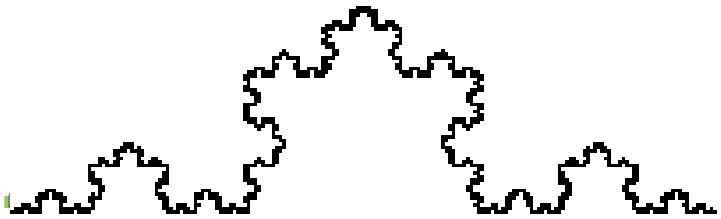
Fraktálok és káosz

Szirmay-Kalos László

A természet geometriája

- Euklideszi geometria metrikus
  - „Sima” egyenesre/síkra épít (analízis: differenciálás)
    - Kicsiben mindenki lineáris: Skálafüggőség
  - Méret lényeges (milyen méret: dimenzió kapcsolat)
    - Méret: szakasz, négyzet, kocka lefedés
    - Dimenzió: algebra = független változók, topológia=kettévágás
  - Hossz (1D); terület (2D); térfogat (3D)
  - Ember alkotta objektumokra jó
- Természet
  - Méret nem releváns (zérus vagy végtelen):
    - dimenzió?
  - Nem sima (nem differenciálható)
    - Kicsiben is ugyanolyan: Skálafüggetlenség

(Helge von) Koch görbe: hossz végtelen



$$l_n = l_0 \left(\frac{4}{3}\right)^n \to \infty$$

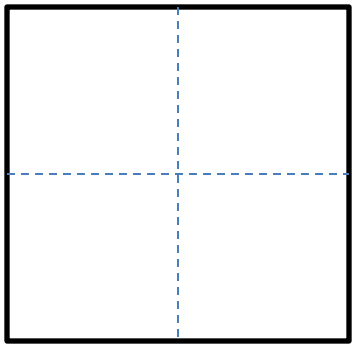
- Véges tartományban végtelen hosszú:
  - Dimenzió > 1
- Területe zérus
  - Dimenzió < 2
- Folytonos
- Sehol sem differenciálható (tüskés)
- Önhasonló

(Felix) Hausdorff dimenzió önhasonló objektumokra

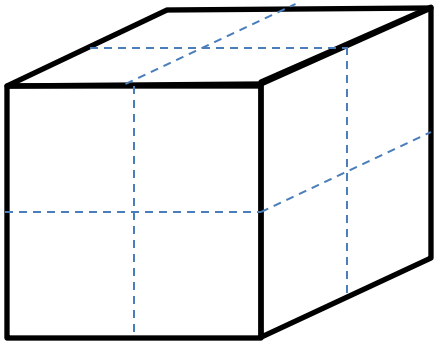
$r = 1/2$

$N = 2$

$N = 4$



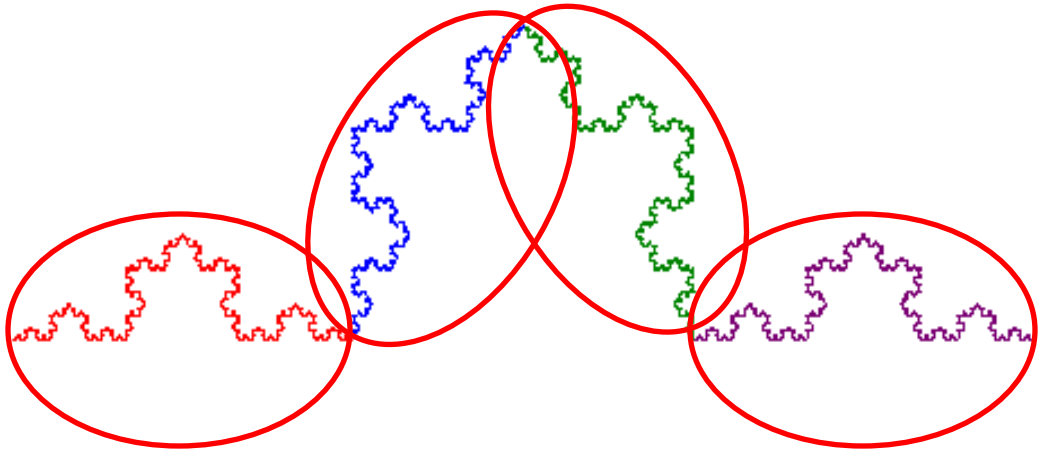
$N = 8$



$N = 1/r^D$

$$D = \frac{\log(N)}{\log(1/r)}$$

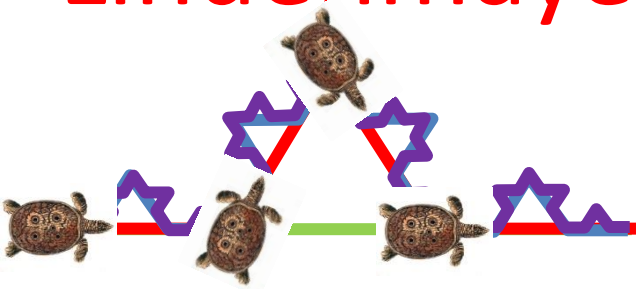
Koch görbe Hausdorff dimenziója



$$r = 1/3$$
$$N = 4$$

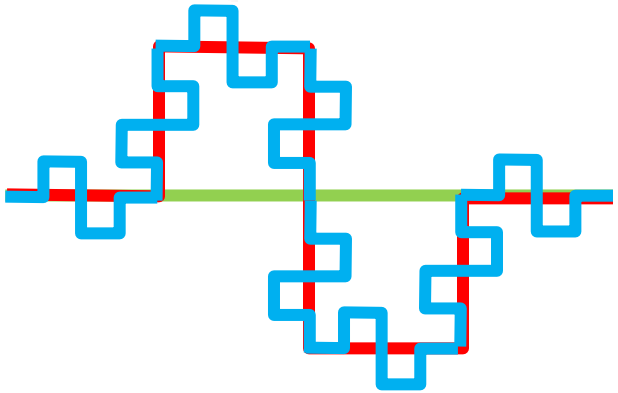
$$D = \frac{\log(4)}{\log(3)} \approx 1.26$$

Lindenmayer (Arisztid) rendszerek



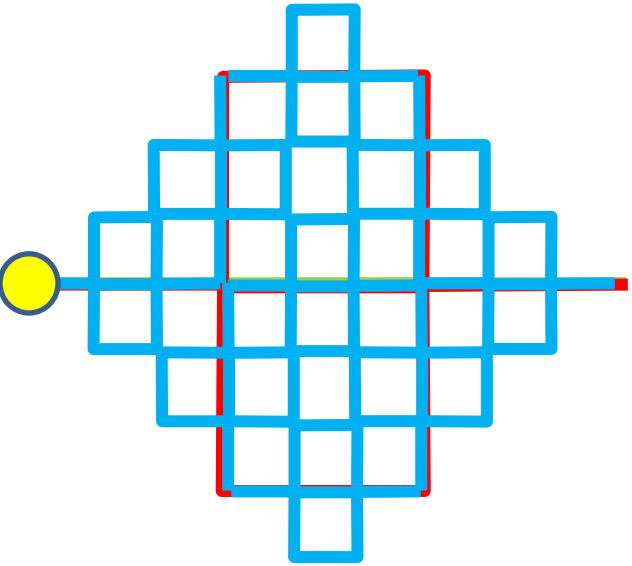
F→F+60F-120F+60F

$r = 1/3$, $N = 4$, $D=\log(4)/\log(3)=1.26$



F→F+90F-90F-90FF+90F+90F-90F

$r = 1/4$, $N = 8$, $D=\log(8)/\log(4)=1.5$



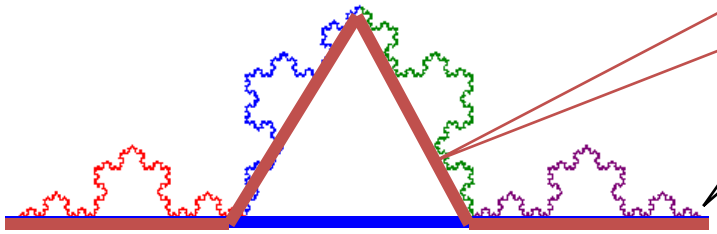
F→F+90F-90F-90F-90F+90F+90F+90F-90F

$r = 1/3$, $N = 9$, $D=\log(9)/\log(3)=2$

Peano/Hilbert térkitöltő görbe

Nem önhasonló objektumok: vonalzó dimenzió

$N = 1/r^D$



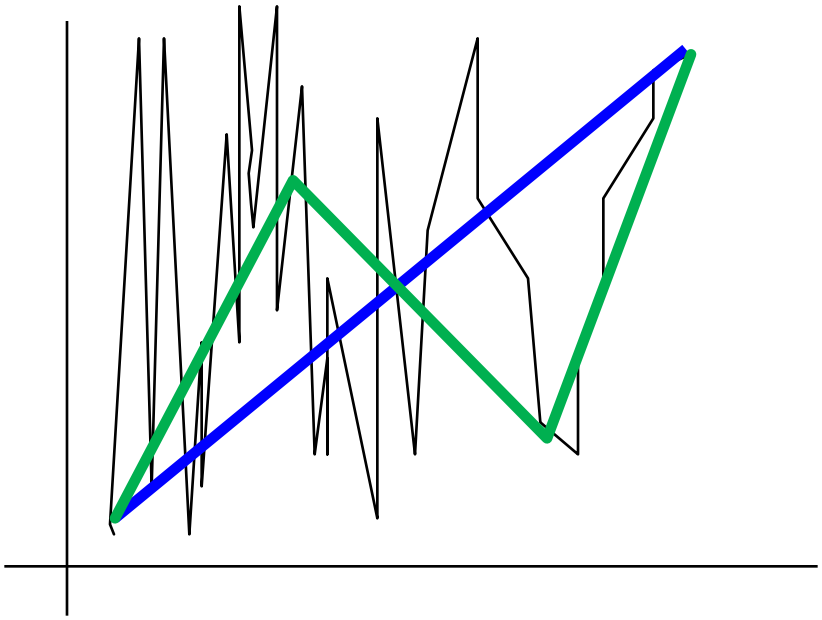
Vonalzó (l)	db
l	1
r =1/3	N = 4
r^2	N^2
r^m	N^m

$$Hossz(l) = l\ \text{db} = l\ N^m = l\,(1/r^D)^m = l\,(1/r^m)^D = 1/l^{D\text{-}1}$$

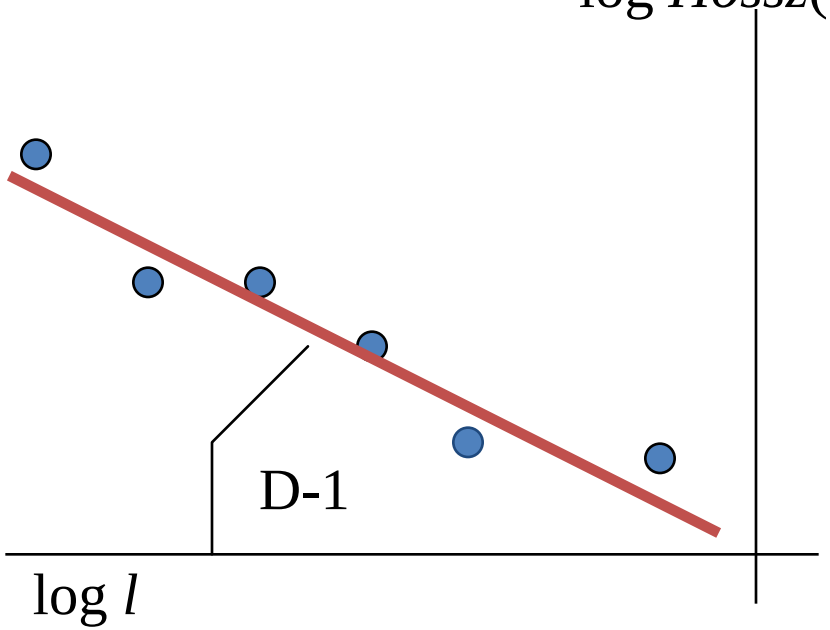
$$D = -\log(Hossz(l))/\log(l) + 1$$

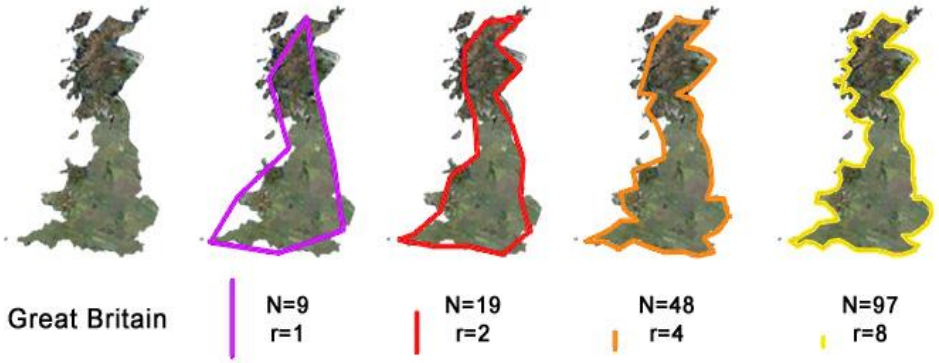
Dimenziómérés = hosszmérés

EEG görbe



log *Hossz*(*l*)

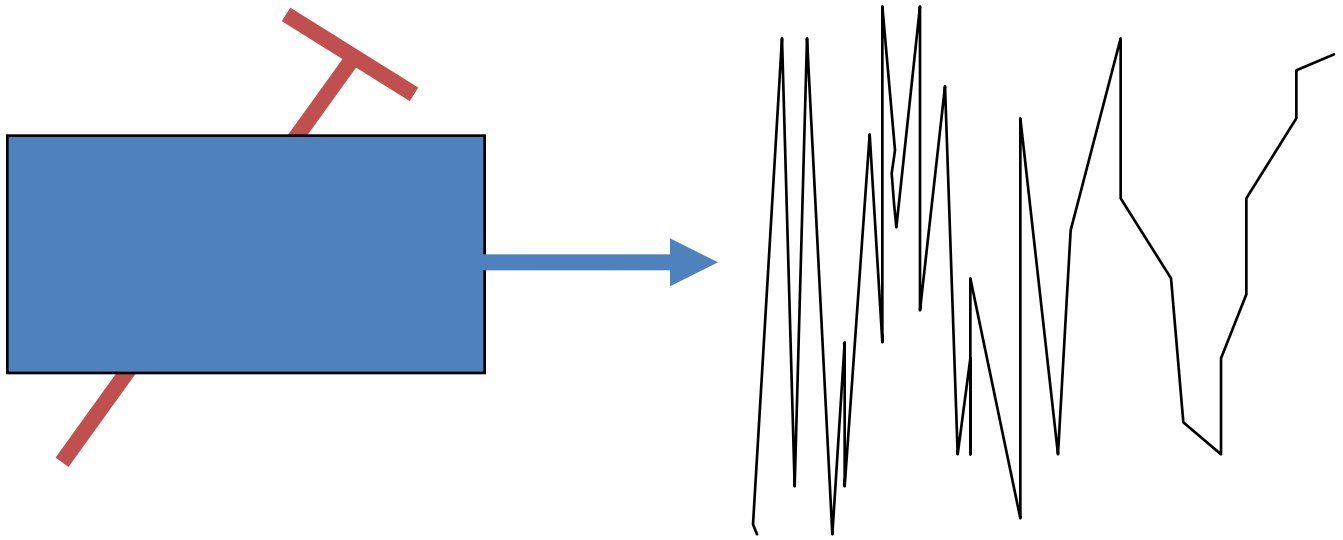




<u>Alkalmazás:</u>

Természetes objektumok elkülönítése és kategorizálása

Fraktálok előállítása

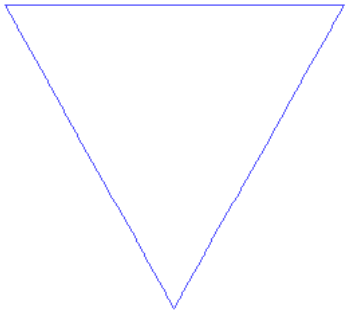


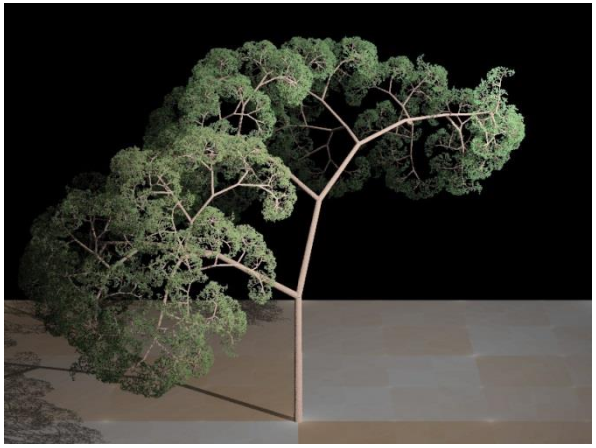
<u>Matematikai gépek:</u>

- L-rendszerek
- Fraktális ($1/f$) zaj: Brown mozgás, Perlin zaj
- Kaotikus dinamikus rendszerek (véletlenszám generátor)

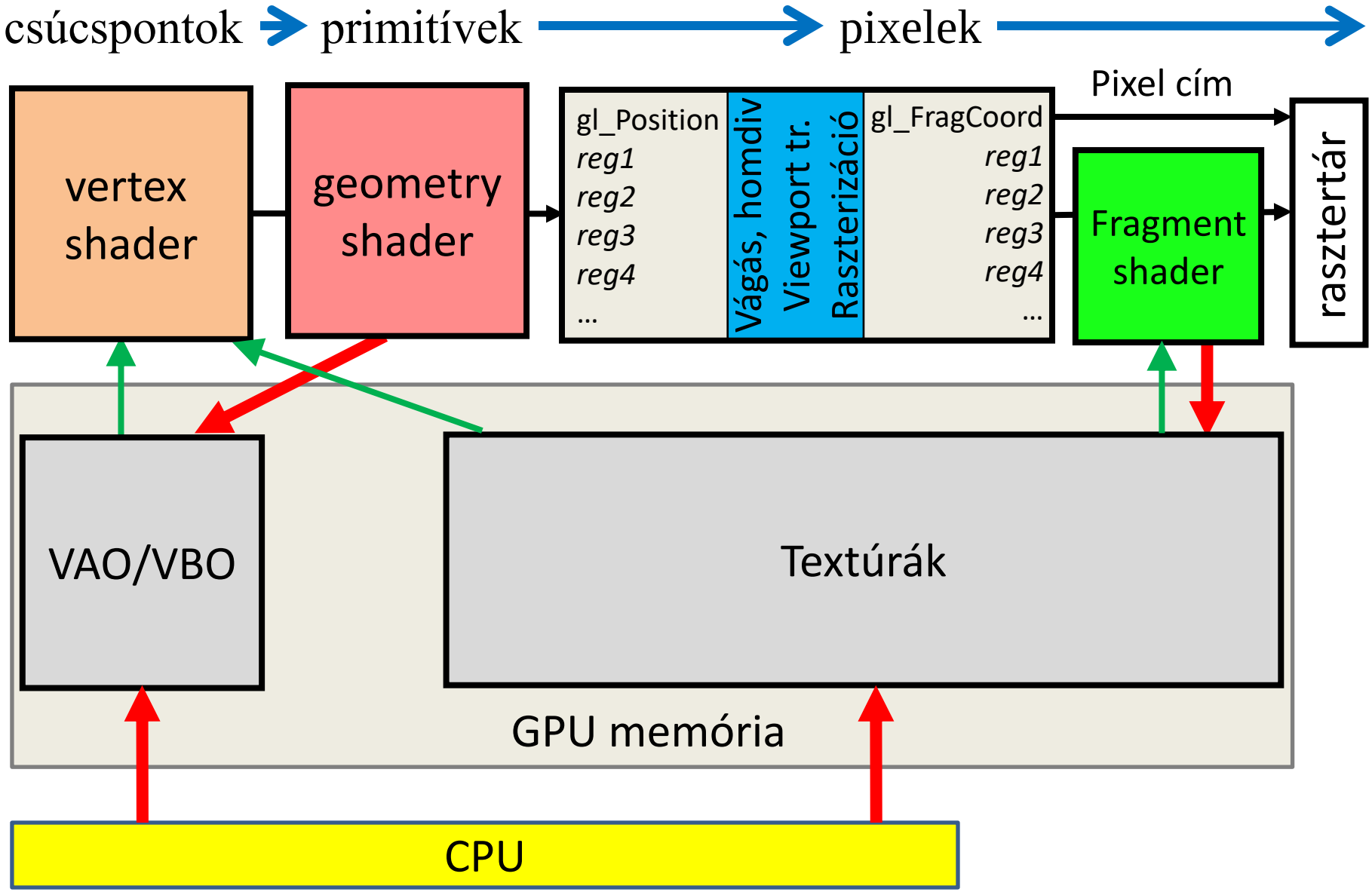
Lindenmayer rendszerek

F→F+60F-120F+60F

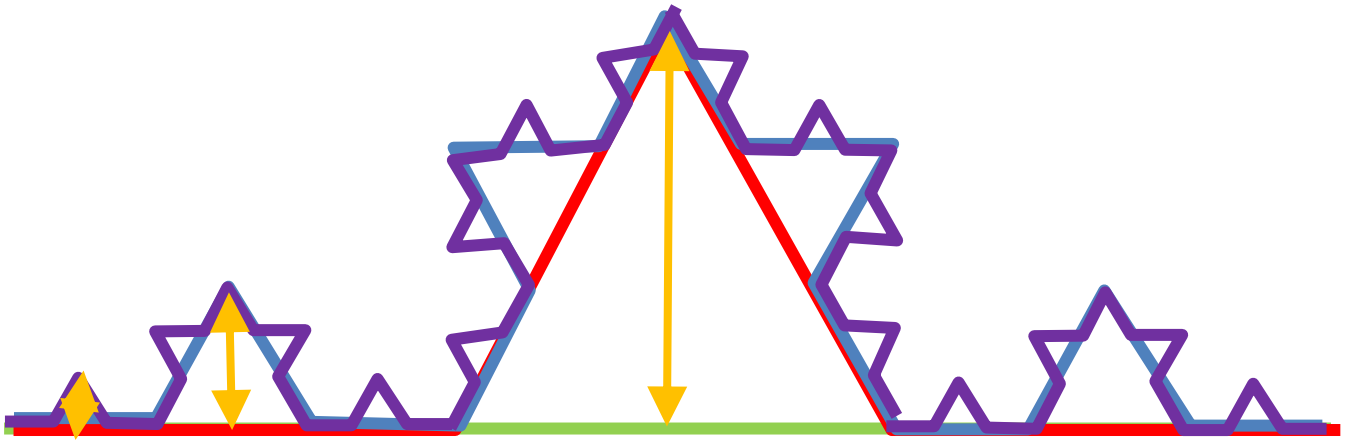




Geometry shader



Fraktális zaj

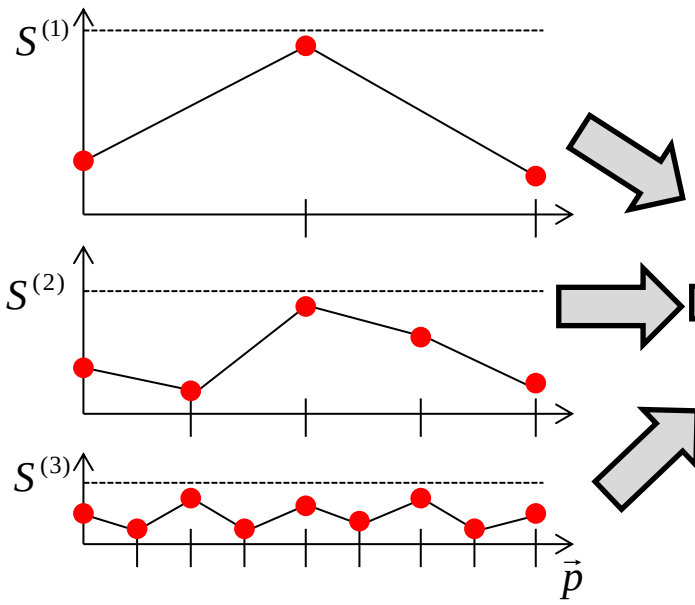


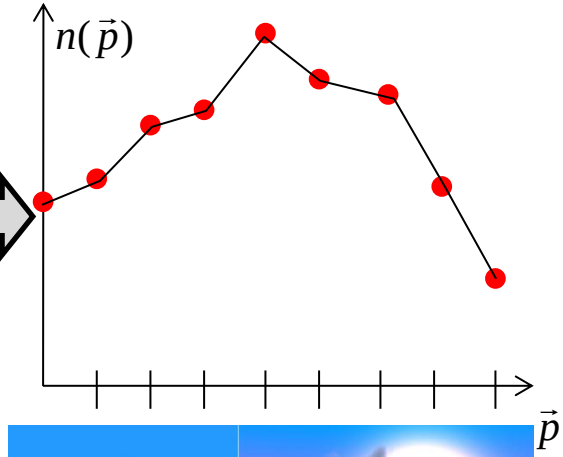
$\sigma/9$ $\sigma/3$ $\sigma = 1/(2\sqrt{3})$

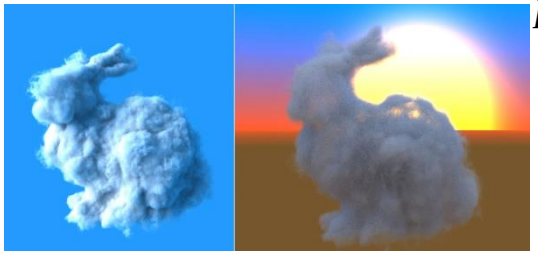
Skála	Részlet
$N = 4$	σ
$N^2 = 16$	$\sigma/3$
…	
$N^m = 4^m$	$\sigma/3^{m-1}$

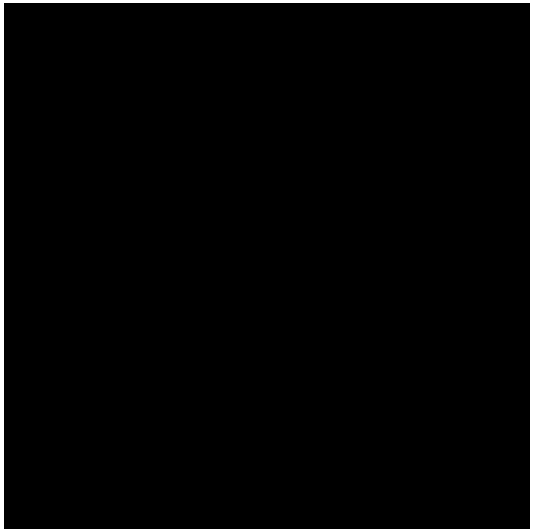
szórás

(Ken) Perlin zaj









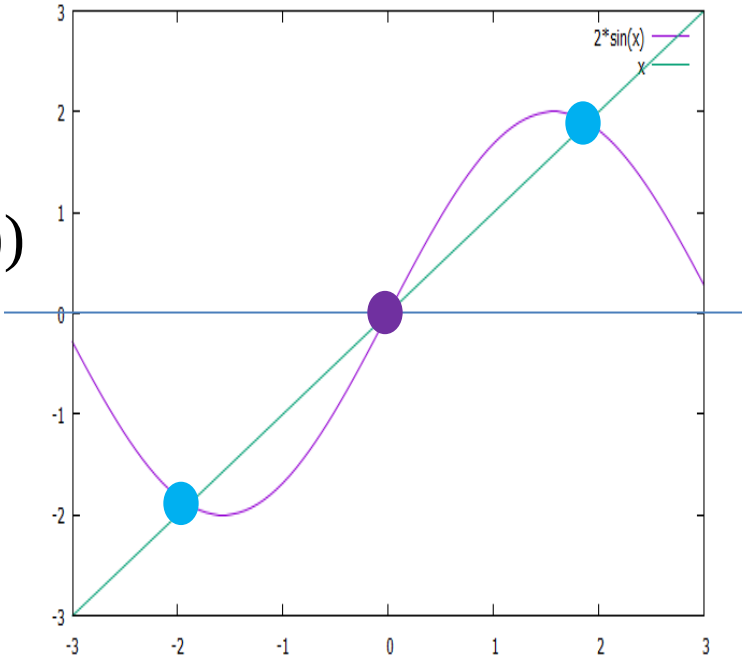
Iterált függvények, fix point, stabilitás

$$x_{n+1} = F(x_n)$$

$$x^* = F(x^*)$$

Fix point

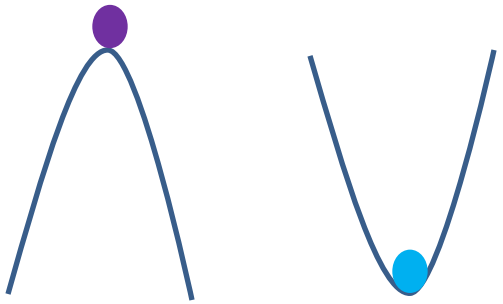
$$x_1 = F(x_0)$$
$$x_2 = F(x_1) = F(F(x_0))$$
$$x_3 = F(x_2) = F(F(F(x_0)))$$
$$\dots$$



Stabilitás: $|F'(x^*)| < 1$

$$x^*+\Delta x_{n+1} = F(x^*+\Delta x_n)$$
$$\approx F(x^*) + F'(x^*)\cdot\Delta x_n$$

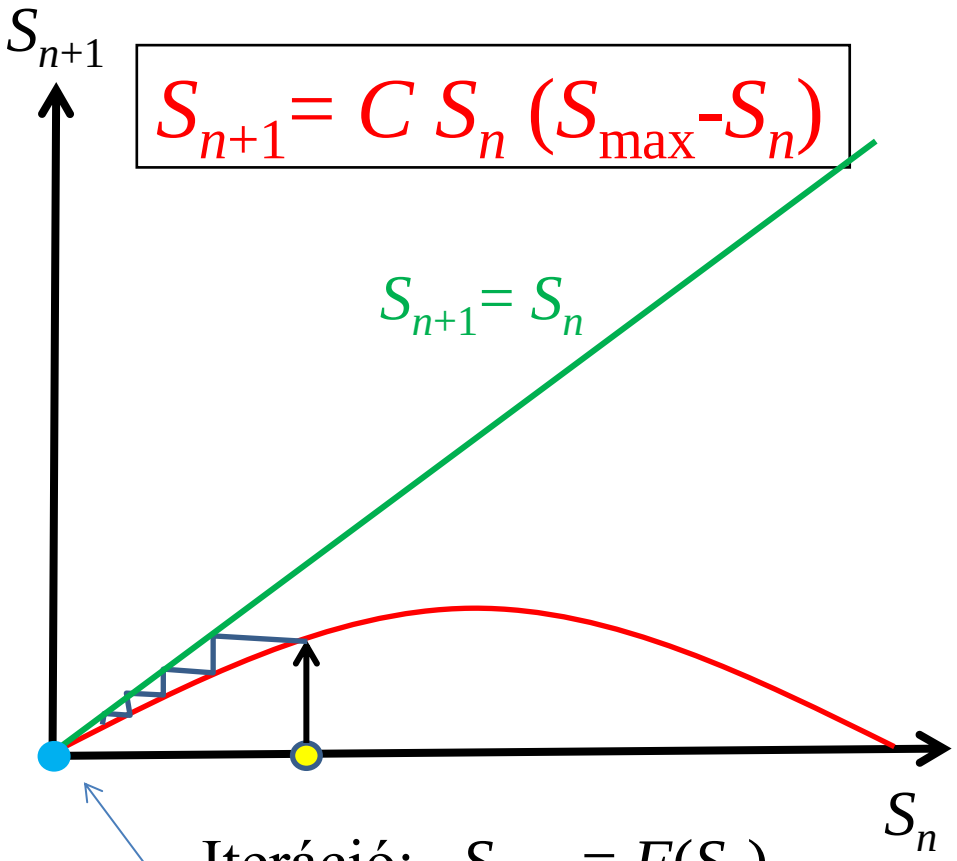
$$|\Delta x_{n+1}| \approx |F'(x^*)|\cdot|\Delta x_n|$$

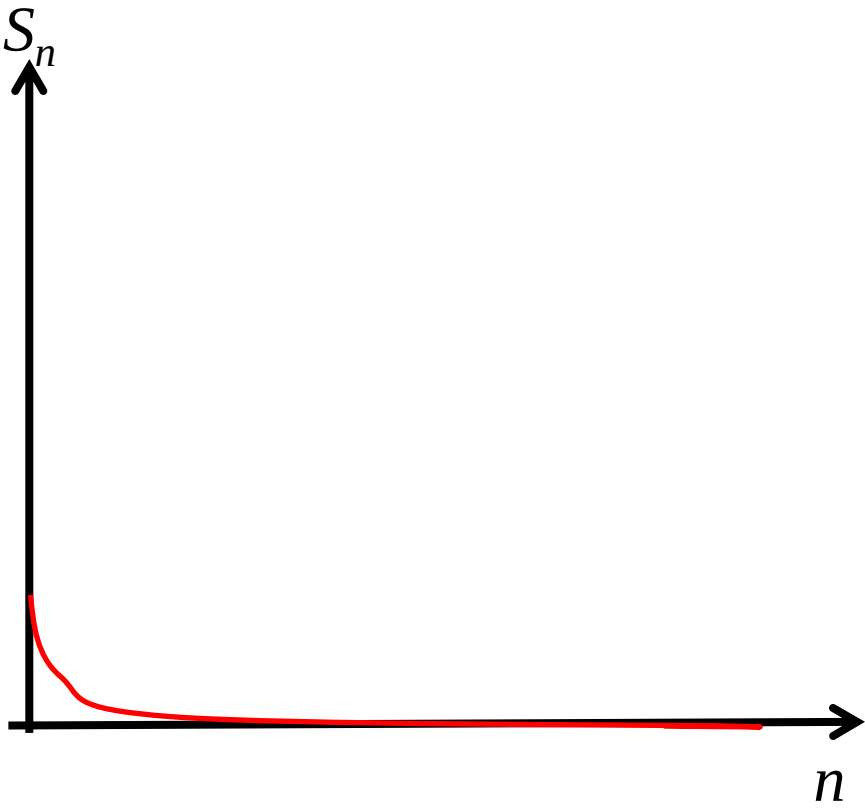


Labilis

Stabil

Kaotikus dinamikus rendszer: nyulak kis *C* értékre



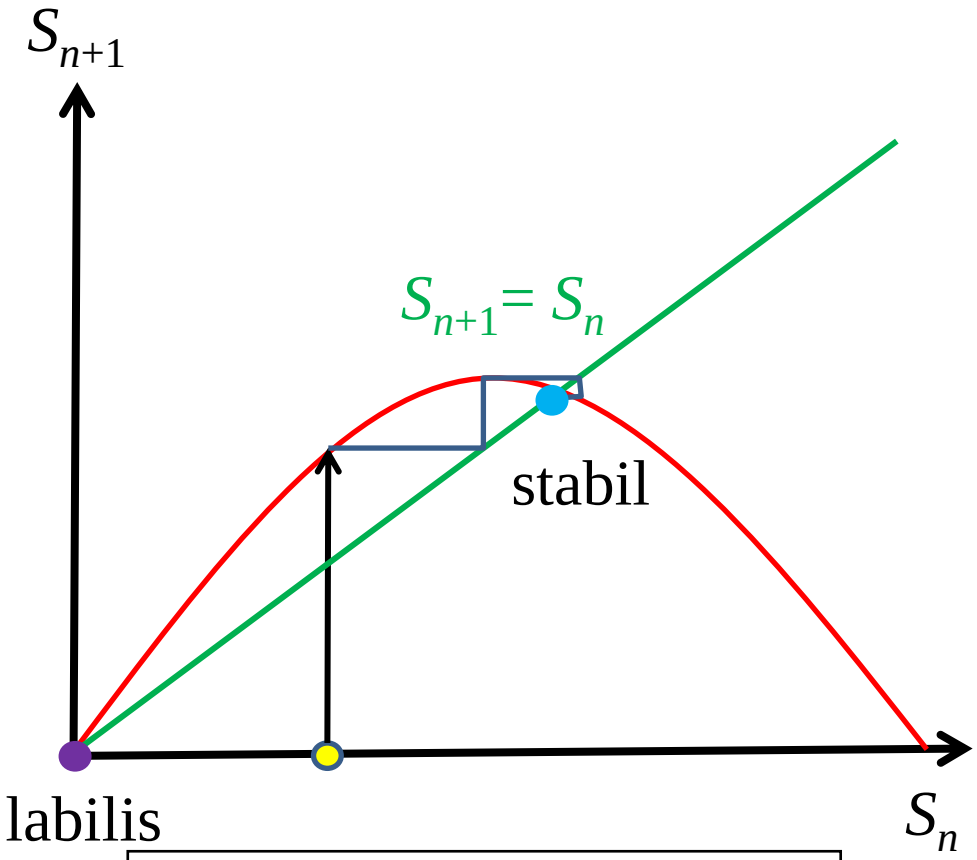


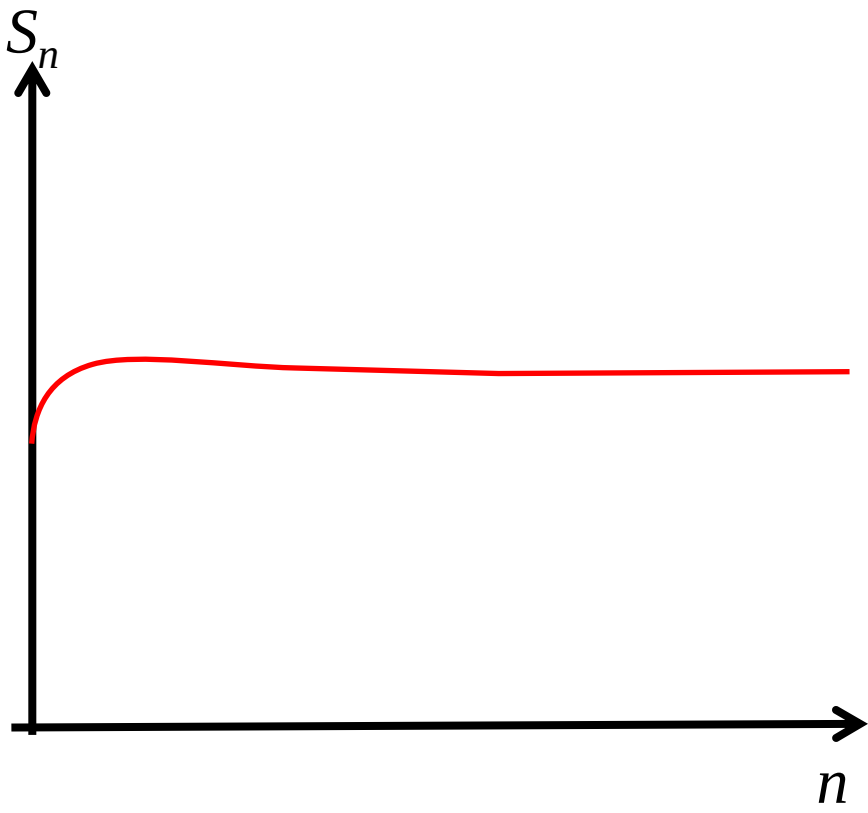
Iteráció: $S_{n+1} = F(S_n)$

Fix pont: $S^* = F(S^*)$

Stabil: $|F'(S^*)| < 1$

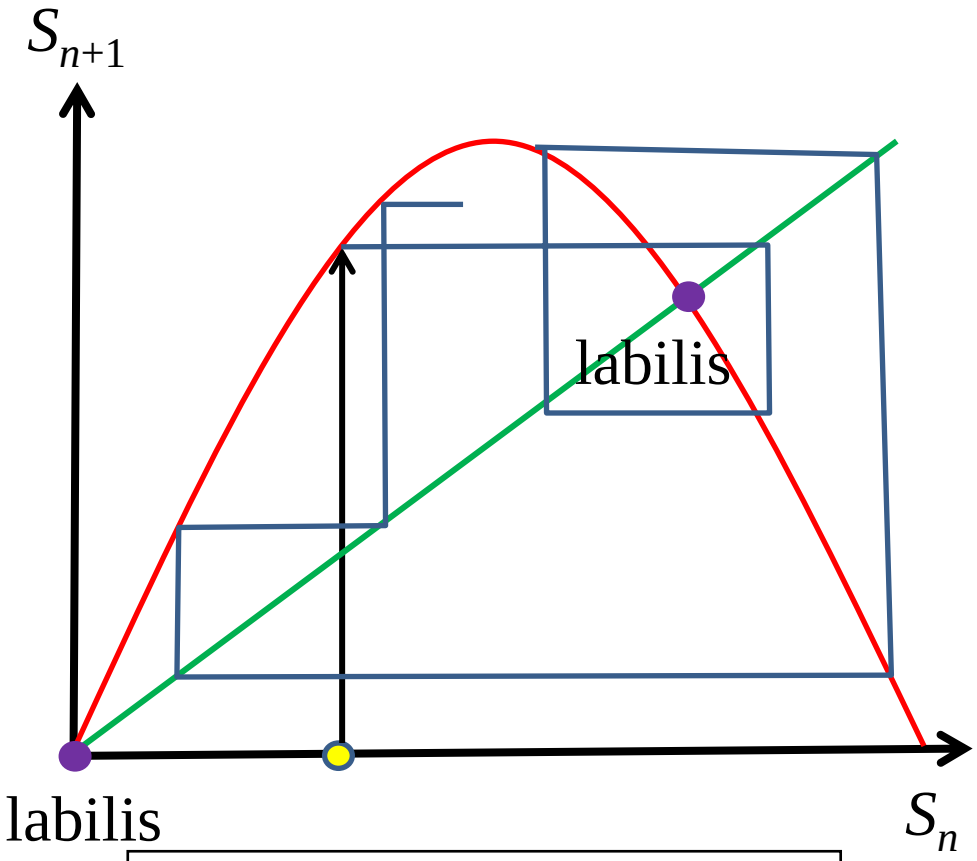
Kaotikus dinamikus rendszer: nyulak közepes *C* értékre

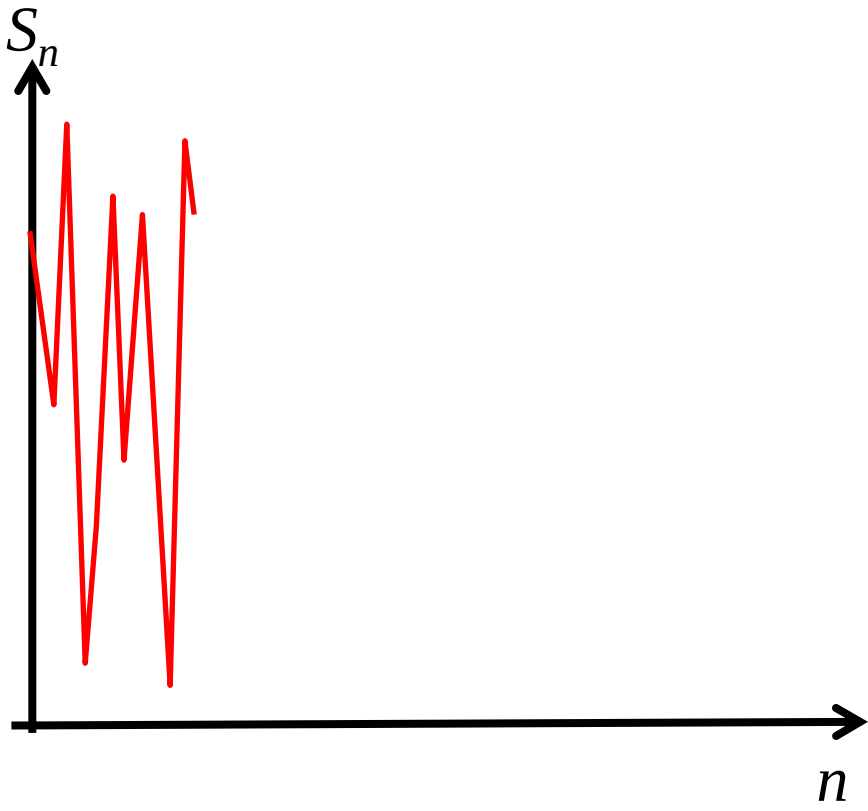




$$S_{n+1}= C\, S_n\, (S_{\max}-S_n)$$

Kaotikus dinamikus rendszer: nyulak nagy *C* értékre





$$S_{n+1} = C\, S_n\, (S_{\max} - S_n)$$

Káosz

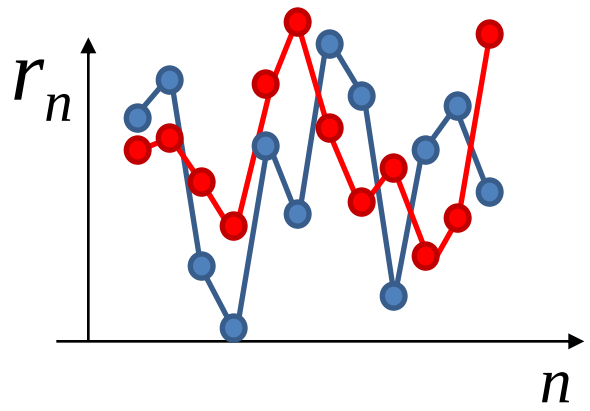
- A rendszer determinisztikus, de
- Kezdeti állapot hatása eltűnik
  - Kis perturbáció nagyon eltérő viselkedéshez vezet
- Auto-korrelációs függvény zérushoz tart
  - Korábbi állapot gyengén befolyásolja a sokkal későbbit
  - Megjósolhatatlanság
- Teljesítmény sűrűség spektrum nem tart zérushoz
  - Nagy frekvencia

```
static int r = 3;

void seed(int s) { r = s; }

int rand( ) {
    r = F(r);
    return r;
}
```

Pszeudó véletlenszám generátor

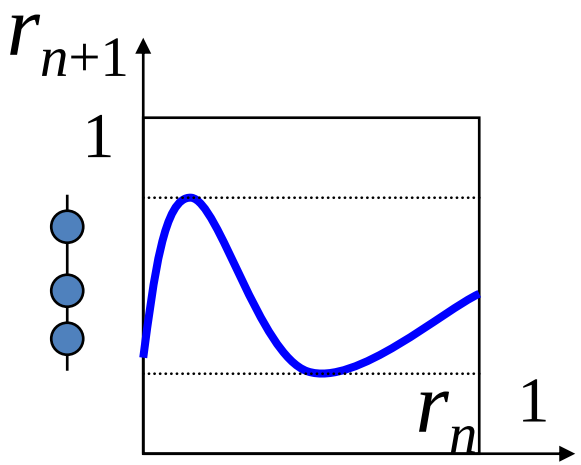
Iterált függvény: $r_{n+1} = F(r_n)$

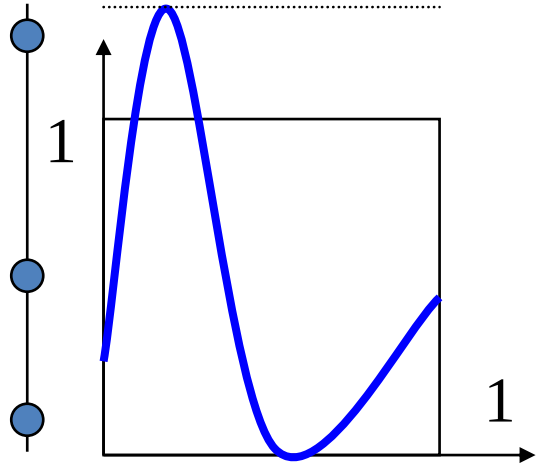
véletlenként hat

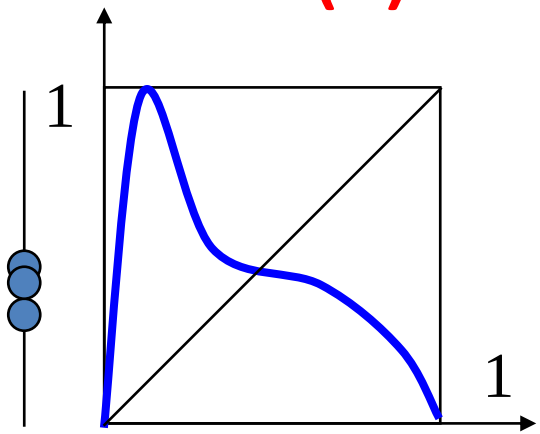


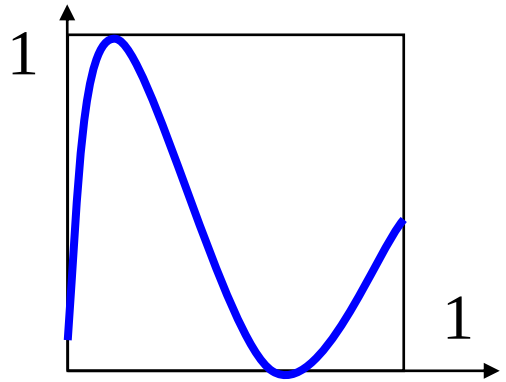
F nagy derivált!

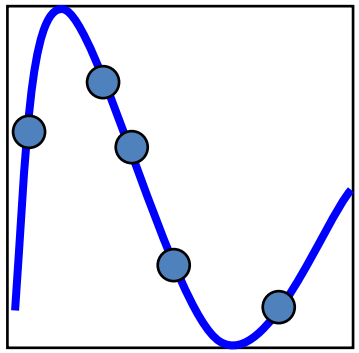
Rossz $F(r)$

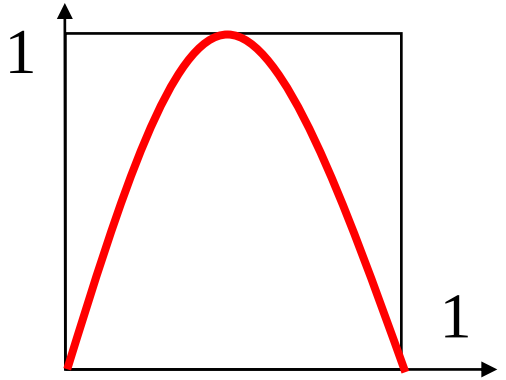


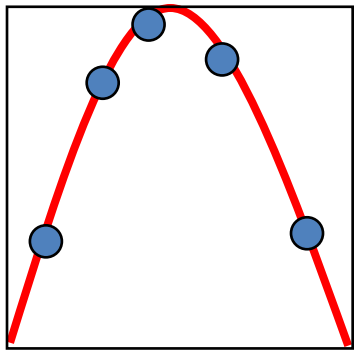












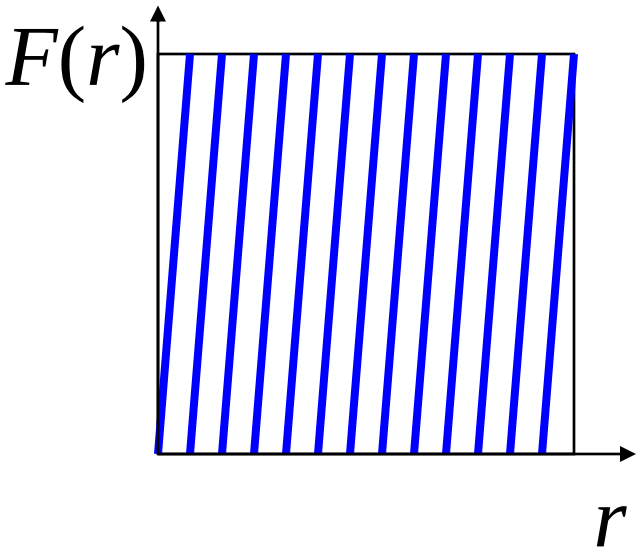
$(r_n, r_{n+1}) = (r_n, F(r_n))$
párok

Kongruens generátor

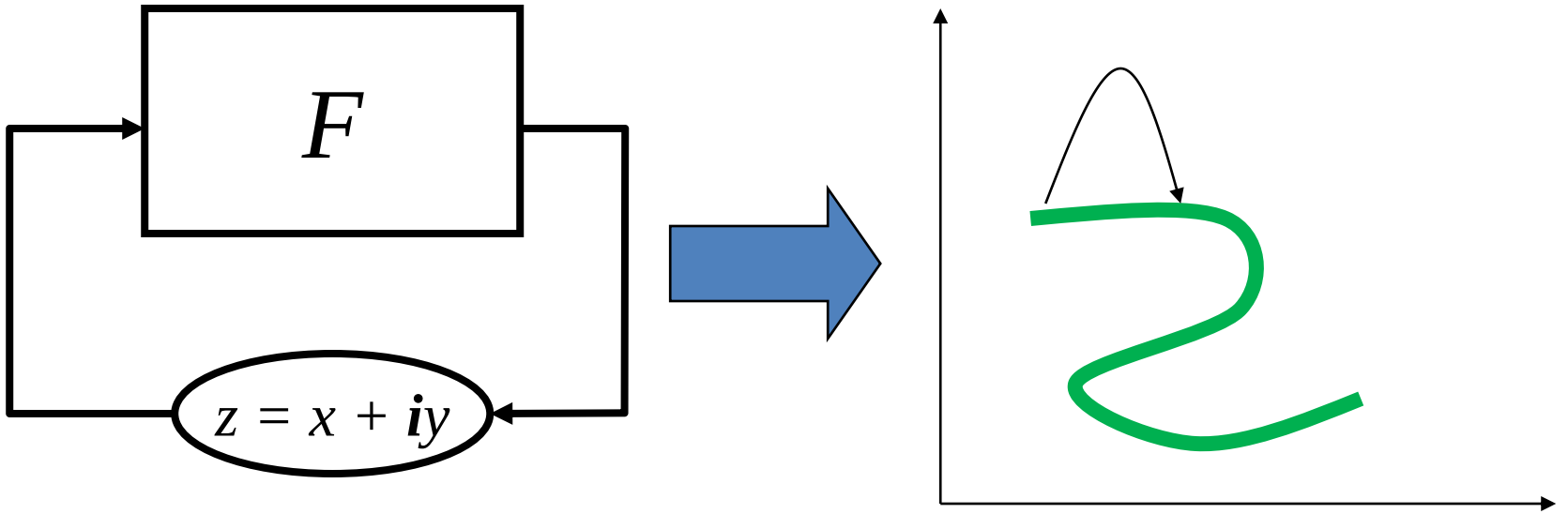
$$F(r) = \{\, g \cdot r + c \,\}$$

$g \cdot r + c$ tört része

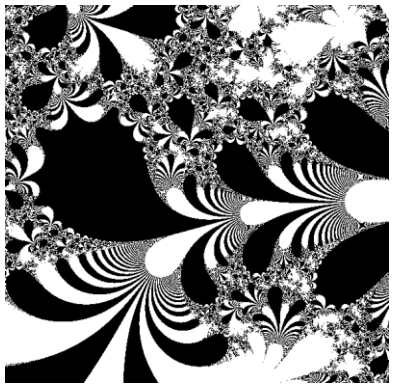
g nagy



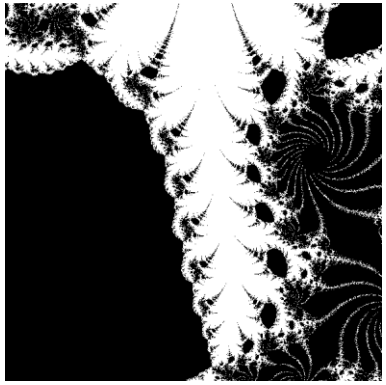
Kaotikus rendszerek a síkon



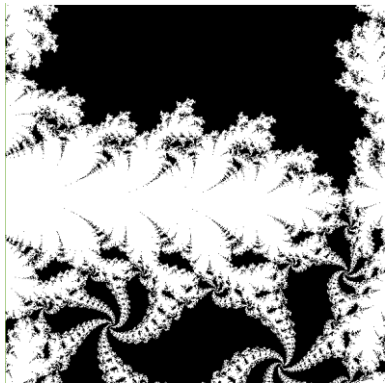
$F(z)=z^2+c$



$F(z)=e^z+c$

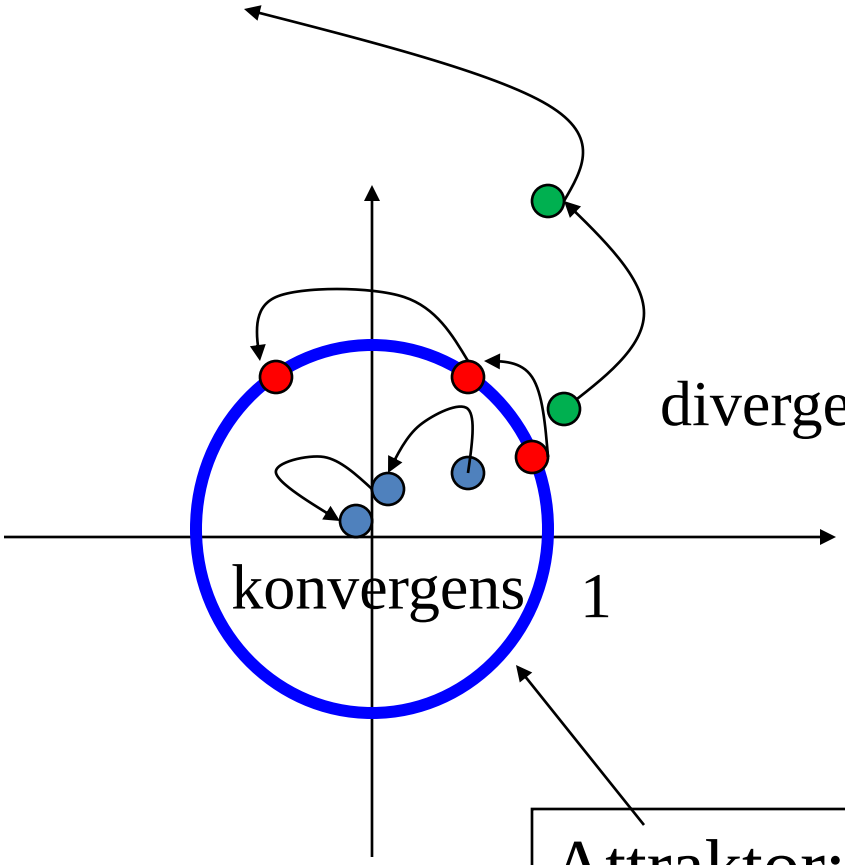


$F(z)=\cos(z+c)$



$F(z)=\sinh(z)+c$

$F: z \to z^2$

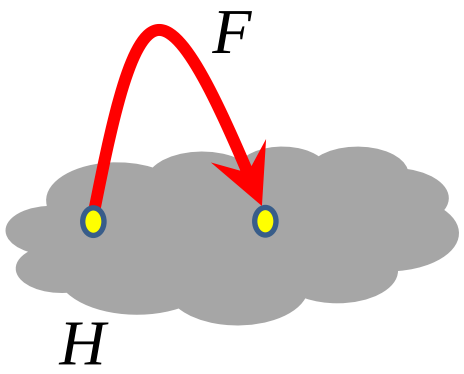


$$z = r\, e^{i\phi}$$

$$r \to r^2$$

$$\phi \to 2\phi$$

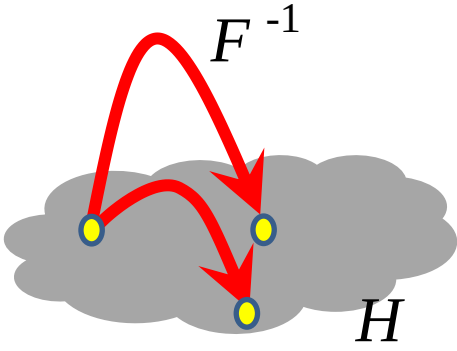
Attraktor: $H = F(H)$



Attraktor felrajzolása

- Attraktor a labilis és a stabil tartomány határa: kitöltött attraktor = nem divergens pontok
  - $z_{n+1} = z_n^2$: ha $z_\infty < \infty$ akkor fekete
- Attraktorhoz konvergálunk, ha az stabil
  - $z_{n+1} = z_n^2$ attraktora labilis

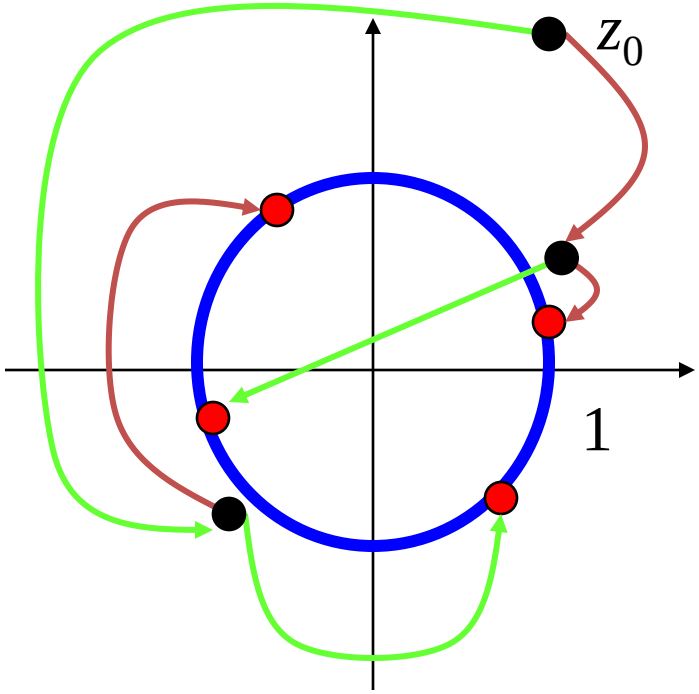
Inverz iterációs módszer



$$H = F(H) \quad \rightarrow \quad H = F^{-1}(H)$$

$$z_{n+1} = z_n^2 \qquad z_{n+1} = \pm\sqrt{z_n}$$

$$r_{n+1} = \sqrt{r_n}$$
$$\phi_{n+1} = \phi_n/2 + \{0|1\}\cdot\pi$$

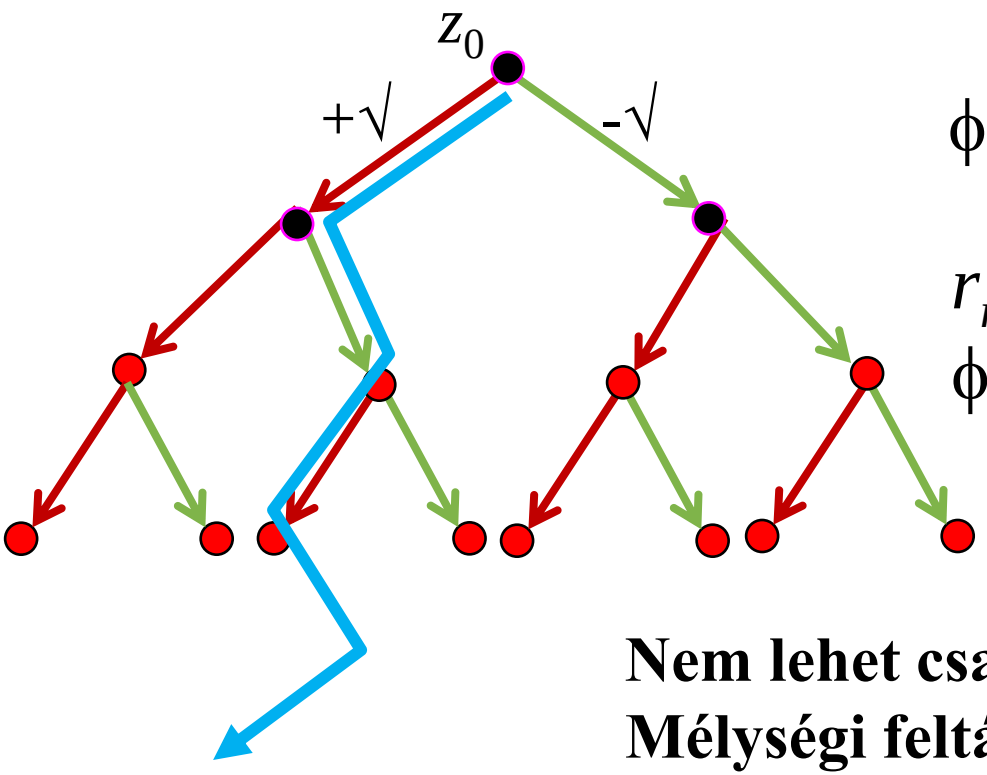


Ha *n* nagy:

$$r_n = (r_0)^{1/2^n} \approx 1$$

$$\phi_n = \phi_0/2^n + \{0|1\}.\{0|1\}\{0|1\}\ldots \cdot\pi$$
$$\approx \{0|1\}.\{0|1\}\{0|1\}\ldots \cdot\pi$$
$$n \quad n\text{-}1 \quad n\text{-}2$$

Többértékű leképzés: Véletlen bolyongás



$$\phi_{n+1} = \phi_n/2 + \{0|1\}\cdot\pi$$

$$r_n \approx 1$$

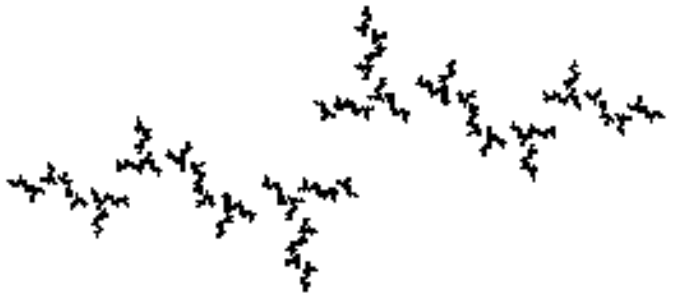
$$\phi_n \approx \underset{n}{\{0|1\}}.\underset{n\text{-}1}{\{0|1\}}\underset{n\text{-}2}{\{0|1\}}\ldots \cdot\pi$$

Nem lehet csak egy értékkel dolgozni ???
Mélységi feltárás szélességi helyett???

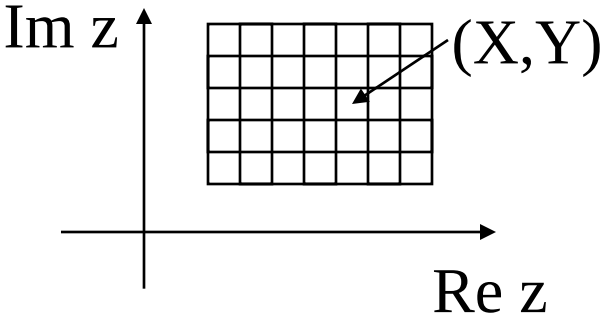
- Csak a **+:** $\phi_n \approx 0.000\ldots \cdot\pi = 0$
- Felváltva +/-: $\phi_{2n} \approx 1.0101\ldots \cdot\pi = 4\pi/3$; $\phi_{2n+1} \approx 0.1010\ldots \cdot\pi = 2\pi/3$
- **Véletlenszerűen!**

(Gaston) Julia halmaz

$F: z \rightarrow z^2 + c$



Kitöltött Julia halmaz: algoritmus



FilledJuliaDraw ( )

FOR Y = 0 TO Ymax DO
FOR X = 0 TO Xmax DO
ViewportWindow(X,Y → x, y)
z = x + *i* y
FOR n = 0 TO "infinity" DO $z = z^2 + c$
IF |z| >= "infinity" THEN WRITE(X,Y, white)
ELSE WRITE(X,Y, black)
ENDFOR
ENDFOR
END

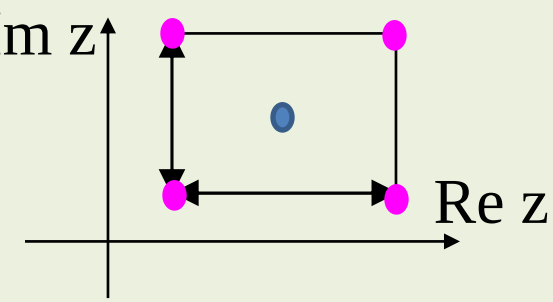
GPU implementáció

CPU program

```
float cVtx[] = { -1, -1,  1, -1,  1, 1,  -1, 1 };

glBufferData(GL_ARRAY_BUFFER, sizeof(cVtx), cVtx, GL_STATIC_DRAW);
…
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

Vertex shader

```
uniform vec2 cameraCenter, cameraSize;

layout(location = 0) in vec2 cVertex;
out vec2 z0;

void main() {
   gl_Position = vec4(cVertex, 0, 1);
   z0 = cVertex * cameraSize/2 + cameraCenter;
}
```

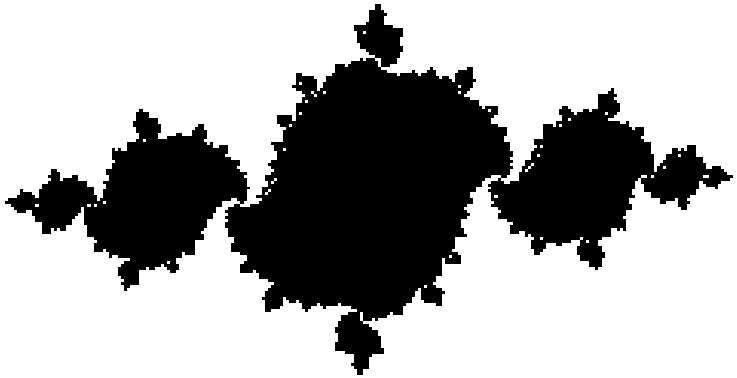


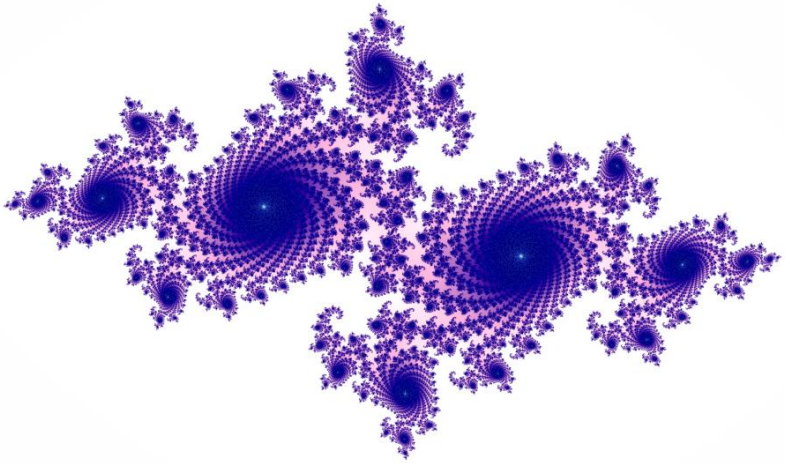
Fragment shader

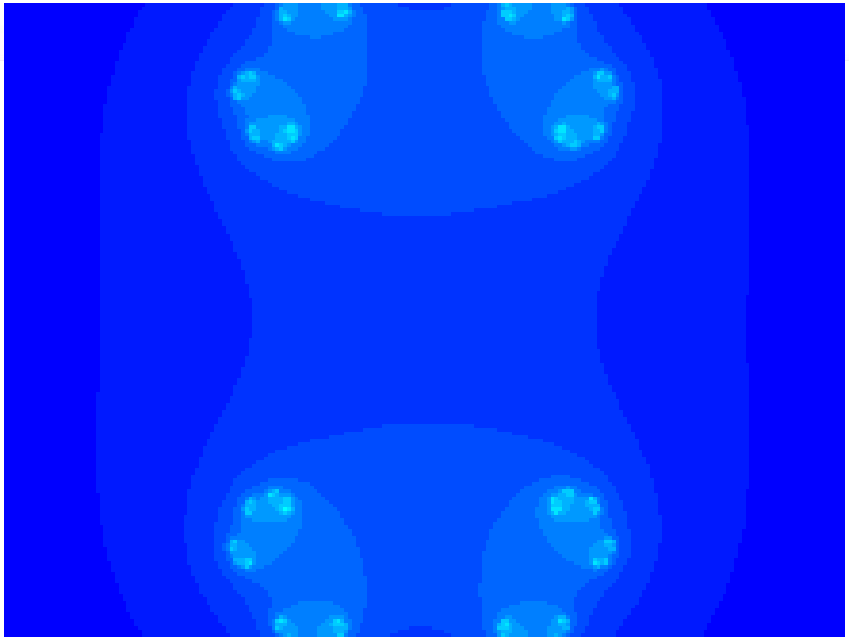
$$z = z^2 + c$$

```
uniform vec2 c;
in vec2 z0;
out vec4 fragCol;

void main() {
   vec2 z = z0;
   for(int i=0; i<1000; i++) z = vec2(z.x*z.x-z.y*z.y,2*z.x*z.y) + c;
   fragCol = (dot(z,z) < 100) ? vec4(0, 0, 0, 1) : vec4(1, 1, 1, 1);
}
```

Kitöltött Julia halmaz: kép







Julia halmaz inverz iterációval

JuliaDrawInverseIterate ( )

Kezdeti z érték választás

FOR n = 0 TO “infinity” DO

x = Re z, y = Im z

IF InWindow(x, y)

WindowViewport(x, y ⇨ X, Y)

WRITE(X,Y, black)

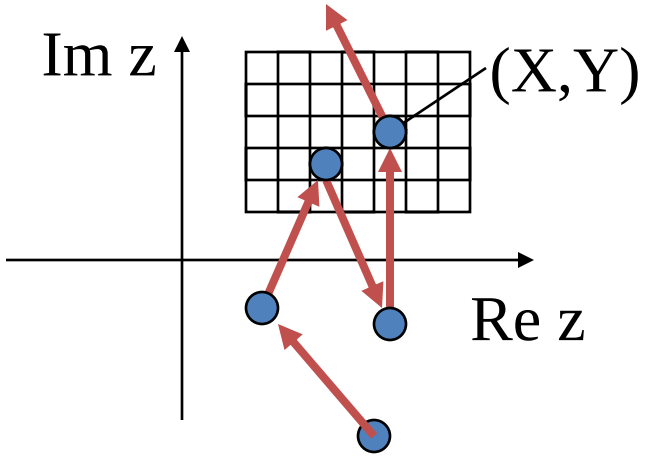
ENDIF

$z = \sqrt{z - c}$

if (random( ) > 0.5) z = -z

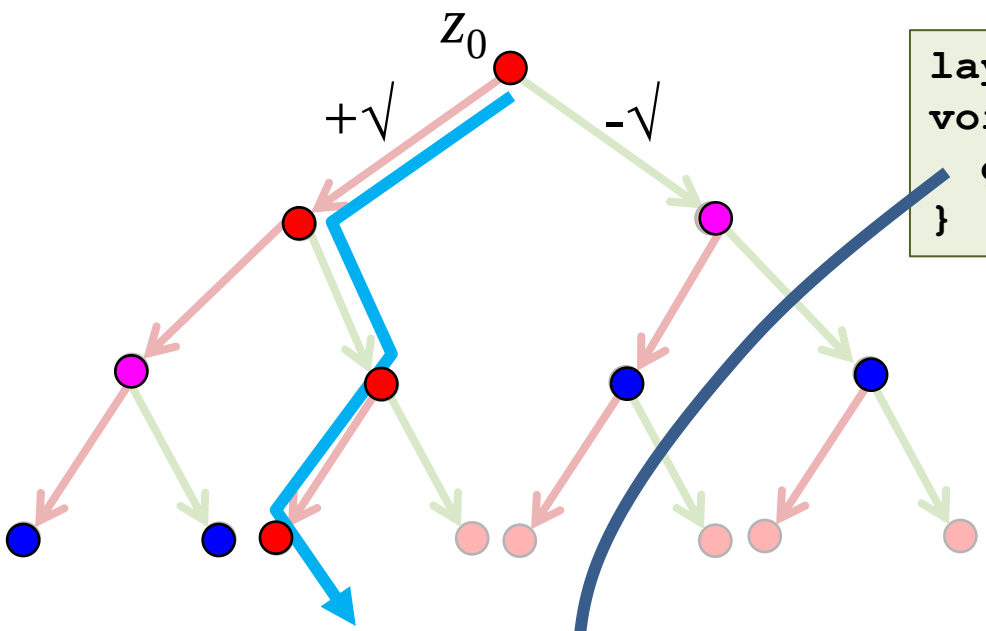
ENDFOR

END



Kezdeti *z* érték:
$z^2 = z - c$ gyöke

GPU implementáció

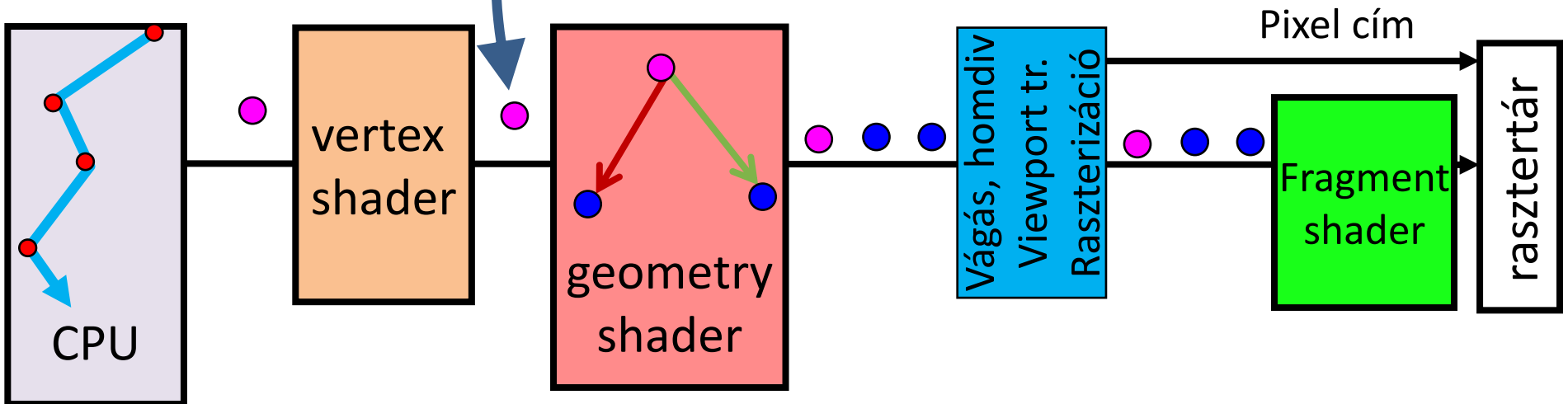


Vertex shader

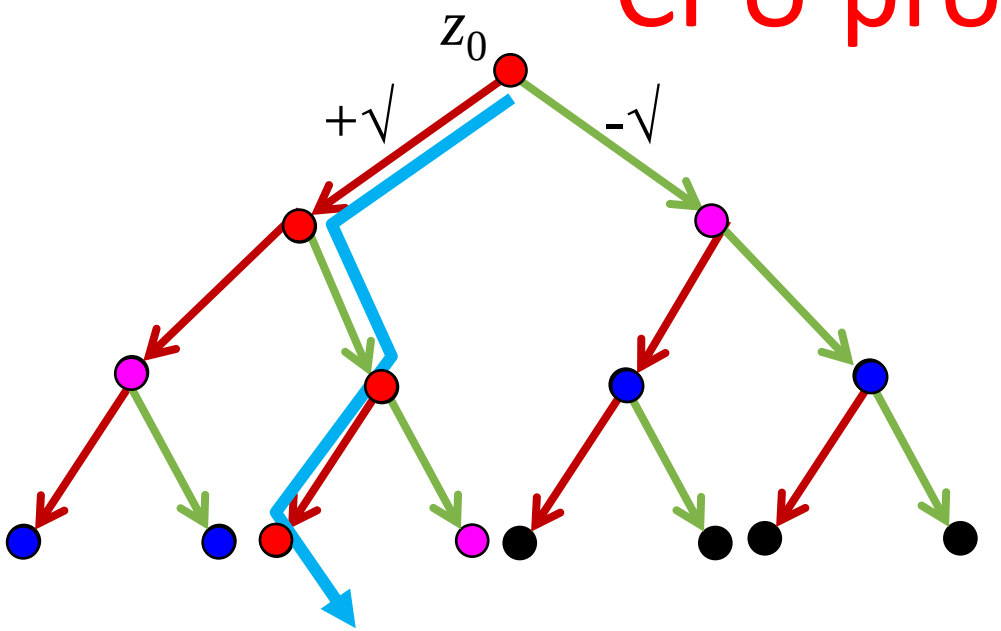
```
layout(location = 0) in vec2 zRoot;
void main() {
  gl_Position = vec4(zRoot, 0, 1);
}
```

Fragment shader

```
void main() {
  fragColor = vec4(0, 0, 0, 1);
}
```



CPU program

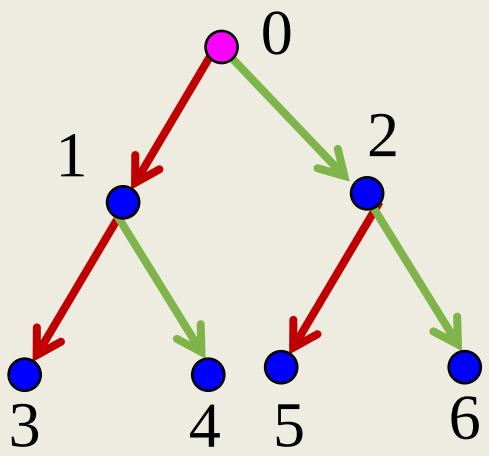


Kezdeti *z* érték:
$z^2 = z - c$ gyöke

```
vec2 z = vec2(0.5, 0) + sqrtComplex(vec2(0.25 - c.x, -c.y));
for (int p = 0; p < nPackets; p++) {
   vec2 vtx[nSeeds];
   for (int i = 0; i < nSeeds; i++) {
       z = sqrtComplex(z - c) * (rand() & 1 ? 1 : -1);
       vtx[i] = -z;
   }
   glBufferData(GL_ARRAY_BUFFER, sizeof(vtx), vtx, GL_DYNAMIC_DRAW);
   glDrawArrays(GL_POINTS, 0, nSeeds);
}
```

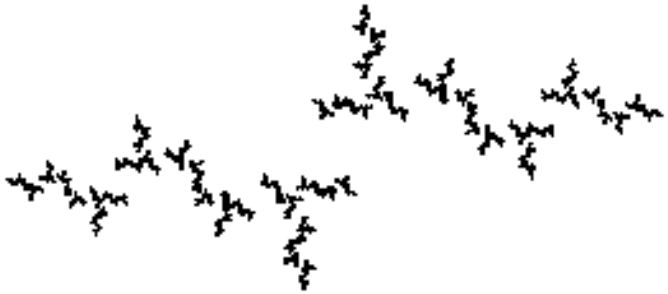
Geometry shader

```
uniform vec2 cameraCenter, cameraSize, c;

layout(points) in;
layout(points, max_vertices = 63) out;

vec2 sqrtComplex(vec2 z) {
    float r = length(z), phi = atan(z.y, z.x);
    return vec2(cos(phi/2), sin(phi/2)) * sqrt(r);
}

void main() {
    vec2 zs[63];
    zs[0] = gl_in[0].gl_Position.xy;
    gl_Position = vec4((zs[0]-cameraCenter)/(cameraSize/2),0,1);
    EmitVertex();

    for(int i = 0; i < 63/2; i++) {
        vec2 z = sqrtComplex(zs[i] - c);
        for(int j = 1; j <= 2; j++) {
            zs[2 * i + j] = z;
            gl_Position = vec4((z-cameraCenter)/(cameraSize/2),0,1);
            EmitVertex();
            z = -z;
        }
    }
    EndPrimitive();
}
```



Julia halmaz összefüggése

összefüggő

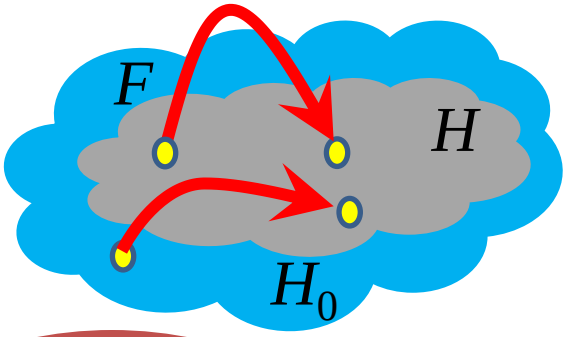


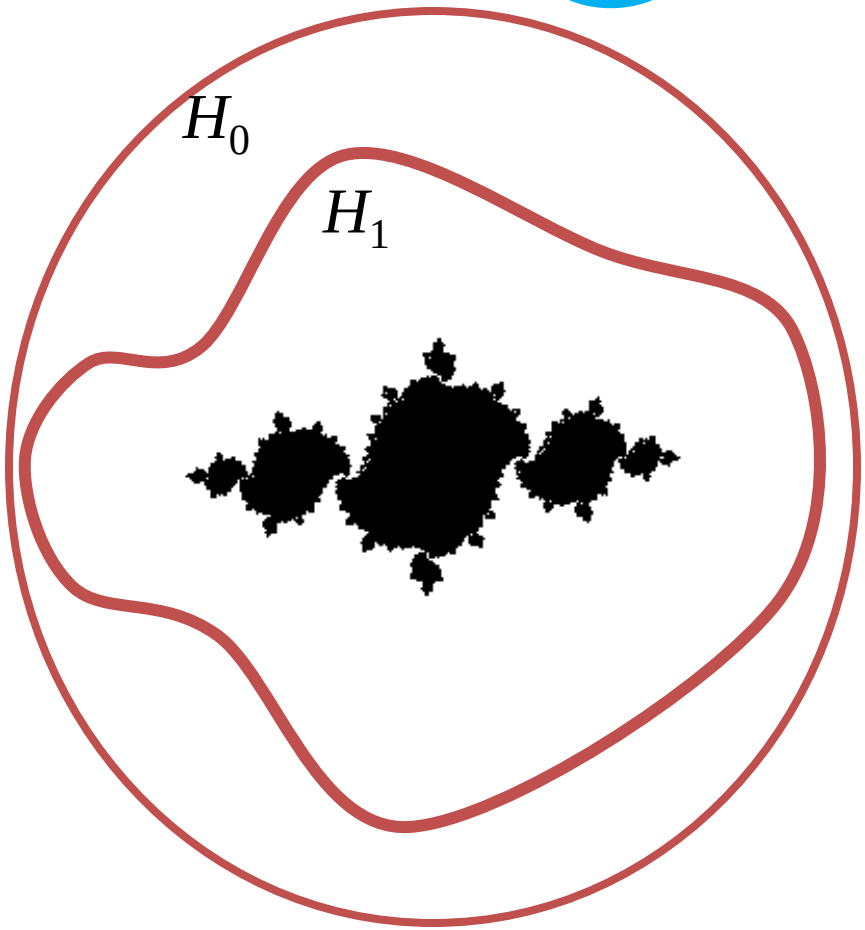
nem összefüggő,
Cantor féle halmaz

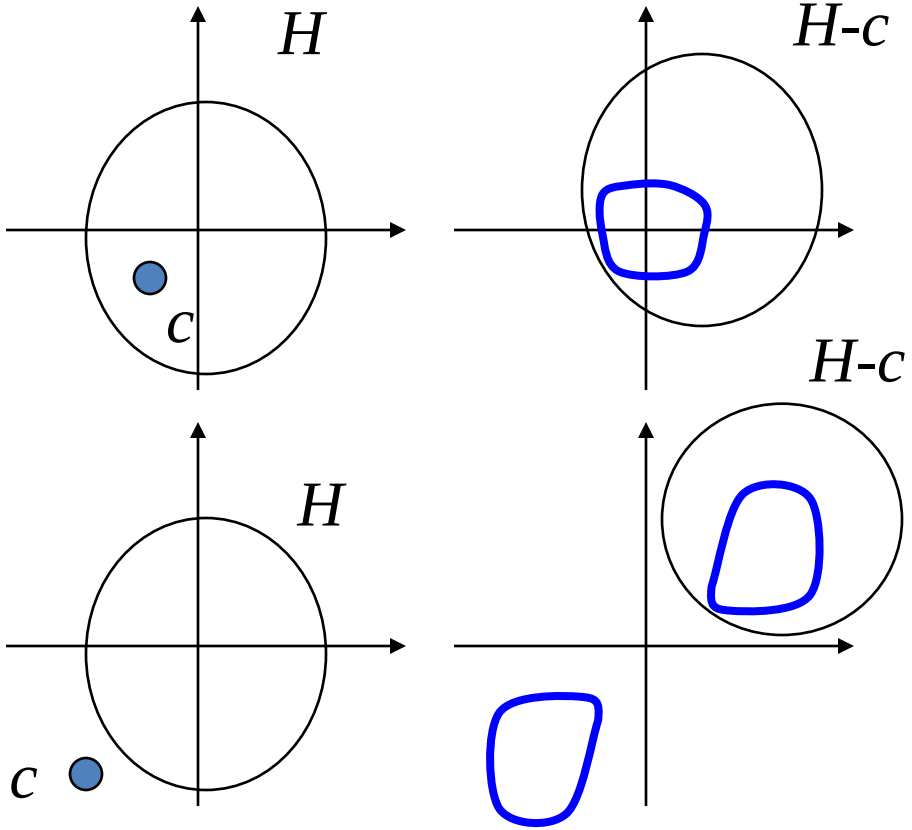
Julia halmaz összefüggősége

$H = F(H)$

$H_1 = F(H_0)$



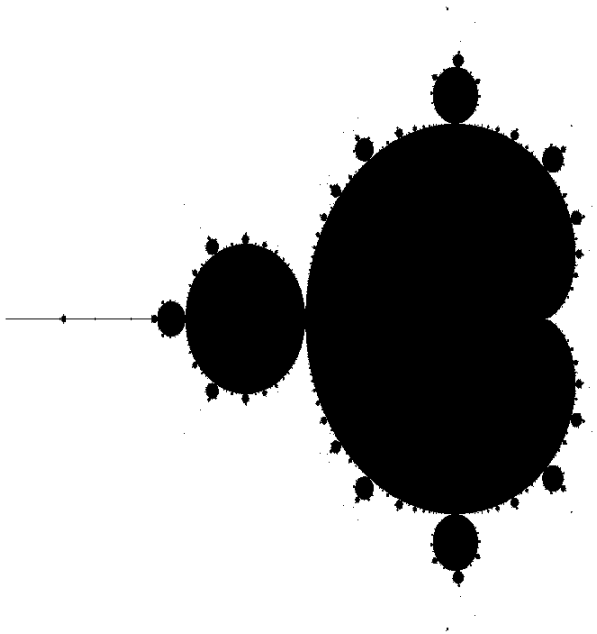


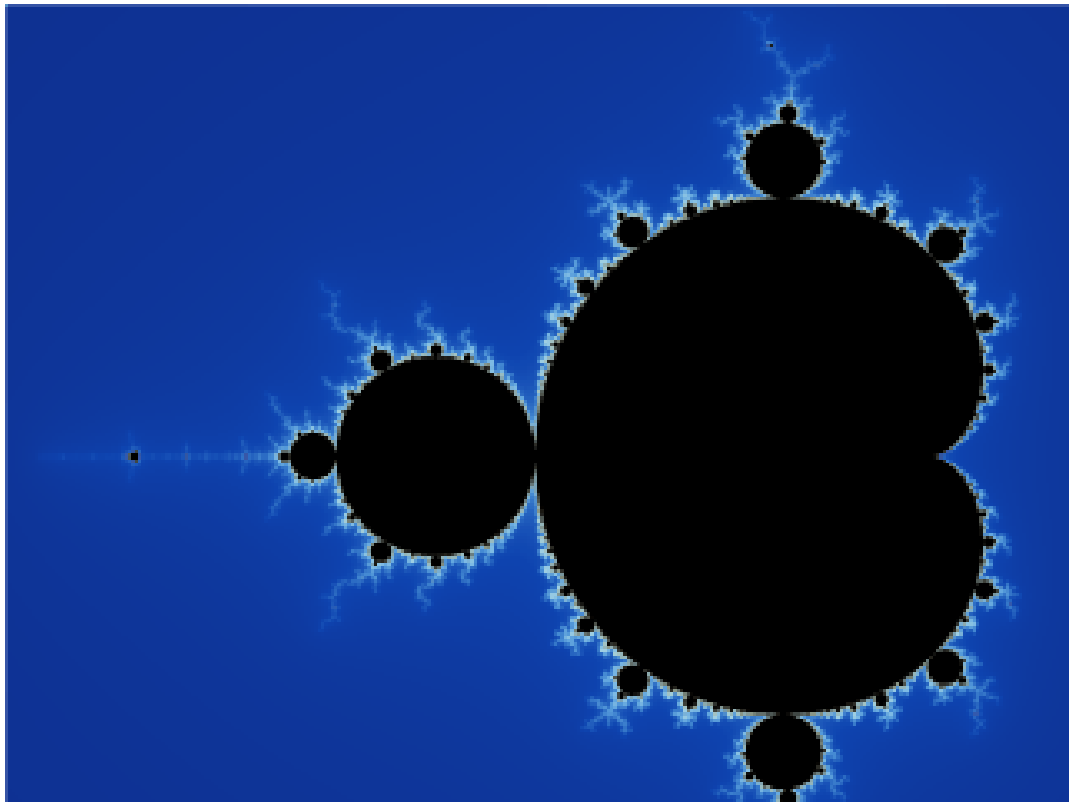


$$z_{n+1} = \pm \sqrt{z_n - c}$$

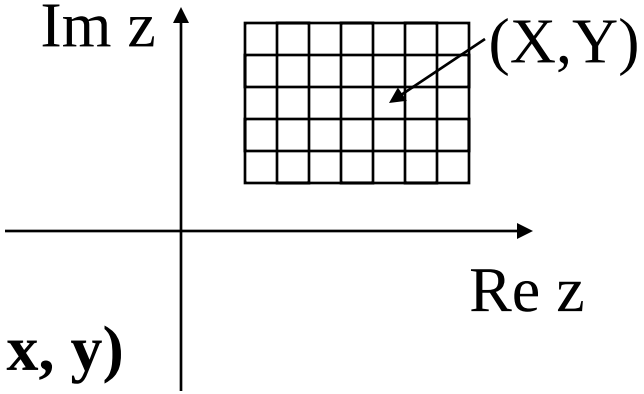
(Benoit) Mandelbrot halmaz

Azon c komplex számok, amelyekre a $z \to z^2 + c$ Julia halmaza összefüggő.





Mandelbrot halmaz, algoritmus



MandelbrotDraw ( )

```
    FOR Y = 0 TO Ymax DO
        FOR X = 0 TO Xmax DO
            ViewportWindow(X,Y → x, y)
            c = x + i y
            z = 0
            FOR n = 0 TO "infinity" DO z = z^2 + c
            IF |z| >= "infinity"  THEN  WRITE(X,Y, white)
            ELSE                        WRITE(X,Y, black)
        ENDFOR
    ENDFOR
END
```

|z| > 2

GPU implementáció

CPU program

```
float cVtx[] = { -1, -1,  1, -1,   1, 1,  -1, 1 };
glBufferData(GL_ARRAY_BUFFER, sizeof(cVtx), cVtx, GL_STATIC_DRAW);
…
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

Vertex shader

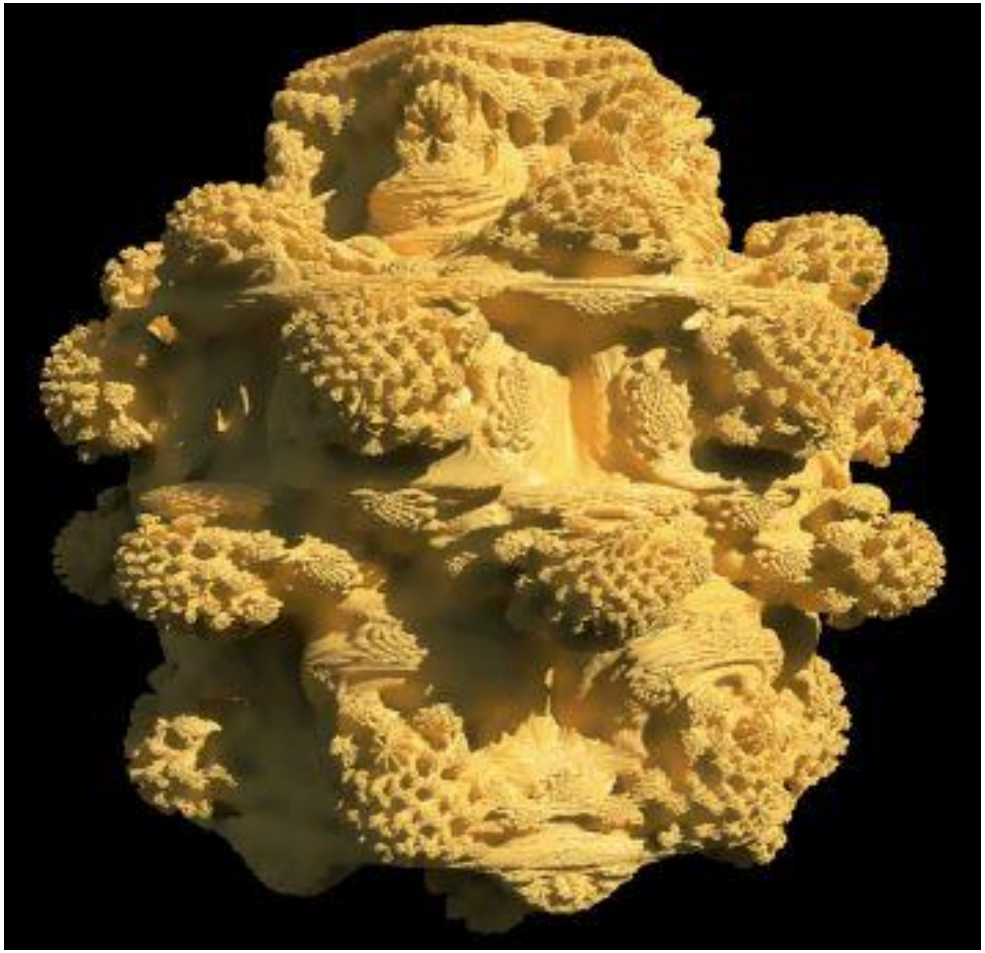
```
uniform vec2 cameraCenter, cameraSize;

layout(location = 0) in vec2 cVertex;
out vec2 c;

void main() {
   gl_Position = vec4(cVertex, 0, 1);
   c = cVertex * cameraSize/2 + cameraCenter;
}
```

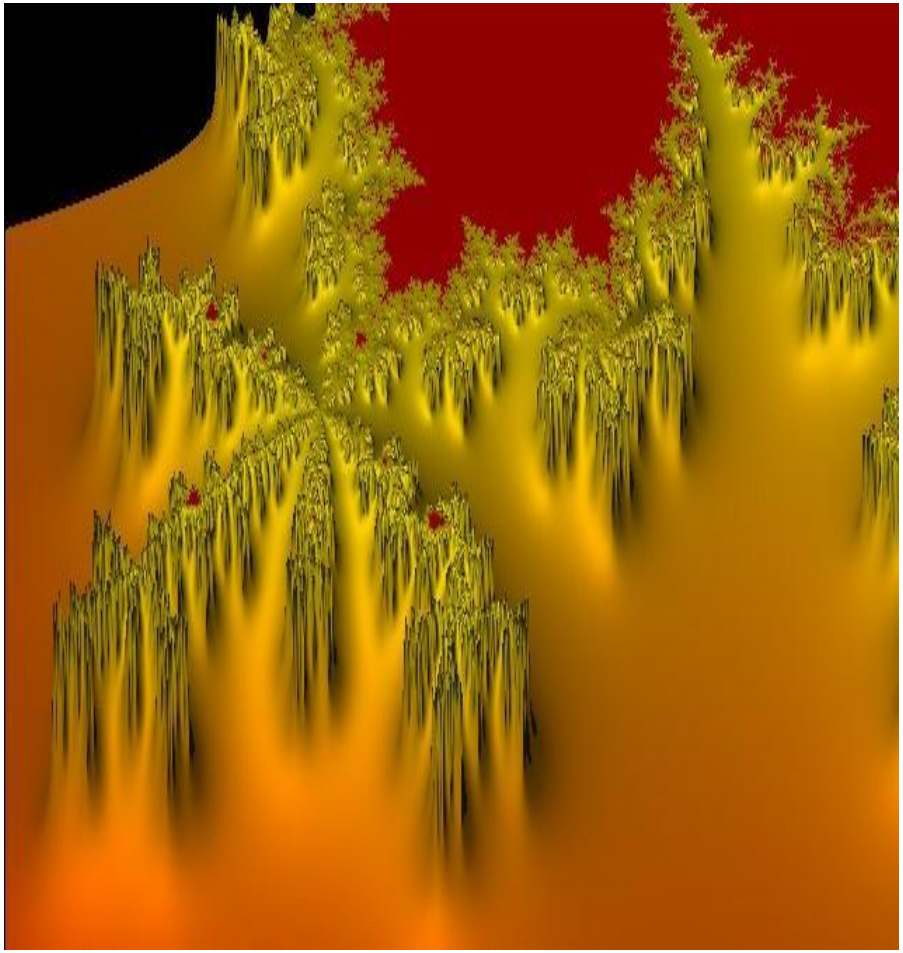
Fragment shader

$$z = z^2 + c$$

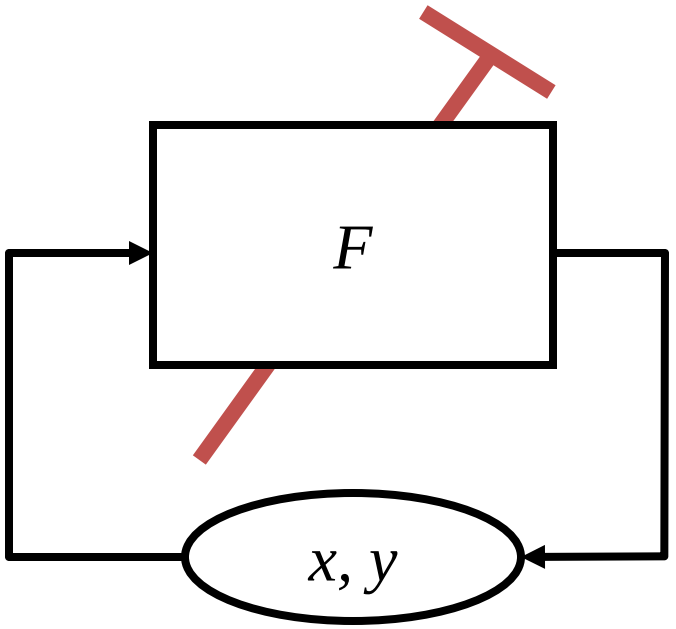
```
in vec2 c;
out vec4 fragCol;

void main() {
  vec2 z = c;
  for(int i = 0; i < nIteration; i++) {
     z = vec2(z.x * z.x - z.y * z.y, 2 * z.x * z.y) + c;
     if (dot(z, z) > 4) break;
  }
  fragCol = (i == nIteration) ? vec4(0, 0, 0, 1) : vec4(1, 1, 1, 1);
}
```

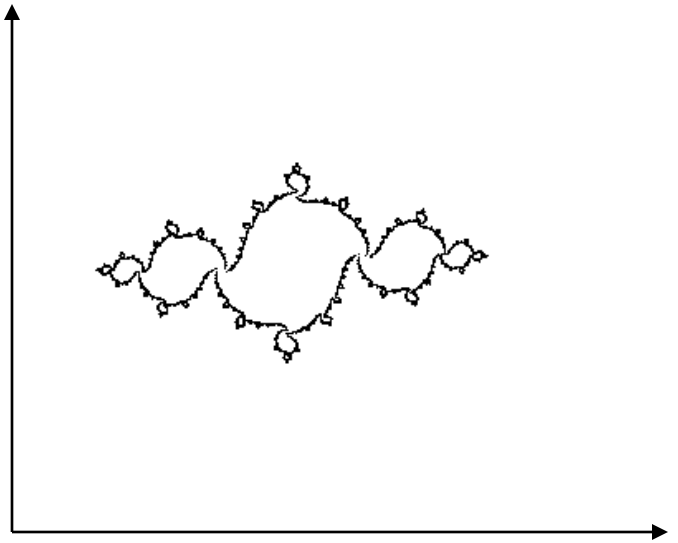
„Matematikát kijátszó” Mandelbrot halmazok





Inverz feladat: IFS modellezés





$H \Rightarrow F$

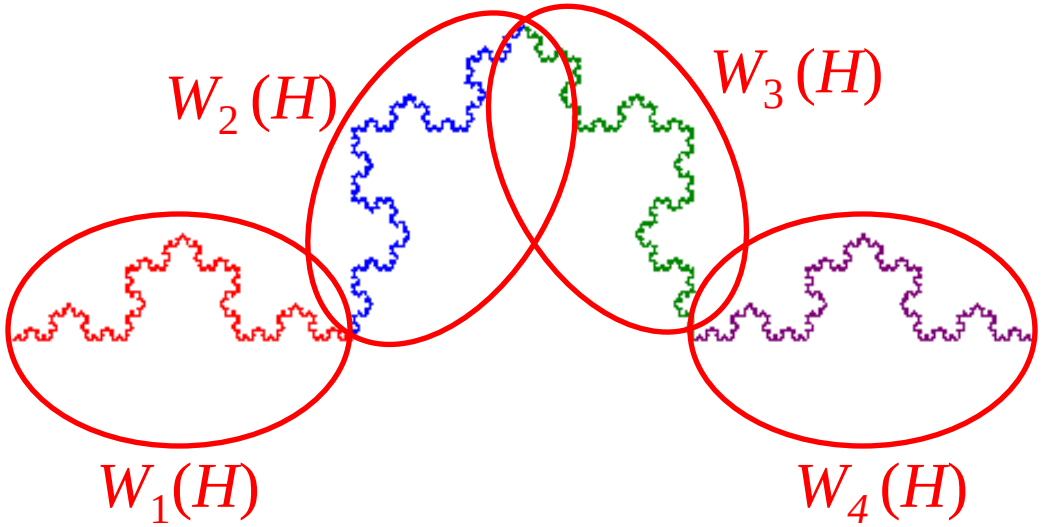
Attraktor:

$H = F(H)$

F: szabadon vezérelhető, legyen stabil attraktora

Melyik függvény attraktora a Koch görbe?

Attraktor: $H = F(H) = W_1(H) \cup W_2(H) \cup W_3(H) \cup W_4(H)$

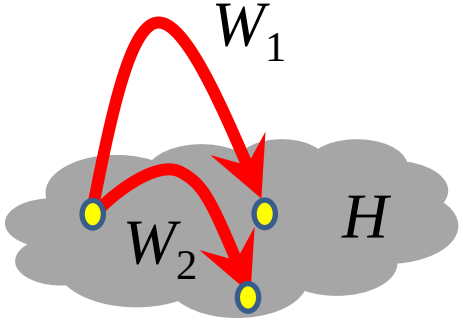


$$W_k(x,y) = [x,y] \cdot \boldsymbol{A}_k + \boldsymbol{q}_k$$

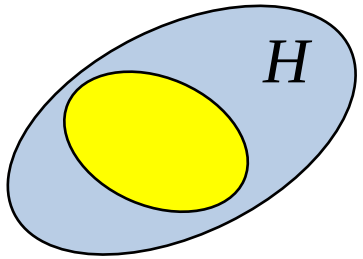
F: többértékű lineáris leképzés

Nem feltétlenül hasonlósági transzformáció!
Nem csak önhasonló objektumok.

$$F = W_1 \vee W_2 \vee \ldots \vee W_m$$

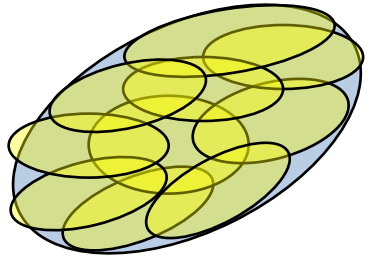


$$W_k(x,y) = [x,y]\cdot \boldsymbol{A}_k + \boldsymbol{q}_k$$



Stabilitás = kontrakció
$|\boldsymbol{A}$ sajátértékei$| < 1$

$$H = W_1(H) \cup W_2(H) \cup \ldots \cup W_m(H)$$



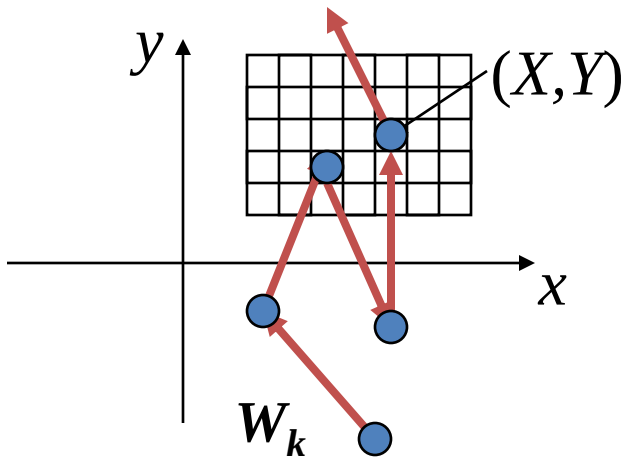
$$H = F(H)$$

Kollázs:
lefedés a kicsinyített változatokkal
Nem feltétlenül diszkjunkt

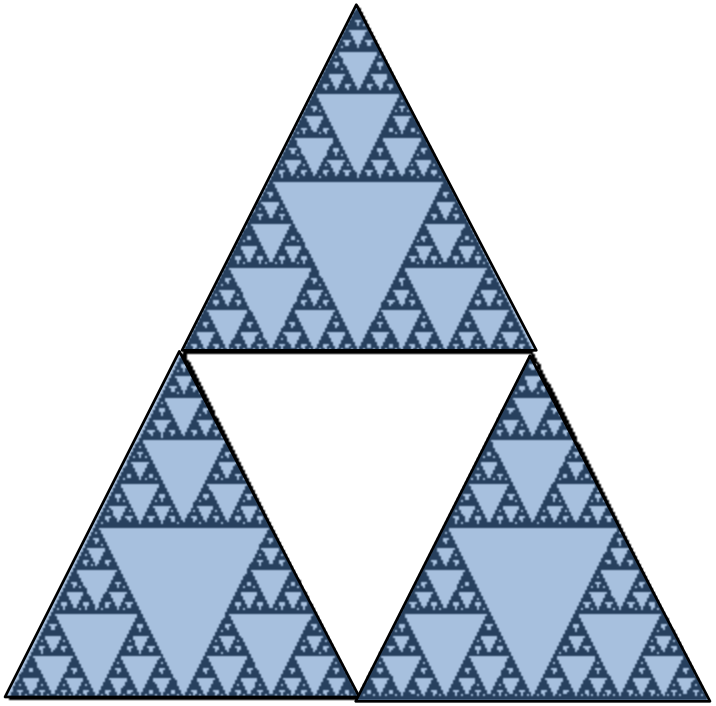
IFS rajzolás: iterációs algoritmus

IFSDraw ( )

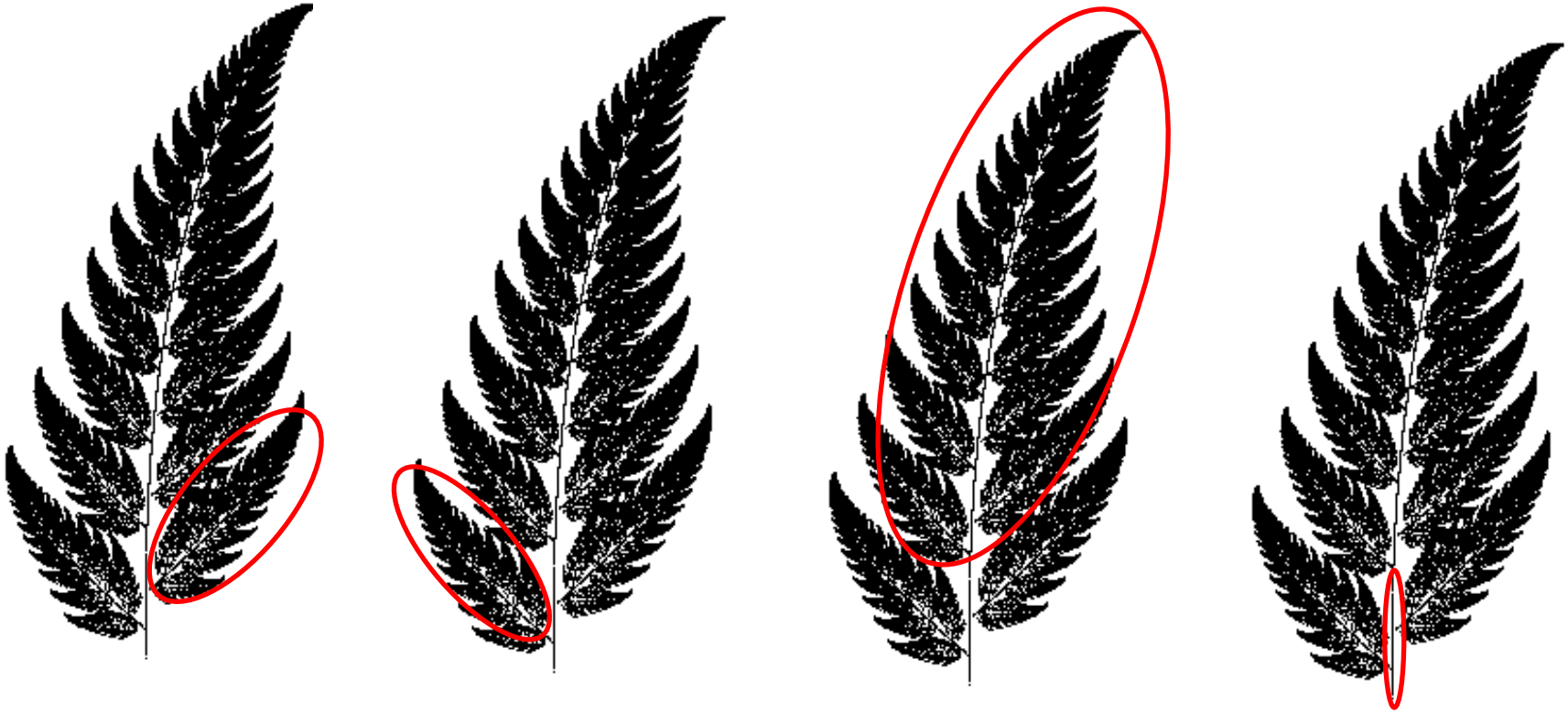
Legyen [x,y] = [x,y] A_1 + q_1 megoldása a kezdő [x,y]
FOR i = 0 TO "infinity" DO
IF InWindow(x, y)
WindowViewport(x, y ⇨ X, Y)
Write(X, Y, color);
ENDIF
Válassz k-t p_k valószínűséggel
[x,y] = [x,y] A_k + q_k
ENDFOR
END



Egyszerű IFS-ek



IFS modellezés



IFS képek