabb, jobban töredezett felületet kapunk. Ez már egy jóval mutatósabb ábra lesz, mely igazán valódinak tűnő hegy képét imitálja. [19]

21. ábra. "Nagyobb tömb és magasabb lépésszám eredménye."[19]



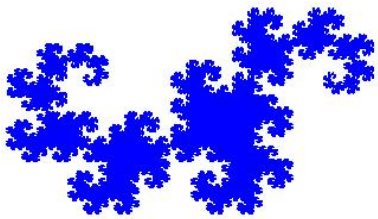
4.2. Képtömörítés

4.2.1. Bevezetés

A **Fraktál képtömörítés** (Fractal Image Compression) egy fraktálokon alapuló eljárás digitális képeken. A módszer leginkább a természet, illetve anyagok képeire alkalmas, abból kiindulva hogy a kép egyes részletei gyakran hasonlítanak ugyanazon kép más részeire. Az algoritmus ezeket a részleteket matematikai leírással úgy nevezett *fraktál kóddá* alakítja át, melyek az ábra elkódolását adják.
A kép fraktál reprezentációja matematikailag, egy IFS-sel írható le. Az egyszerűség kedvéért tekintsünk egy csak két színből álló képet, így az megfeleltethető egy $A \subseteq \mathbb{R}^2$ halmaznak. Az ötlet az, hogy egy megfelelő IFS-t konstruálva, eleget tegyünk a $H(A) = A$ egyenlőségnek, tehát azt a függvényrendszert kell megtalálnunk, melynek attraktora az A halmaz, vagyis a kódolandó képünk lesz.
Michael Barnsley állt a fraktál tömörítés fejlesztésének élén, az 1980-as évek végén. Mindegyik módszer az iterált függvényrendszereken alapul, és legismertebb tömörítő algoritmus megalkotása *Barnsley* és *Alan Sloan* nevéhez fűződik. 1987-ben *Barnsley* és *Sloan* megalapították az *Iterated Systems Inc.* nevű szoftverfejlesztő céget.
Az első automatikus szoftvert *Barnsley* egy tanítványa, *Arnaud Jacquin* valósította meg 1992-ben, és ez jelentette a legnagyobb áttörést a cég számára, hiszen így már nem volt szükség emberi beavatkozásra. [20]

4.2.2. Sárkány elkódolása (IFS)

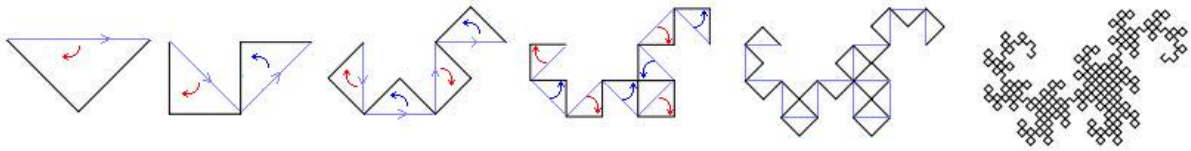
22. ábra. "A Heighway sárkány."



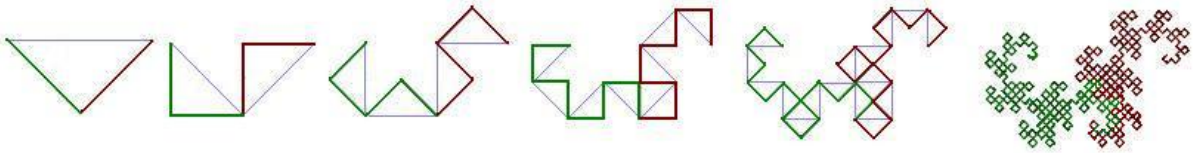
Sárkány görbék olyan önhasonló fraktál görbék, melyek rekurzívan, transzformációkkal közelíthetők. A legismertebb közülük, az úgynevezett *Heighway* (vagy *Jurassic Park*) *sárkány*. Generálása igen egyszerű, és több irányból megközelíthető. Tekinthetünk rá *jobb- és balkanyarok sorozata*ként, *Lindenmayer-rendszer*ként és *IFS*-ként is.
Vegyük ezek közül az iterált függvényrendszerrel való leírást. Az alábbi ábrán jól látható, hogy egy iterációs lépés során az eredeti alakzatot $\frac{1}{\sqrt{2}}$-szeresére ki-

csinyítjük, majd egyszer 45°-kal elforgatva, illetve egyszer 135°-kal elforgatva és egy egységgel (a kiinduló szakasz hossza) eltolva ábrázoljuk.

23. ábra. "Például: minden szakaszt helyettesítünk két új derékszöget bezáró szakasszal váltakozva a jobb és bal oldalon."



24. ábra. "IFS-sel generálva."



Ezeket a transzformációkat az $\mathbb{R}^n$ sík pontjaira felírva, a következő két leképzést kapjuk. [21]

$$f_1(x,y) = \frac{1}{\sqrt{2}} \begin{pmatrix} \cos 45^\circ & -\sin 45^\circ \\ \sin 45^\circ & \cos 45^\circ \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_2(x,y) = \frac{1}{\sqrt{2}} \begin{pmatrix} \cos 135^\circ & -\sin 135^\circ \\ \sin 135^\circ & \cos 135^\circ \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

4.2.3. Elkódolás és visszafejtés

Amint az a bevezetésben is olvasható, egy kép elkódolása a megfelelő, a képet leíró IFS megtalálását jelenti. A rendszer függvényei mind hasonlósági tranfszormációk, melyek a kép egy-egy részlete közötti hasonlóságot írják le.

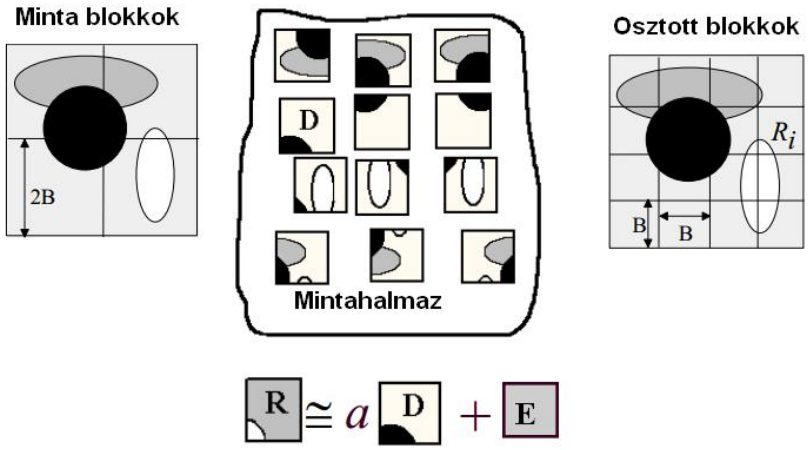
Ezek megtalálásához különböző felosztási módszerek alkalmazhatók, melyekről a fejezet végén lesz néhány szó. Az eljárás bemutatásához pedig egyszerű, négyzet-blokkos felosztást fogunk alkalmazni. A kép felosztásával létrejött blokkokat (*range blocks, R*) a továbbiakban **osztott blokkoknak**, a

kiválasztott, transzformálandó képrészleteket (*domain blocks, D*) **minta blokkoknak**, a minta blokkok összes lehetséges transzformációjából származó képek halmazát pedig (*domain pool*, $\{f_j(D)\}$) **mintahalmaznak** fogjuk nevezni. Az osztott blokk feleakkora oldalhosszú, azaz negyedakkora területű, mint a hozzá tartozó mintablokk. Természetesen az a cél, hogy a teljes kép minél kevesebb mintablokk segítségével előállítható legyen.
Ezeken felül a színintenzitásra is alkalmazhatók leképezések. A probléma egyszerűsítése érdekében fekete-fehér képekkel foglalkozunk, hiszen a kép színezésének megoldása igen bonyolult lenne.

Az elkódolás szerint, minden R_i osztott blokkhoz létezik egy $f_j(D)$ mintahalmaz elem, egy a intenzitás leképzés és E hiba, vagy zaj, melyekre

$$R_i \approx a\big(f_j(D)\big) + E.$$

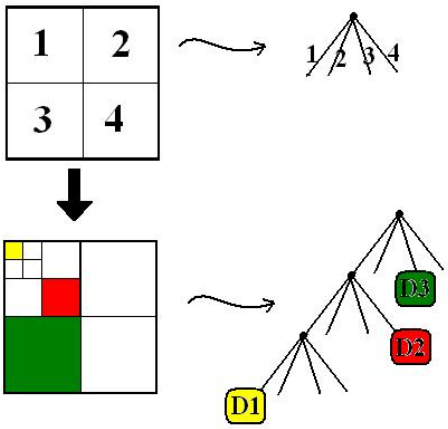
25. ábra. "Az elkódolás eszközei."[6]



A legegyszerűbb blokkfelosztás esetén a keresést az egész kép négy darab, $\frac{1}{2}$ arányú blokkra való osztásával kezdjük, és a kapott blokkok mindegyikéhez megpróbálunk egy minta blokkot találni az eredeti képen. A választott minta blokk oldalhossza kétszerese az osztott blokknak, és a lehetséges transzformációk valamelyikével, egy kis hibát megengedve hasonlóvá tudjuk tenni az osztott blokkal. Az első lépésben ilyet természetesen csak akkor találunk, ha az eredeti képünk erősen önhasonló. Amennyiben egy osztott blokkhoz nem sikerült minta blokkot találni, azt tovább osztjuk, négy kisblokkra. A blokkok felosztását addig folytatjuk, amígy a kép minden egyes blokkjához nem találunk az eredeti képen egy minta blokkot. Nyilvánvaló, hogy hétköznapi képek esetén nagyon apró felosztásra van szükség és a minta blokkok megkeresése sem egyszerű, így arra nem térek ki.

A négyblokkos felosztás előnye továbbá, hogy a felosztás fában eltárolható, ezzel könnyen átláthatóvá, kezelhetővé válik. [6]

26. ábra. "A fa és a blokkfelosztás."



4.3. Saját program

A dekódolási algoritmus működésének bemutatására, írtam egy egyszerű C++ programot, mely egy kiválasztott fraktál generálásának első néhány lépését ábrázolja. A felhasználó a következő opciók közül választhat:

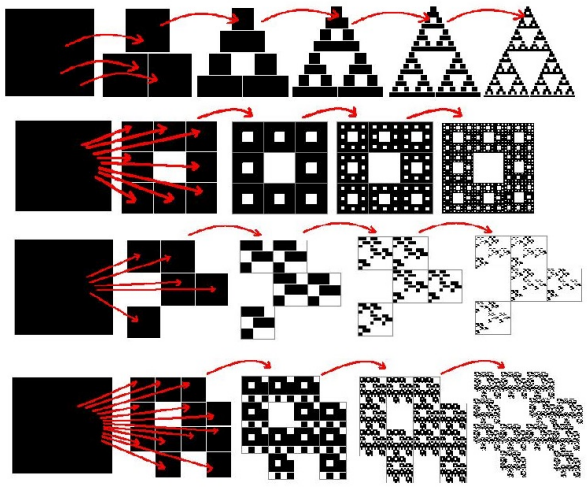
- Sierpinski háromszög képe $0, 1, \ldots$ vagy 9 lépés után,
- Sierpinski szőnyeg képe $0, 1, \ldots$ vagy 6 lépés után,
- Tetszőleges 3×3-as szőnyeg képe $0, 1, \ldots$ vagy 6 lépés után,
- Tetszőleges 4×4-es szőnyeg képe $0, 1, \ldots$ vagy 5 lépés után,
- A fenti négy ábra animációi, a megadott lépésszámig a lépésenkénti állapotról (negy külön menüpont),
- Véletlen szőnyeg játék,
- Véletlen Sierpinski háromszög.

A Sierpinski háromszög és szőnyeg kivételével minden ábra mellett egy kis ablakban az alapmintázat is megjelenik.
Ez a program erősen leegyszerűsített változata az előző fejezetben tárgyalt technikának. Egyetlen mintablokként az egész képet használja és azt kicsinyítve másolja a megfelelő, azonos méretű osztott blokkokba minden lépés során.

4.3.1. Az alapötlet

Az ötlet a Sierpinski szőnyeg generálásán alapul: egy négyzetet felosztunk kilenc egyforma kisebb négyzetre, melyek közül a középsőt kivágjuk. Ugyanezt a felosztást és kivágást alkalmazzuk az így megmaradt 8 kisebb négyzetre, azokat tekintve az egészként.
Ezeket a lépéseket tetszőlegesen sokszor ismételve kapjuk eredményül a szőnyeget. Vegyük észre azt is, hogy hasonló módon a Sierpinski háromszög is megkapható: szintén egy négyzetből indulunk ki, majd azt négy egyforma kisebb négyzetblokkra osztjuk. Ezek közül az alsó kettőbe a négyzet $\frac{1}{2}$ arányban kicsinyített képét másoljuk, majd azok tetejére, középre helyezünk egy harmadik másolatot is. Az így kapott képet is felére kicsinyítve, a megfelelő három pozícióba helyezzük és így tovább. Néhány lépés után tisztán látható hogy ezzel a módszerrel Sierpinski háromszöget kapunk. így háromszögek helyett egyszerűbb négyzetblokkokkal dolgozhatunk, hiszen minden lépés után finomodik az ábra felbontása.
Hogy még többféle izgalmas alakzatot kaphassunk eredményül, lehetőség van olyan 3×3-as vagy 4×4-es felosztású ábrát csinálni, ahol mi adjuk meg, hogy melyik blokkokba kerüljön kicsinyített ábra, és melyek maradjanak üresen.

27. ábra. "Egy-egy példa négy ábratípus elkészítésére."



4.3.2. A program működésének elmélete

Ezzel a módszerrel elkészítve az ábrákat könnyű dolgunk van, hiszen csak négyzetekkel kell foglalkoznunk, és minden négyzetet egyszerűen a bal alsó koordinátája segítségével tarthatunk számon. Emellett kézenfekvőnek tűnhet az

adatokat egy fában eltárolni, ahol minden négyzetblokknak, mint csúcsnak, a benne lévő kisebb blokkok a gyerekei. Természetesen csak azokkal a blokkokkal foglalkozunk, amelyek nem maradtak üresen. Ebben az esetben arra is ügyelni kell, hogy minden koordinátájának sajátmaga is leszármazottja legyen, különben leszármazottjai egy részét elveszítjük. Gondoljuk csak meg, hiszen fa formában tárolás esetén:

- 0. szint: a gyökér a kiinduló teljes négyzet lesz
- 1. szint: gyermekei a fél oldalhosszúságú kitöltött négyzetek lesznek
- 2. szint: ezen a szinten a negyed oldalhosszú blokkok szerepelnek
- ...

Amennyiben a bal alsó blokk kitöltendő, úgy a kezdő koordináta először egész oldalhosszú, majd fél, negyed, és így folytatva minden méretben kell szerepeljen. Így valóban szerepelnie kell a saját leszármazottjai között, hogy ne maradjon ki a rekurzió későbbi lépéseiből, és a végeredmény pontos legyen. A fa felépítése után a legalsó szinten lévő koordináták leolvasásával megkapjuk az ábra összes négyzetének bal alsó koordinátáját, a fa mélységéből pedig megtudjuk oldalhosszukat is, hiszen az az ábratípustól függően, egy szinttel lejjebb lépve felére, harmadára vagy negyedére csökken.

4.3.3. Méret és felbontás

A menü felvázolásában látható, hogy a különböző ábrák esetében különböző számú iterációs lépést tehetünk meg. Ennek magyarázatát a felbontás adja. A Sierpinski háromszög generálásakor az oldalhosszak felükre, 3×3-as és 4×4-es szőnyegek esetében harmadukra, illetve negyedükre csökkennek. A cél az, hogy a maximális lépésszám megtétele után a kép a lehető legnagyobb felbontású legyen, azaz egy pont, vagyis kis négyzet, egy pixelnek feleljen meg. Ezért a Sierpinski háromszög eleinte $(1 \cdot 2^10 = 1024)$ 1024×1024-es, a 3×3-as szőnyegek $(1 \cdot 3^6 = 729)$ 729×729-es, és a 4×4-es szőnyegek is $(1 \cdot 4^5 = 1024)$ 1024×1024-es ablakban jelentek meg.
Azonban az 1024-es méret nehány számítógép képernyőjébe - leginkább laptopokéba - nem fér bele, ezért méretüket felére csökkentettem, 512×512-esre. Ez viszont azt is jelenti, hogy a Sierpinski háromszög csak 9 lépést tehet meg, illetve a 4×4-es szőnyegek ábrái minimálisan torzulnak, hiszen 5 lépést megtéve "félpixelnyi" pontokat kéne kirajzolniuk. Mivel a kép méretének negyedére csökkentésével az ablak túl kicsi lenne, és a maximális lépésszámot is csökkenteni kéne eggyel, célszerűbb a 4×4-es szőnyegek enyhén torzult ábráinak megtartása, hiszen azt szabad szemmel szinte észre se lehet venni.

Az animációk esetében viszont más a helyzet, csökkentett méret mellett nem tud végigfutni a képsor a maximális lépésszámon, így a lépéseket kell csökkenteni. Négy lépést megtéve az 512×512-es képen, ($4^4 = 256$) a pontok, vagyis kiskockák mérete 2×2 pixeles lesz.

4.3.4. A megfelelő adatszerkezet

Az olyan fáknak, melyekben minden csúcsnak azonos számú gyermeke van, további előnye, hogy bármely csúcs gyerekeinek indexét egyszerű megállapítani. Egy k-gyermekű fa n-edik csúcsának gyermekei: $k \cdot n + i, \quad i = 1, 2, \ldots, k$. Ennek belátásához elég azt végiggondolni, hogy mire n gyermekeihez érünk, kiosztottunk már az őt megelőző n darab csúcs - mivel 0-val kezdtük az indexelést - egyenként k darab gyerekének, összesen $n \cdot k$ darab sorszámot.
Erre a tulajdonságra, a jól indexelhetőségre támaszkodva célszerű a *C++* egyik olyan adatszerkezetét választani a koordináták tárolására, amelynek elemeire index alapján is hivatkozhatunk. Ezért először *listával* próbálkoztam, mivel elemei indexelhetők, mégis rugalmasabb adatszerkezet mint a *tömb*, melynek méretét létrehozásakor meg kell határozni és ezen később nem változtathatunk. Viszont arról nem szabad megfeledkezni, hogy minden koordináta szerepel a saját gyermekei közt, ami azt jelenti hogy az első előfordulásától kezdődően minden további szinten megjelenik. Ez jelentősen növeli az adatszerkezet méretét, hiszen nekünk tulajdonképpen csak a leveleken szereplő koordinátákra lenne szükségünk, ezzel szemben az egész fát eltároljuk. Azaz ha egy d mélységű, k-gyermekű fáról van szó, akkor ahelyett, hogy csak a k^d darab levelet tárolnánk, mind a $\sum_{i=0}^{d} k^i$ csúcsot a listába tesszük.

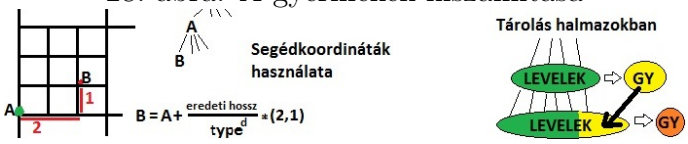
Az ismétlődési probléma megoldásához le kell mondani a kényelmesnek tűnő indexhasználatról, illetve a fa szerkezetét is újra át kell gondolni. A *halmaz* tűnik a jó megoldásnak, ám gondot okoz hogy elemein csak iterátorral lehet végighaladni és ezért azt a változtatást vezetjük be, hogy nem az egész fát tároljuk, hanem mindig csak a leveleit. Minden újabb szint létrehozásakor az iterátor végigmegy a "levélhalmaz" összes elemén, az újonan kiszámolt koordinátákat pedig egy ideiglenes halmazba, mondhatni "gyermekhalmazba" teszi. Mikor az iterátor végigért a halmazon, és az összes gyermeket kiszámoltuk, a gyermekhalmazban lévő koordinátákat bemásoljuk a "levélhalmazba", az ideiglenes halmazt pedig kiürítjük. Észrevehetjük, hogy így valóban minden csúcs leszármazottja saját magának, de csak a tőlük különböző gyermekeket számoljuk ki az újabb szint létrehozásakor, és hozzáadjuk a meglévőkhez. Így megoldódott az ismétlősédek problémája.

4.3.5. A gyermekek kiszámítása

Most, hogy rendelkezésre áll a megfelelő adatszerkezet is a koordináták eltárolására, nézzük meg milyen módszerrel tudjuk meghatározni egy adott csúcs gyermekeit. Tulajdonképpen azt a három különböző esetet kell kezelnünk, amikor az eredeti blokkot felére, harmadára, vagy negyedére kicsinyítjük, de mindhárom esetben ugyanazt a szisztémát alkalmazzuk. Az ábrát tekintve minden iterációs lépés során a négyzetek oldalhossza kisebb, ezt az oldalhosszt a lépésszámot tároló d, és az ábratípust mutató $type = 2, 3, 4$ változó segítségével könnyen megadhatjuk: $\frac{eredetihossz}{type^d}$.
Ennek birtokában már csak arra az esetekre specifikus tömbre van szükségünk, ami a felosztás blokkjainak helyzetét tárolja az ábra bal alsó sarkából kiindulva. Ez a Sierpinski háromszög esetében azt jelenti, hogy az alapmintázatot alkotó három négyzet bal alsó sarkai az ábra bal alsó sarkából: 0 jobbra +0 fel, $\frac{1}{2}$ jobbra +1 fel és 1 jobbra +0 fel oldalhosznyi lépéssel érhetők el. Ez hasonlóképpen működik a 3×3-as és a 4×4-es felosztás esetében is, ám ott 9, illetve 16 ilyen segédkoordinátát használunk fel. A tetszőleges ábrák esetén a felhasználó adja meg hogy az alapmintázatot mely blokkok alkossák, a Sierpinski szőnyeg esetén ez adott, így a program csak az alapmintázatban résztvevő blokkok koordinátáit használja fel.

Ezzel minden eszköz rendelkezésre áll az ábrák elkészítéséhez. A kiinduláshoz az alapmintázat blokkjaihoz tartozó koordinátákat betesszük a levélhalmazba. Iterátort indítunk el a levélhalmaz elemein, és minden csúcs esetében kiszámoljuk az $\frac{eredetihossz}{type^d}$ formulával az aktuális oldalhosszt, melyet a segédkoordinátákkal szorozva hozzáadunk az aktuális csúcshoz, ezzel meghatározva a gyerekek koordinátáit. A gyermekkoordinátákat a gyermekhalmazba gyűjtjük, amíg az iterátor végig nem ér a levélhalmaz elemein, majd a levélhalmazba áttesszük az új koordinátákat, ezzel kiürítve a gyermekhalmazt.

28. ábra. A gyermekek kiszámítása



4.3.6. A bal alsó hiba

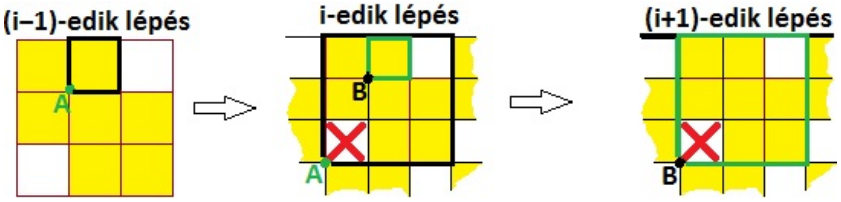
Ez a módszer sikeresnek bizonyul, a program valóban elkészíti az eltervezett ábrákat, de ha jobban megnézzük, valami mégsem működik megfelelően. A *29.* ábrán láthatjuk, hogy az alapmintázat üresen hagyja a *bal alsó* blokkot, ám ez a további lépésekben nem valósult meg. A tároláshoz a bal alsó koordinátákat vettük alapul, ezért azzal az esettel amikor a bal alsó blokk üres, külön

foglalkozni kell. Azok miatt az esetek miatt, amikor a bal alsó sarok üres, a fa felépítését nem a $(0, 0)$ koordináta levélhalmazba helyezésével kezdtük, hanem az alapmintázatban szereplő blokkok koordinátáit vettük kezdeti értékeknek.

29. ábra. A bal alsó sarok hibája.

Úgy tűnik azonban, hogy ez nem elég. Gondoljuk csak meg, hogy a leveleken elhelyezkedő koordináták mindegyike a következő szint felépítésekor, azaz az eggyel lejjebb lévő szinten bal alsó sarokká válik. Eddig az algoritmusban csak koordináták hozzáadása történt, de látszik, hogy a bal alsó sarok üresen maradása esetén, minden csúcsot törölnünk kell a levélhalmazból miután gyermekeit kiszámoltuk.

30. ábra. A bal alsóvá válik és kikerül a halmazból.



Ennek megvalósításakor csak arra kell ügyelni, hogy mivel iterátorral megyünk végig a halmazon, közvetlen a gyermekek kiszámítása után nem törölhetünk egy elemet, mert az iterátor nem fog tudni továbblépni, ha kitöröljük az elemet amelyre épp mutat. A megoldás tehát az, hogy beiktatunk egy ellenőrzőpontot oda, ahol az iterátor már végigért a levélhalmazon és kiszámoltuk az összes gyermeket. Itt megnézzük, hogy üresen áll-e a bal alsó sarok, és ha igen, akkor az egész levélhalmazt kiürítjük, majd csak ez után tesszük át a gyermekhalmaz elemeit a levélhalmazba. Ezt a hibát is kiküszöbölve, a program első négy menüpontja az elvárások szerint működik.

4.3.7. Animációk

Az animációk jól szemléltetik az ábra alakulásást, és elkészítésük nem sokban különbözik a képekétől, hiszen a módszer valójában ugyanaz. Az animáció megvalósításakor, minden kép megjelenése után egy kis késleltetéssel frissíteni kell a grafikus ablakot az újraábrázoláshoz. A képsor végigjátszása után pedig automatikusan újraindul az animáció.
Ez azt jelenti, hogy minden iterációs lépés után szükségünk van az addig kiszámolt koordinátákra és az ábrázolandó négyzetek oldalhosszára. Ehhez elég a meglévő kódot lemásolni és úgy átalakítani, hogy az összes koordináta kiszámolása utáni ábrázolás helyett, kiszámolás közben ábrázoljuk az aktuális négyzeteket. Tehát csupán az ábrázolás és az ablakfrissítés néhány sorát kell beszúrnunk a ciklusba, amely a koordinátákat számolja, minden iterációs lépés végére. Amiről viszont nem szabad megfeledkeznünk az a koordináta halmaz kiürítése az utolsó ábrázolás után, hiszen így a képsor újrakezdésekor a koordináta kiszámítást is újra végigcsinálja a program. Ha a halmaz kiürítése elmarad, ahányszor végigfut a képsor, annyiszor kerül be az összes koordináta a halmazba, és ábrázoláskor mindegyik négyzet megjelenik egyszerre az aktuális oldalhosszal, ami elég nagy káoszhoz vezet.

4.3.8. Véletlen szőnyeg játék

A véletlen szőnyeg játékban egy menüpont alatt több különböző ábra vagy animáció közül választhat a felhasználó. Azért kapta a játék nevet, mert a felhasználó a következő beállítások mindegyikéből a megfelelő opciót kiválasztva határozhatja meg, hogy mit szeretne látni:

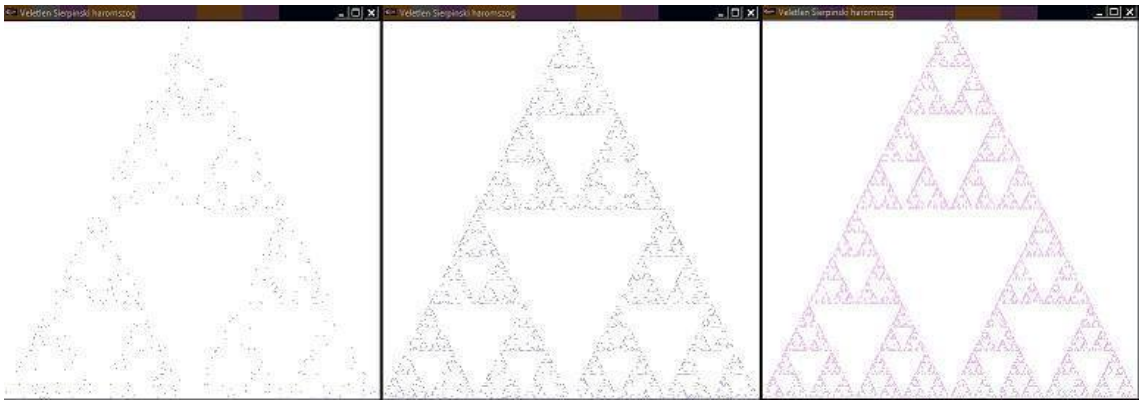
- Alapmintázat: egyetlen véletlenszerű alapmintázat, iterációs lépésenként különböző alapmintázat, vagy a kettő összehasonlítása.
- Méret: 3×3-as vagy 4×4-es szőnyeg.
- Lépésszám: hány iterciós lépést tegyen meg a program.
- Árbázolás: kép a végeredményről vagy animáció a folyamatról.

Az egyetlen véletlenszerű alapmintázat esete egyszerű, hiszen csak annyiban különbözik a tetszőleges szőnyegtől, hogy az alapmintázatot nem a felhasználó adja meg, hanem a program véletlenszerűen dönti el hogy mely blokkok legyenek kitöltve és melyek maradjanak üresen. Az alapmintázat elkészülése után ugyanúgy folytatódik a kiszámolás és az ábrázolás mint a tetszőleges esetében. A lépésenként különböző alapmintázat választásakor a megadott lépésszámmal megegyező mennyiségű véletlenszerű alapmintázatot tárolunk el egy tömbben.

Itt a változatatás annyi, hogy újabb iterációs lépésben a tömbben soron következő alapmintázatot alkalmazzuk. Ekkor az alapmintázatot mutató ablakban az alapmintázatok animációként jelennek meg, citromsárga színnel az első, majd fokozatosan narancsosabb árnyalatban a továbbiak.
Amikor a fentiek összehasonlításra kerülnek, az egyetlen véletlenszerű alapmintázata lesz a lépésenként különböző elsője. Ezáltal az első lépésben kialakuló formája, alakja a két fraktálnak ugyanaz lesz, az ezen belüli barázdák és mintázat lesz csak különböző. Kép megtekintését választva egy ablakba kerül, különböző színnel a két véletlen szőnyeg, ám animációik külön ablakban jelennek meg. Természetesen az alapmintázatok most is animációként jelennek meg.

4.3.9. Véletlen Sierpinski háromszög

31. ábra. Véletlen Sierpinski háromszög (negatív)



A véletlen Sierpinski háromszög generálása különbözik az eddigi módszertől. Ebben az utolsó menüpontban a háromszöget alkotó pixelek egyesével kapják meg koordinátáikat és villannak fel a képernyőn. Ez természetesen animáció formájában látható, melynek végén, amikor minden pont megjelent, a háromszög zölden villog, majd újra kezdődik.
Amint azt láthattuk az elején, az 512-es háromszög esetében kilenc lépést tudunk megtenni, míg minden kiskocka egyetlen pont, vagyis pixel lesz. Vegyük észre, hogy minden kilenc hosszú $\{0, 1, 2\}$ számokból álló sorozathoz a Sierpinski háromszög egy bizonyos pontja tartozik. Az első számjegy mehgatározza, hogy a három $\frac{1}{2}$ oldalhosszú háromszög közül melyikben helyezkedik el a pont, a következő számjegy azt, hogy az azon belüli $\frac{1}{4}$ oldalúak közül melyikben, és így tovább. Ezt végiggondolva nincs más hátra, mint véletlenszerű kilenc hosszú $\{0, 1, 2\}$ sorozatokat generálunk, majd ábrázoljuk azt általuk meghatározott pontokat a képernyőn.
Kérdés lehet, hogy hány pontra van szükség ahoz, hogy jól kivehetően kirajzolódjon a végeredmény. Ehhez elég azt végiggondolnunk, hogy 1 darab nagy

háromszögből indulunk ki, amelyből az első lépés során 3 kisebb lesz, majd a következő lépésben a 3 kisebb mindegyikéből még 3, összesen 9. Így kilenc lépés megtétele után 3^9, azaz 19683 kis háromszög, a mi esetünkben pont lesz. A program 19600 véletlen számsort generál, ám ezek között is lehetnek megegyezők, azaz illeszkedő pontok. Emiatt a kapott Sierpinski háromszögnek hiányoznak pontjai, "lyukacsosnak" tűnik, tükrözve generálásának módját.

5. CD Melléklet

5.1. Waterfall program

A futtatandó file a *"waterfall\bin\Debug\waterfall.exe"*. A program Windows 7 alatt fut.

5.2. Fractal program

A futtatandó file a *"fractal backslash bin\Debug\fractal.exe"*. A program Windows 7 alatt fut.

5.2.1. Megjegyzés

A programok futásához szükségesek bizonyos *header* fájlok. Ezek közül nem biztos hogy mindegyik megtalálható a számítógépen, ezt egy felugró hibaüzenet jelzi indítás kísérletekor. Ebben az esetben a hiányzó fájlokat a *"C:\Windows\System32"* könyvtárba kell másolni.

5.3. Szakdolgozat

A szakdolgozat *PDF* formátumban.

6. Konklúzió

A dolgozat során tisztázhattunk néhány matematikai fogalmat, amely segít a fraktálok vizsgálatában és megértésében. Megismertük néhány nevezetes példáját ezeknek az euklideszi geometriával le nem írható alakzatoknak, önhasonló tulajdonságuk felhasználásával becslést adhattunk nem egész dimenziójukra, sőt még a természetben megtalálható fraktálszerű képződményekről is esett szó. Ezek egyikének szemléltetésére készült a *waterfall* program.
A későbbi fejezetekben láthattunk olyan algoritmusokat, melyeket a számítógépes grafikában használnak élethű domborzat megalkotására. Ezekben a Koch-görbéhez, illetve a Mandelbrot-halmaz határához hasonló bonyolult, eltarajosodó horizontvonalat, illetve felszínt próbáltunk generálni, a vonal vagy sík pontjainak véletlenszerű magasságba mozdításával. A számítógépes játékok, filmek tájképeinek készítéséhez gyakran használják ezeket az algoritmusokat.
Ezután következett a fraktál képtömörítés ötletének és módszerének felvázolása. A módszer legnagyobb hibája, hogy a kevés önhasonlóságot mutató hétköznapi képek esetében nehéz az elkódolás. Így hiába érhetünk el jóval kisebb méretet és gyorsabb dekódolási folyamatot, mint más tömörítési módszerek, nem biztos hogy megéri az elkódolásba befektetett sok munka.
A dekódolás módját és néhány fraktál generálását alapul véve íródott a *fractal* program. Ugyan a kép dekódolási folyamatánál jóval egyszerűbb, mégis egy szemléletes és szórakoztató alkalmazás. Implementálása sok kérdést és nehézséget vetett fel, ráadásul grafikus ábrázolással, animációval eddigi tanulmányaim során még nem foglalkoztam. Ezen problémák vizsgálatával, megoldásával sok új ismeretre tehettem szert. A dolgozat utolsó fejezeteiben a program megvalósításának alapjait vázoltam, valamint a közben elkövetett és javított hibák némelyikére is kitértem. Akár komplett alfejezetként, mint a megfelelő adatszerkezet megtalálása és a bal alsó hiba esetén, vagy egyszerűen csak megjegyzésként, hogy mire fontos ügyelni.
A dolgozattal elvégzett munka során sok új és hasznos dolgot tanultam, és úgy gondolom az érdeklődők számára is könnyen megérthető olvasmány, melyet a mellékelt programok még izgalmasabbá és interaktívabbá tesznek.

Hivatkozások

[1] Kenneth Falconer
Fractal Geometry: Mathematical Foundations and Applications
John Wiley & Sons, Ltd., Chichester, 1990

[2] Michael F. Barnsley
Fractals everywhere
Second edition. Boston, MA, 1993

[3] Benoit B. Mandelbrot
The Fractal Geometry of Nature
Henry Holt and Company, 1983

[4] Gerald Edgar
Measure, Topology, and Fractal Geometry
Springer, 2007

[5] Fractal Geometry MN1
http://read.pudn.com/downloads36/ebook/112856/FRACTAL%20GEOMETRY%20MN1.pdf

[6] Nimrod Peleg
Fractal Image Coding(IFS), 2008
http://cs.haifa.ac.il/ nimrod/Compression/Image/I4frct2008.pdf

[7] Wikipedia (angol nyelvű): Fractal
http://en.wikipedia.org/wiki/Fractal

[8] Wikipédia (magyar nyelvű): Cantor-halmaz
http://hu.wikipedia.org/wiki/Cantor-halmaz

[9] Wolfram MathWorld: Julia set
http://mathworld.wolfram.com/JuliaSet.html

[10] Paul Bourke
Julia Set Fractal, 2001
http://paulbourke.net/fractals/juliaset/

[11] Wikipédia (magyar nyelvű): Koch-görbe
http://hu.wikipedia.org/wiki/Koch-g%C3% B6rbe

[12] Wikipedia (angol nyelvű): Sierpinski triangle
http://en.wikipedia.org/wiki/Sierpinski_triangle

[13] Wikipédia (magyar nyelvű): Sierpinski-szőnyeg
http://hu.wikipedia.org/wiki/Sierpi%C5%84ski-sz%C5%91nyeg

[14] Wikipédia (magyar nyelvű): Manger-szivacs
http://hu.wikipedia.org/wiki/Menger-szivacs

[15] Three-Cornered Things: Manger sponge
http://blog.zacharyabel.com/tag/menger-sponge/

[16] M. K. Hassan
Fraktálok, multifraktálok és a bonyolultság tudománya
Virtual Physics c. folyóirat
http://www.kfki.hu/ cheminfo/hun/olvaso/fraktal/frakmult.html

[17] Milan A. Joshi, Dr. S. M. Padhye
Hidden Dimensions of Nature
http://www.slideshare.net/MILANJOSHIJI/fractal-geometry-and-its-applications-by-milan-a-joshi

[18] Wikipedia (angol nyelvű): Fractal landscape
http://en.wikipedia.org/wiki/Fractal_landscape

[19] Paul Martz
Generating Random Fractal Terrain
http://www.gameprogrammer.com/fractal.html

[20] Wikipedia (angol nyelvű): Fractal compression
http://en.wikipedia.org/wiki/Fractal_compression

[21] Wikipedia (angol nyelvű): Dragon curve
http://en.wikipedia.org/wiki/Dragon_curve

[22] Molnaár Katalin
Lindenmayer-rendszerek a középiskolában
ELTE TTK, Szakdolgozat, 2008