Procedural Methods

CS 432 Interactive Computer Graphics
Prof. David E. Breen
Department of Computer Science

Shader Applications

- Moving vertices
  - Wave motion
  - Morphing
  - Particle systems
  - Newtonian dynamics
  - Fractals
- Lighting
  - More realistic models
  - Cartoon shaders

Wave Motion Vertex Shader

```
uniform float time;
uniform float xs, zs, // frequencies
uniform float h; // height scale
uniform mat4 ModelView, Projection;
in vec4 vPosition;

void main() {
  vec4 t =vPosition;
  t.y = vPosition.y
     + h*sin(time + xs*vPosition.x)
     + h*sin(time + zs*vPosition.z);
  gl_Position = Projection*ModelView*t;
}
```

Modeling

- Geometric
  - Meshes
  - Hierarchical
  - Curves and Surfaces
- Procedural
  - Particle Systems
  - Fractal

Particle Systems

CS 432 Interactive Computer Graphics

Prof. David E. Breen

Department of Computer Science

Introduction

- Most important of procedural methods
- Used to model
  - Natural phenomena
    - Clouds
    - Terrain
    - Plants
  - Crowd Scenes
  - Real physical processes

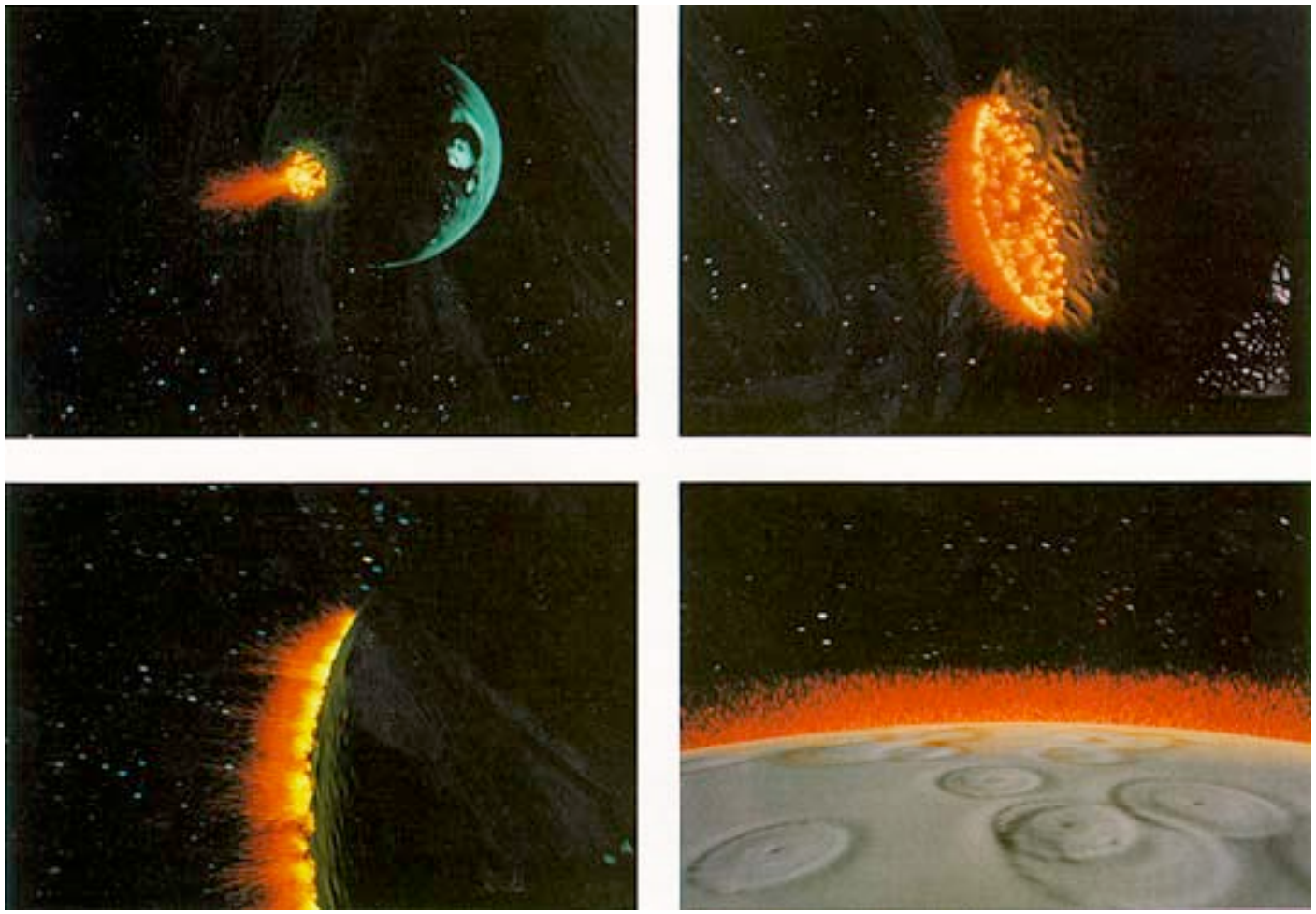
Particle Systems – A Technique for Modeling a Class of Fuzzy Objects

William T. Reeves, Lucasfilm
ACM Transactions on Graphics, 1983

Presented in CS536 by Walt Mankowski
19 October 2006

Drexel
UNIVERSITY

Genesis Project Demo



Prior to Khan

State of the art in computer animation:

- Static geometric objects
- Primitive surface elements (e.g. polygons)
- Deterministic
- Simple affine transformations

Prior to Khan

State of the art in computer animation:

- Smooth surfaces
- Well-defined
- Shiny surfaces

What about “fuzzy” objects?

- Clouds, smoke, water, fire
- Irregular, ill-defined, complex
- Dynamic
- Fluid

Representation of "fuzzy" objects

- Cloud of primitive particles that define its volume
- Particles are born, move, change form, and die over time
- Stochastic processes are basis for all dynamics

Advantages of particles

- Simpler than polygons
- Easier and faster to compute
- Easier to motion-blur
- Model is procedural and random
- Less human modeling required
- Particle systems are “alive”
- More natural way to represent dynamic motion than polygons

High level view

- System is collection of many of tiny particles (25,000–750,00 in sample frames)
- Over time, particles are added to the system, move and change, and then are removed from the system

Drexel UNIVERSITY

For each frame...

- New particles are “born” into the system
- Each new particle is assigned its own individual, random attributes
- Any particles that are too old “die” and are removed from the system
- Remaining particles move and are transformed according to their attributes
- Image is rendered to frame buffer

Particle generation

- For entire frame
  - $NParts_f = MeanParts_f + Rand() * VarParts_f$
- For smaller screen area
  - $NParts_f = (MeanParts_{saf} + Rand() * VarParts_{saf}) * ScreenArea$
- Uniformly distributed over a range
- Dynamically change density →Vary mean over all frames
  - $MeanParts_f = InitMeanParts + DeltaMeanParts * (f - f_0)$

Particle attributes

For each new particle, must determine:

- initial position
- initial velocity
- initial size
- initial color
- initial transparency
- shape
- lifetime

Initial position

- sphere
- circle
- rectangle
- etc.

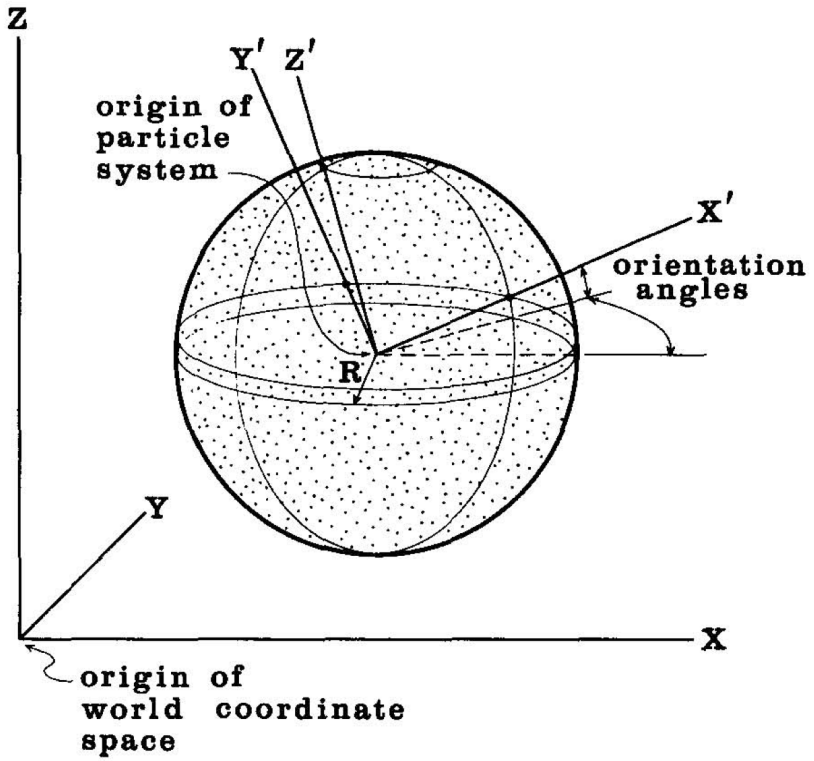


Fig. 1. Typical particle system with spherical generation shape.

Initial settings

- Initial speed is uniformly distributed:
  - InitialSpeed = MeanSpeed + Rand() * VarSpeed
- Color, size, transparency set the same way

Particle dynamics

- To move a particle, just add its velocity vector to its position.
- Can simulate gravity by using an acceleration factor.
- Color, transparency, size controlled by similar rate-of-change parameters.
- Everything is **completely deterministic**.

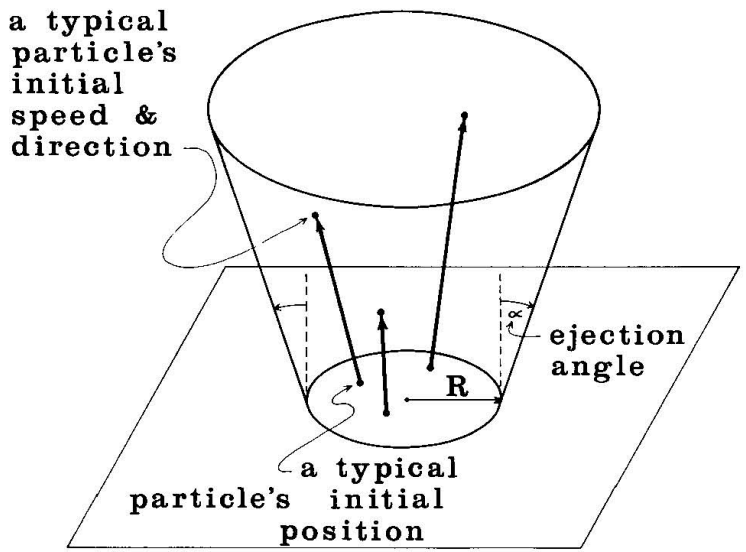


Fig. 3. Form of an explosion-like particle system.

Particle extinction

- Remove particles that exceed lifetime
- Can also remove particles that stray outside a window

Particle rendering

Two simplifying assumptions:

- Particles don't interact with other objects
  - They're composited in later
  - Added "glow" on planet from the fire with an additional light source
- Each particle is a point light source
  - No hidden surfaces
  - Each particle only contributes to the color of the pixel it covers

Particle hierarchy

- Maintain a tree of particle systems.
- Allows more global control over system.
- In Genesis Demo:
  - Top level was point of impact
  - Particles it generates are themselves other particle systems.

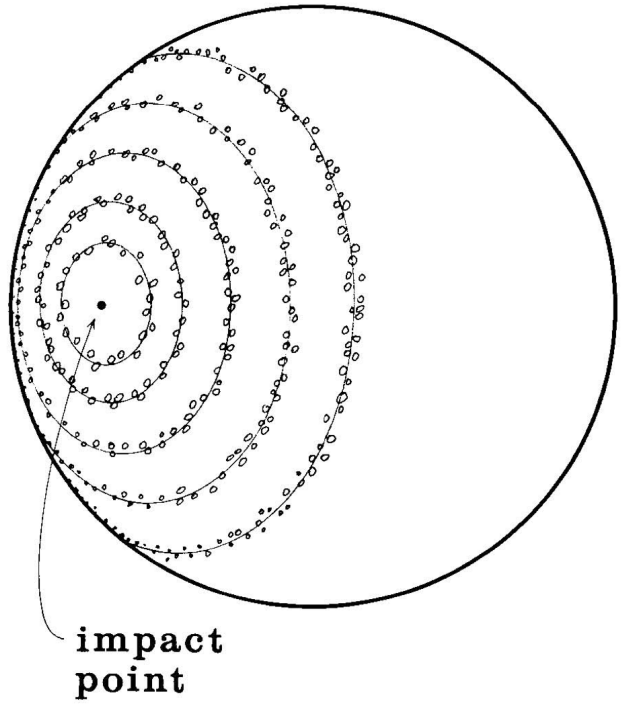


Fig. 2. Distribution of particle systems on the planet's surface.

Motion blur

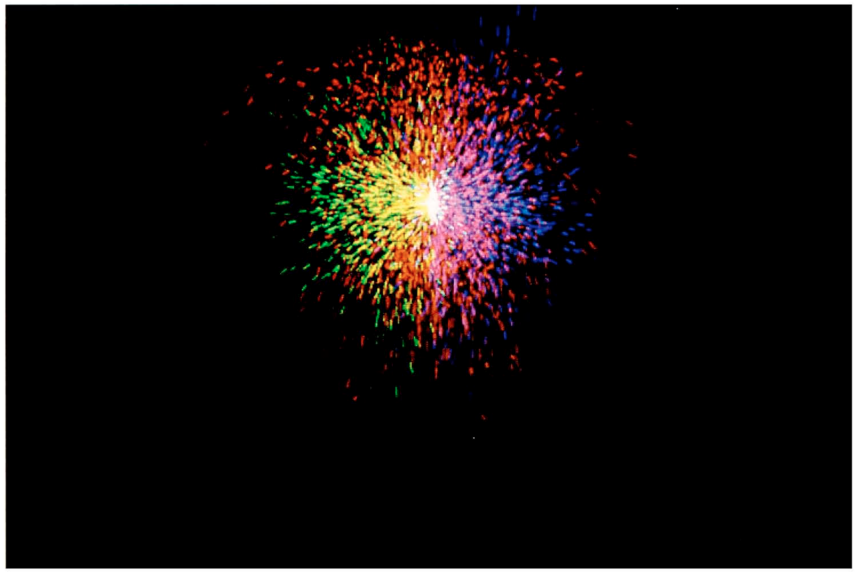
- Computer animation up to then produced one static image at an instant in time
- This can lead to aliasing and strobing
- They motion-blurred the particles:
  - Calculate position at start of frame and half way through
  - Draw an antialiased line between points
- Stills look blurry but video looks better

- Go to videos

-Making of the Genesis Sequence from Star Trek II

- Particle Dreams
  - Karl Sims

Other applications



Fireworks

Grass

Future directions (circa 1982)

- Clouds
  - difficult to model
  - can throw shadows on themselves
  - very large number of particles
- Water
- More sophisticated motion-blur

Newtonian Particle

- Particle system is a set of particles
- Each particle is an ideal point mass
- Six degrees of freedom
  - Position
  - Velocity
- Each particle obeys Newtons' law

$$f = ma$$

Particle Equations

$\mathbf{p}_i = (x_i, y_i\ z_i)$

$\mathbf{v}_i = d\mathbf{p}_i / dt = \mathbf{p}_i' = (dx_i / dt, dy_i / dt\ , z_i / dt)$

$m\ \mathbf{v}_i' = \mathbf{f}_i$

Hard part is defining force vector

Force Vector

- Independent Particles
  - Gravity
  - Wind forces
  - O(n) calulation
- Coupled Particles O(n)
  - Meshes
  - Spring-Mass Systems
- Coupled Particles $O(n^2)$
  - Attractive and repulsive forces

Solution of Particle Systems

```
float time, delta state[6n], force[3n];
state = initial_state();
for(time = t0; time<final_time;
                              time+=delta) {
  force = force_function(state, time);
  state = ode(force, state, time,
                                   delta);
  render(state, time)
}
```

Simple Forces

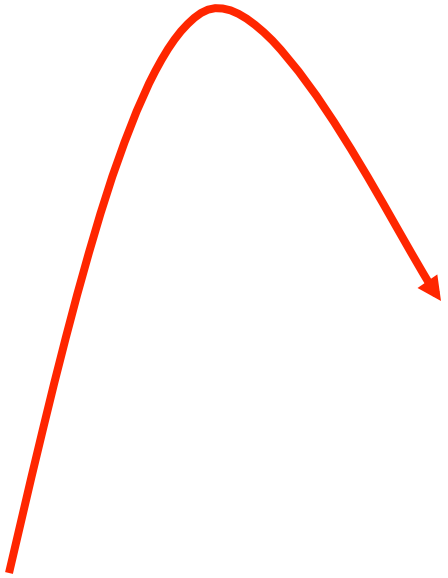
- Consider force on particle i

$$\mathbf{f}_i = \mathbf{f}_i(\mathbf{p}_i, \mathbf{v}_i)$$

- Gravity $\mathbf{f}_i = \mathbf{g}$

$$\mathbf{g}_i = (0, -g, 0)$$

- Wind forces
- Drag
  - $\mathbf{f}(-\mathbf{v}_i)$



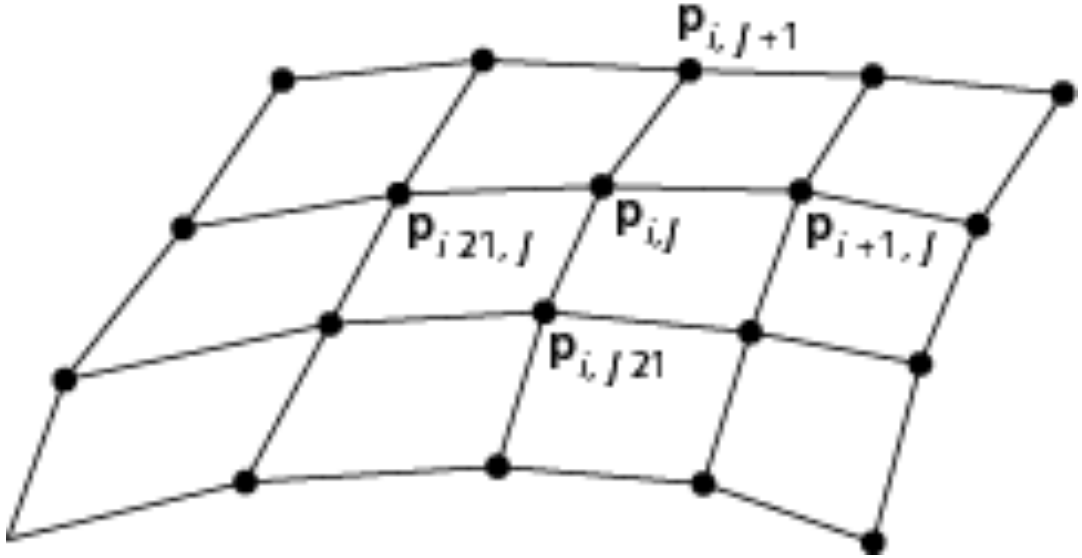
$\mathbf{p}_i(t_0), \mathbf{v}_i(t_0)$

Particle System

```
uniform vec3 init_vel;
uniform float g, m, t;
uniform mat4 Projection, ModelView;
in vPosition;
void main(){
  vec3 object_pos;
  object_pos.x = vPosition.x + vel.x*t;
  object_pos.y = vPosition.y + vel.y*t
                             - g/(2.0*m)*t*t;
  object_pos.z = vPosition.z + vel.z*t;
  gl_Position = Projection*
            ModelView*vec4(object_pos,1);
}
```

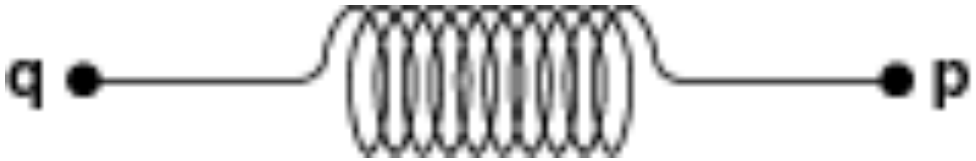
Meshes

- Connect each particle to its closest neighbors
  - O(n) force calculation
- Use spring-mass system



Spring Forces

- Assume each particle has unit mass and is connected to its neighbor(s) by a spring
- Hooke's law: force proportional to distance ($d = ||\mathbf{p} - \mathbf{q}||$) between the points



Hooke's Law

- Let s be the distance when there is no force

$$\mathbf{f} = -k_s(|\mathbf{d}| - s)\,\mathbf{d}/|\mathbf{d}|$$

k_s is the spring constant

$\mathbf{d}/|\mathbf{d}|$ is a unit vector pointed from $\mathbf{p}$ to $\mathbf{q}$

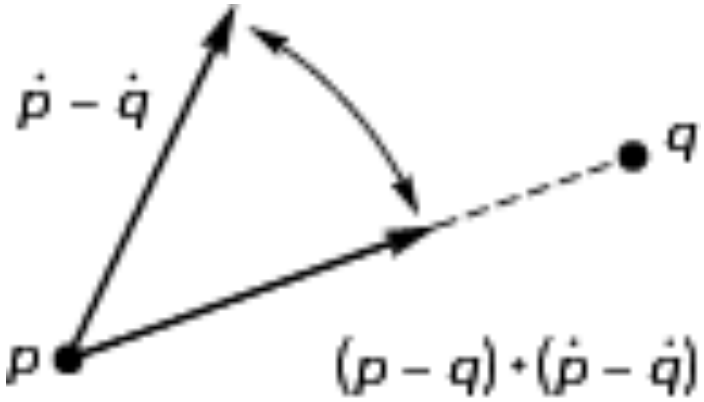
- Each interior point in mesh has four forces applied to it

Spring Damping

- A pure spring-mass will oscillate forever
- Must add a damping term

$$\mathbf{f} = -(k_s(|\mathbf{d}| - s) + k_d \dot{\mathbf{d}} \cdot \mathbf{d}/|\mathbf{d}|)\mathbf{d}/|\mathbf{d}|$$

- Must project velocity



Repulsion

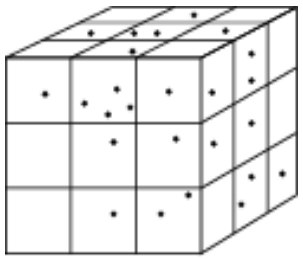
- Inverse square law

$$\mathbf{f} = k_r \mathbf{d} / |\mathbf{d}|^2$$

- General case requires $O(n^2)$ calculation
- In most problems, the drop off is such that not many particles contribute to the forces on any given particle
- Sorting problem: is it O(n log n)?

Boxes

- Spatial subdivision technique
- Divide space into boxes
- Particle can only interact with particles in its box or the neighboring boxes
- Must update which box a particle belongs to after each time step

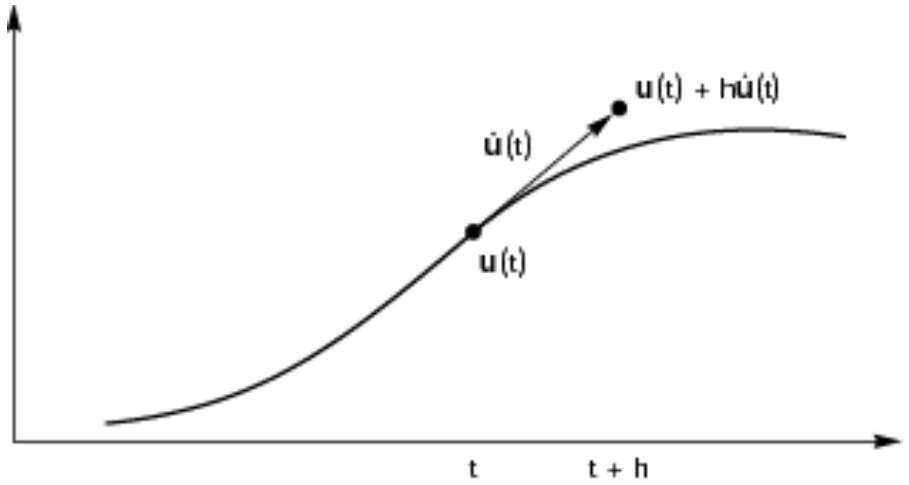


Linked Lists

- Each particle maintains a linked list of its neighbors
- Update data structure at each time step
- Must amortize cost of building the data structures initially

Solution of ODEs

- Particle system has 6n ordinary differential equations
- Write set as $d\mathbf{u}/dt = g(\mathbf{u},t)$
- Solve by approximations using Taylor's Thm



Euler's Method

$$\mathbf{u}(t + h) \approx \mathbf{u}(t) + h\, d\mathbf{u}/dt = \mathbf{u}(t) + h\mathbf{g}(\mathbf{u}, t)$$

Per step error is $O(h^2)$

Require one force evaluation per time step

Problem is numerical instability

Depends on step size

Improved Euler

$$\mathbf{u}(t + h) \approx \mathbf{u}(t) + h/2(\mathbf{g}(\mathbf{u}, t) + \mathbf{g}(\mathbf{u}, t+h))$$

Per step error is $O(h^3)$

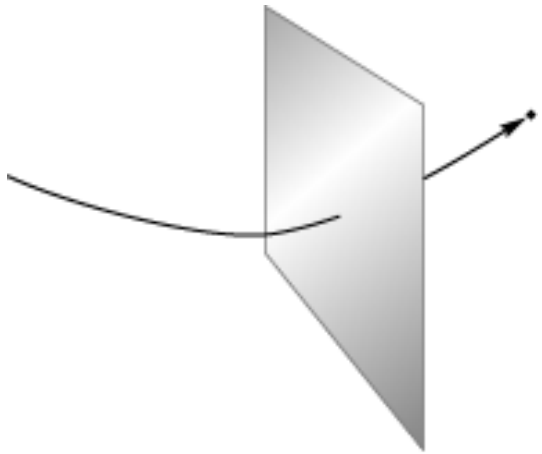
Also allows for larger step sizes

But requires two function evaluations per step

Also known as Runge-Kutta method of order 2

Contraints

- Easy in computer graphics to ignore physical reality
- Surfaces are virtual
- Must detect collisions separately if we want exact solution
- Can approximate with repulsive forces



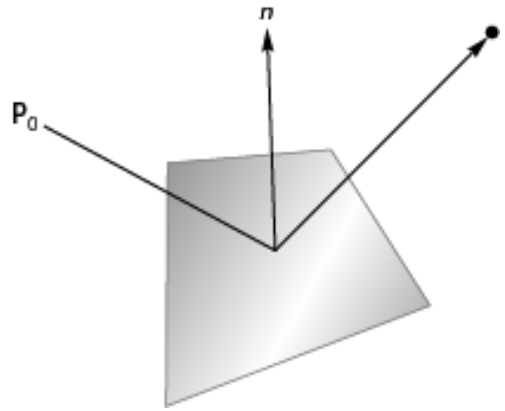
Collisions

Once we detect a collision, we can calculate new path

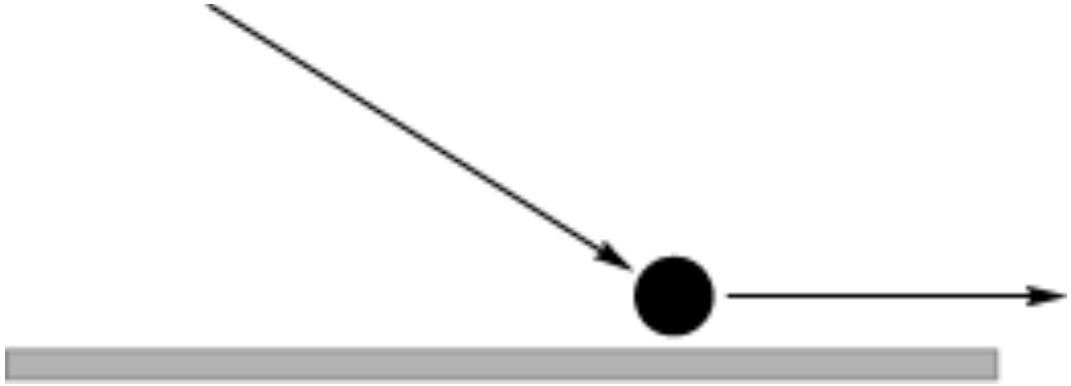
Use coefficient of resititution

Reflect vertical component

May have to use partial time step



Contact Interactions



- Sliding contact
- Normal force that prevents particle from penetrating surface
- Friction force in the tangential direction

Drexel
UNIVERSITY

Flocks, Herds, and Schools: A Distributed Behavioral Model

Craig W. Reynolds

SIGGRAPH 1987

Presented by Duc Nguyen

November 15, 2007

Simulated Flocks of "Boids"

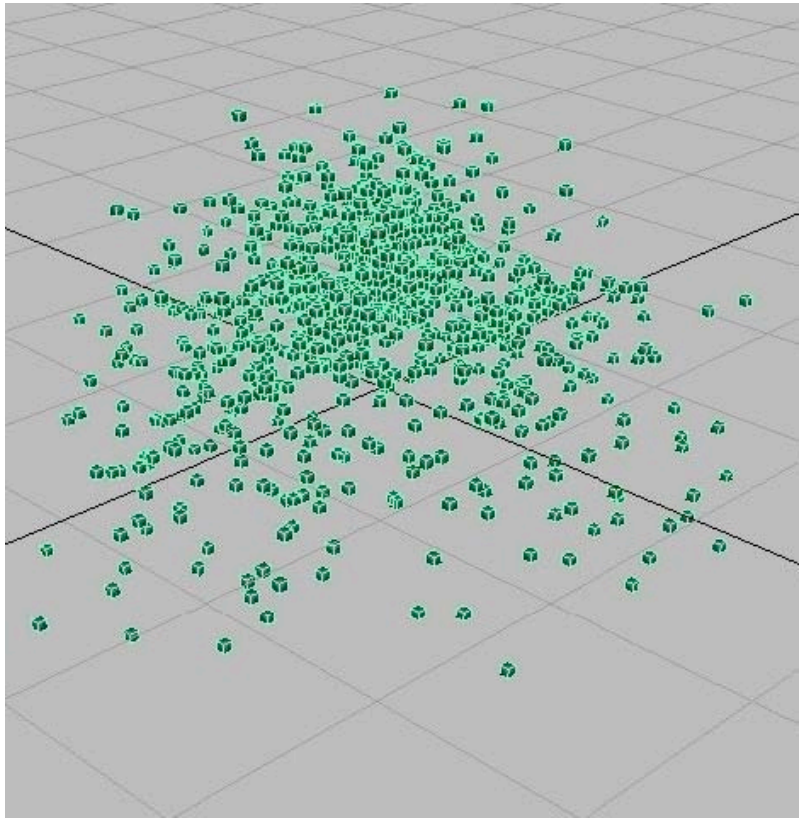
- Simulate motion of flocks of birds or other animals following individual behaviors
- Extends particle systems
- More realistic (and easier!) than scripting the paths of the individual birds

Prior work

- *Eurythmy* (1985)
  - Flocking simulations used force-fields around each bird and around each object.
  - The animator sets the initial positions, headings, and velocities.

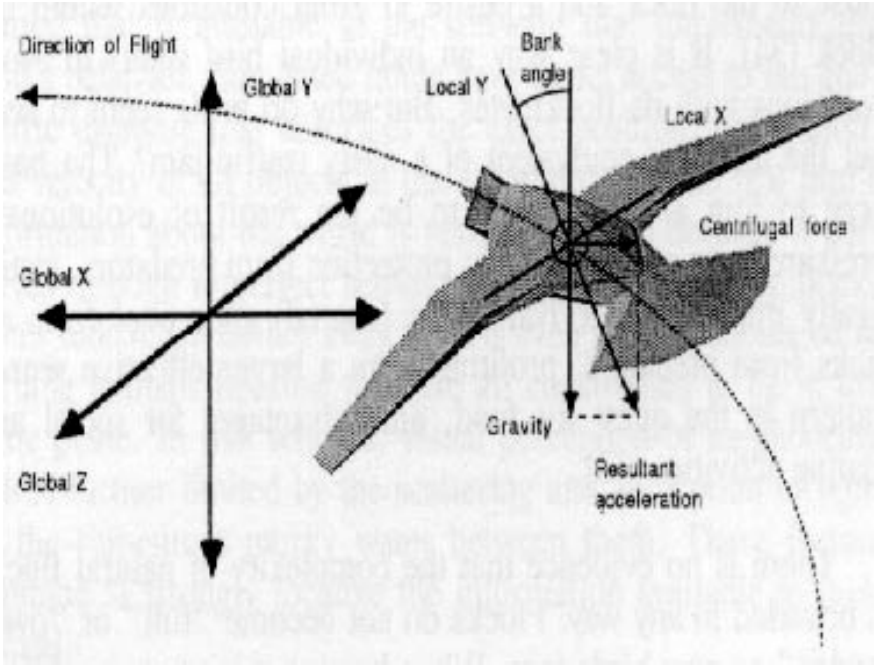
- Other behavioral control work by Karl Sims was based around groups of single objects, not flocks.

Particle Systems



- Particle System is a collection of large number of particles each with their own behaviors
  - At the time of the paper, they were used to model fire, smoke, clouds and ocean waves.
- A Boid is a generalization of a particle
  - Each boid follows a flight model

Boid Model

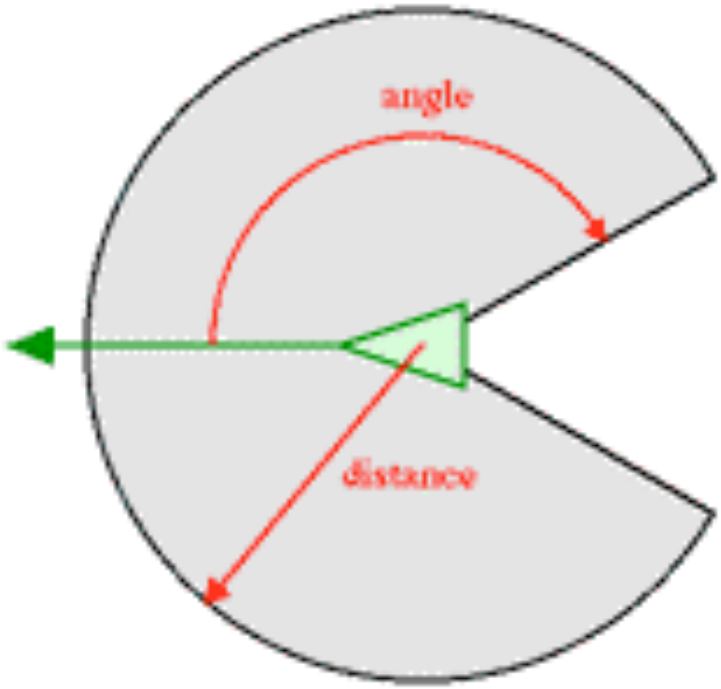


- Local coordinate system
- Direction of flight
- Bank angle
- Affected by
  - Gravity
  - Centrifugal force

Boid Simulation

- Create a boid model that supports flight
- Add individual behaviors that interact with each other:
  - Separation
  - Alignment
  - Cohesion
  - Obstacle Avoidance
- Boids must be able to arbitrate conflicting behaviors as well

Boid Neighborhood

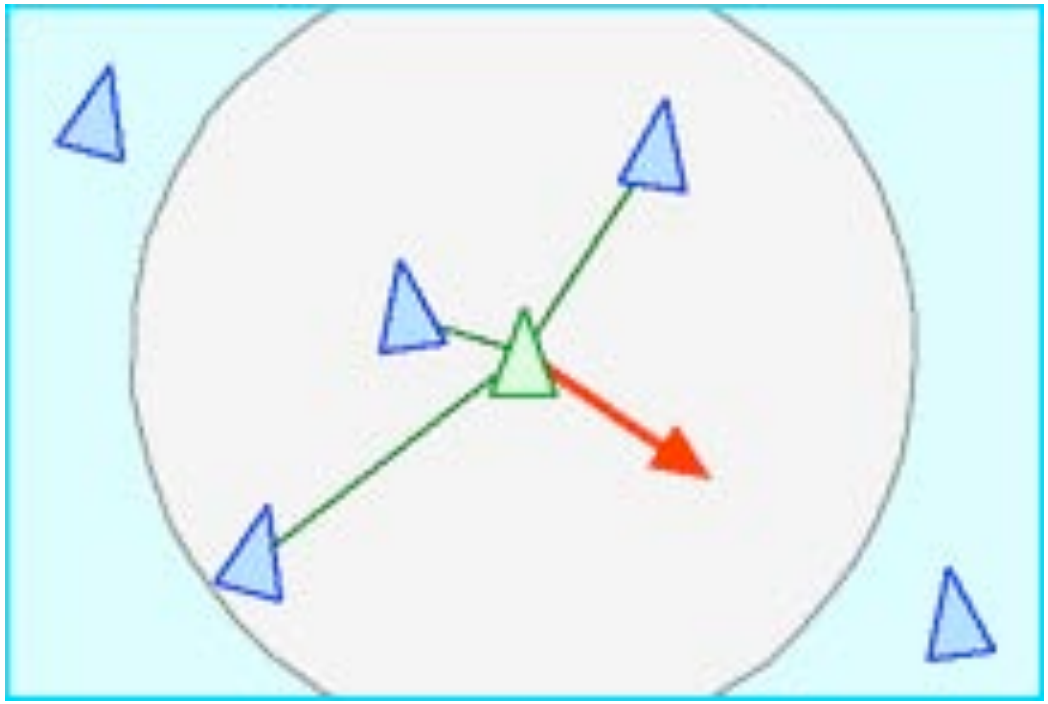


Only sense boids

- within a certain distance
- that are in “front” of you

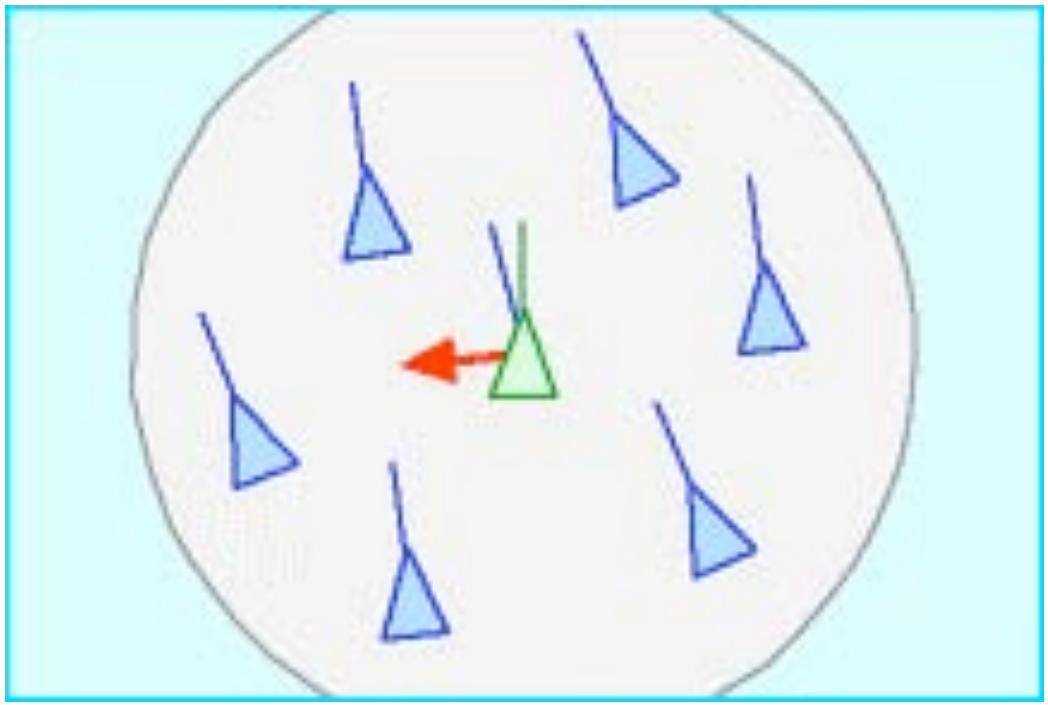
Boid Behavior: Separation

- Keep a certain distance from neighboring boids



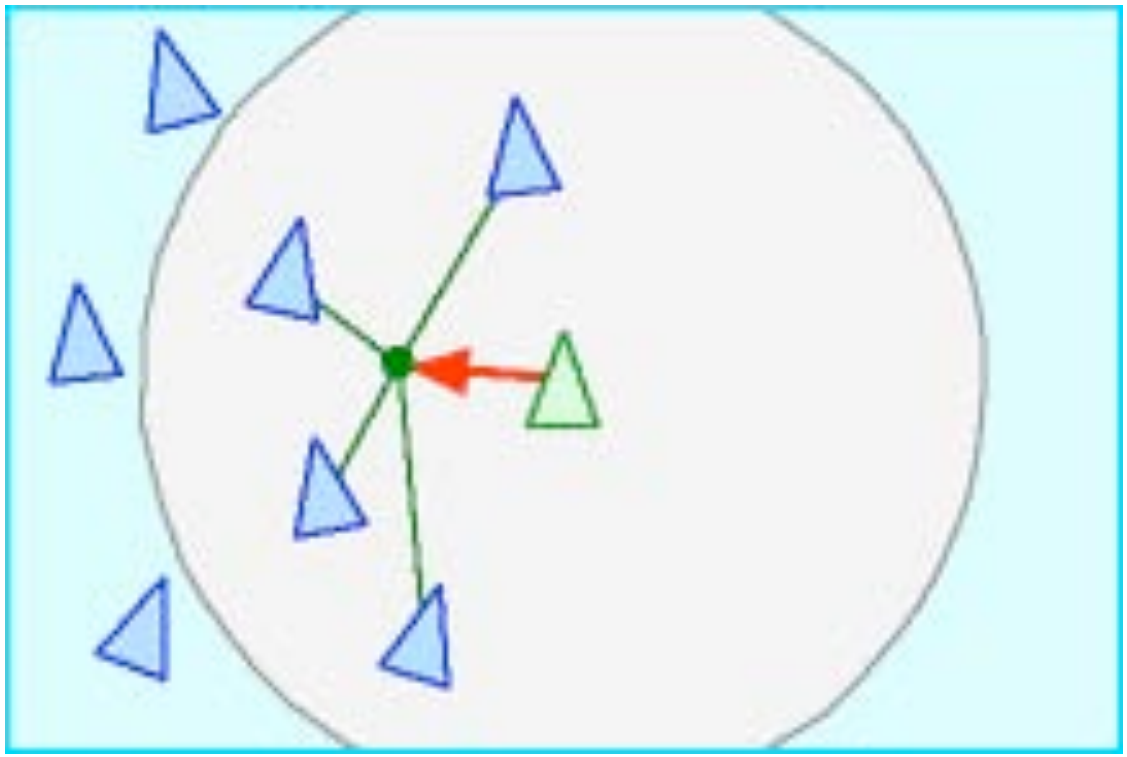
Boid Behavior: Alignment

- Steer toward the general heading of neighboring boids



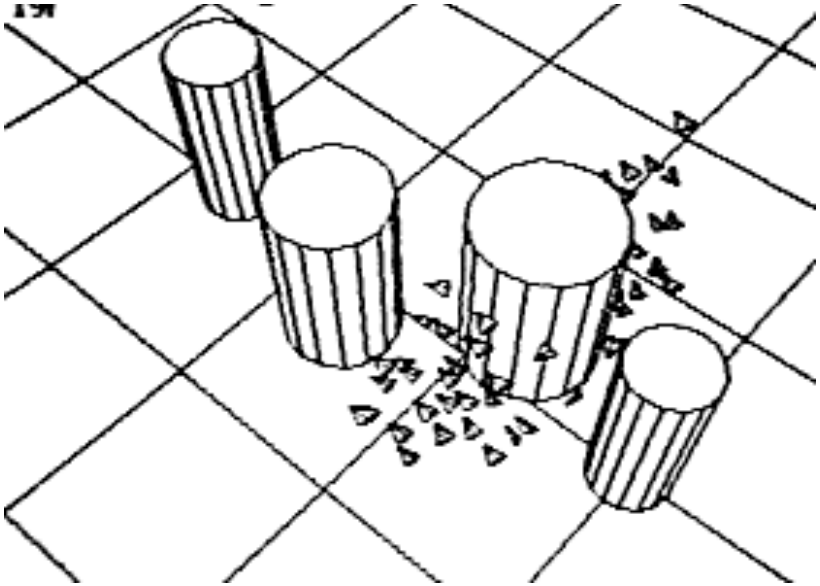
Boid Behavior: Cohesion

- Move toward the average position of neighboring boids



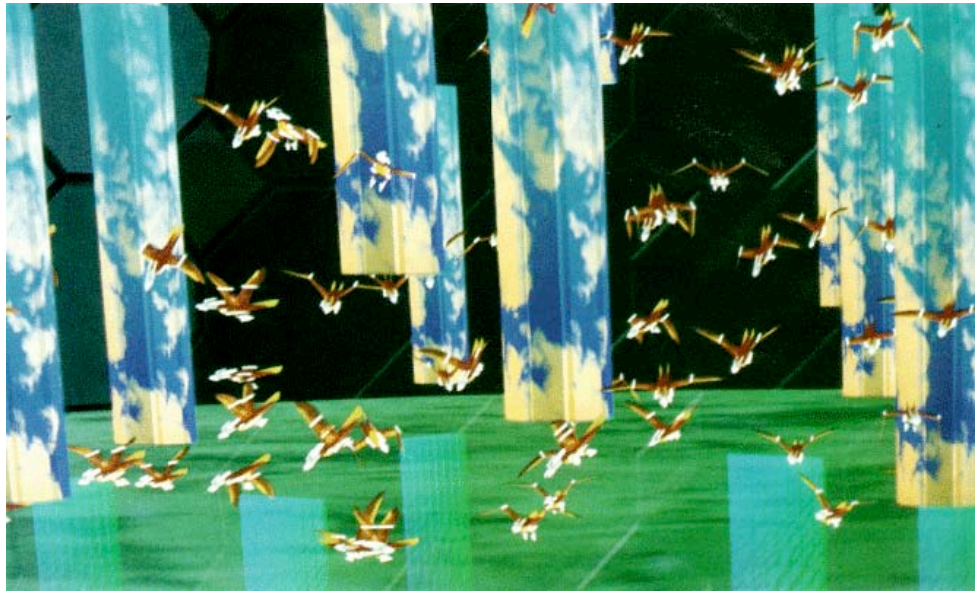
Avoiding Obstacles

- Independently model shape for rendering and collision avoidance
- Boids model uses a *steer-to-avoid* as opposed to force-field approach



Putting it all together

- Simulated flocking behavior that mimics real life flocks, schools and herds
- Combined with a low priority goal seeking results in a scripted path of the flock



- Go to videos

  -Boids demo

  -Stanley & Stella in Breaking the Ice

Summary

- Boids is a model of group motion of flocks based on actions of individual boids
- Boid simulation is an example of emergent behavior
  - Simple local rules lead to complex global behavior
- This paper (from 1987!) has spawned the field of collective behavior modeling/ analysis in physics, biology, artificial life and computer science

Fractals

CS 432 Interactive Computer Graphics

Prof. David E. Breen

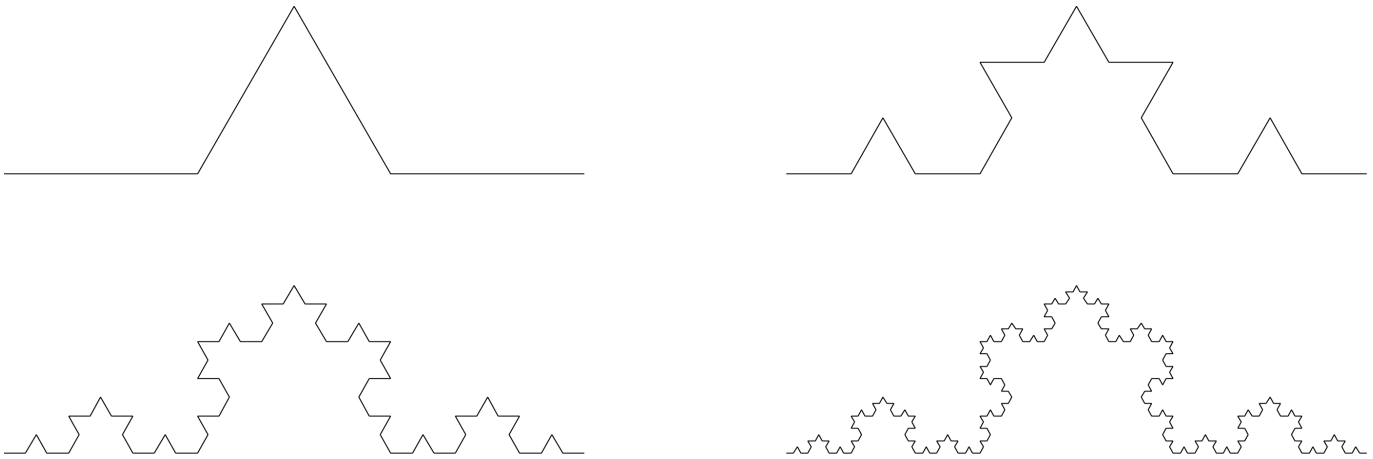
Department of Computer Science

Overview

- Fractal representations
- Algorithms for drawing fractals
- Mandelbrot Set
- Brownian Motion

Introduction to Fractals

- Term Fractal coined by Benoit Mandelbrot
- Properties of fractal:
  - Self-similarity (small portion looks like the whole object)
  - Have fractional dimensions
  - Non-differentiable
  - Infinite length

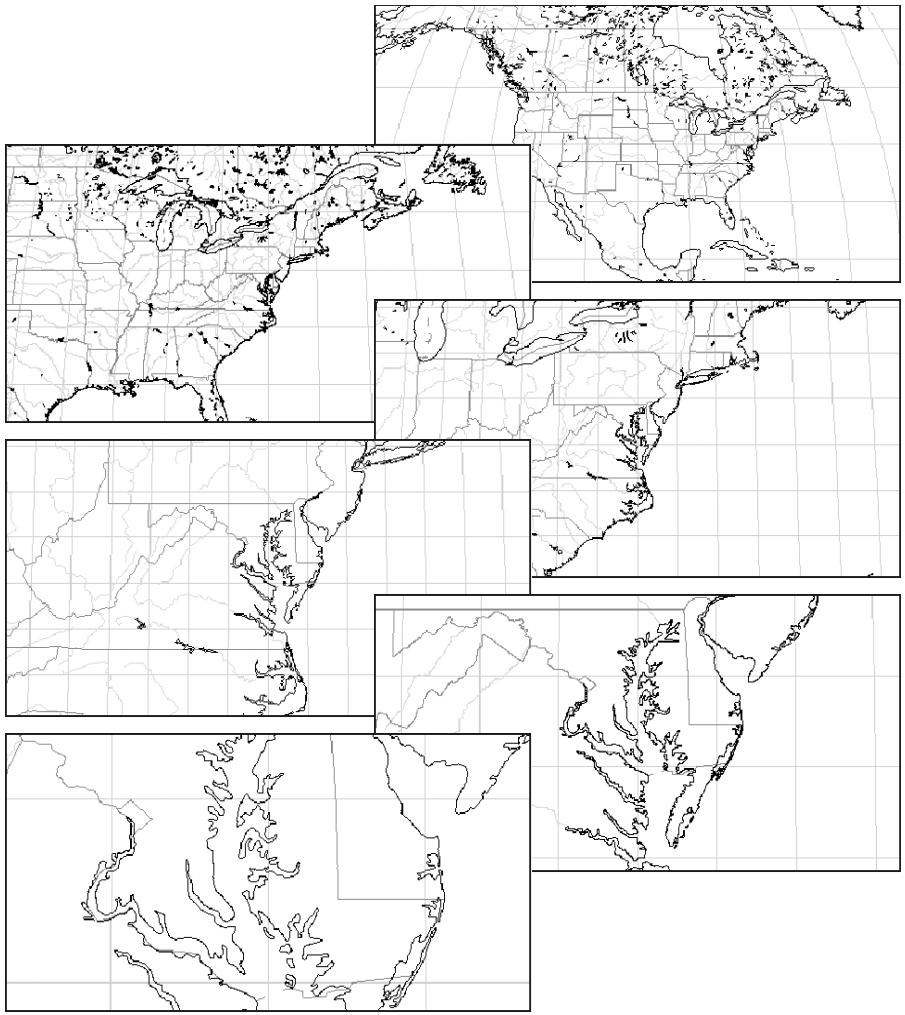


Construction of Koch Curve

Compiled from Gary W. Flake "The Computational Beauty of Nature"

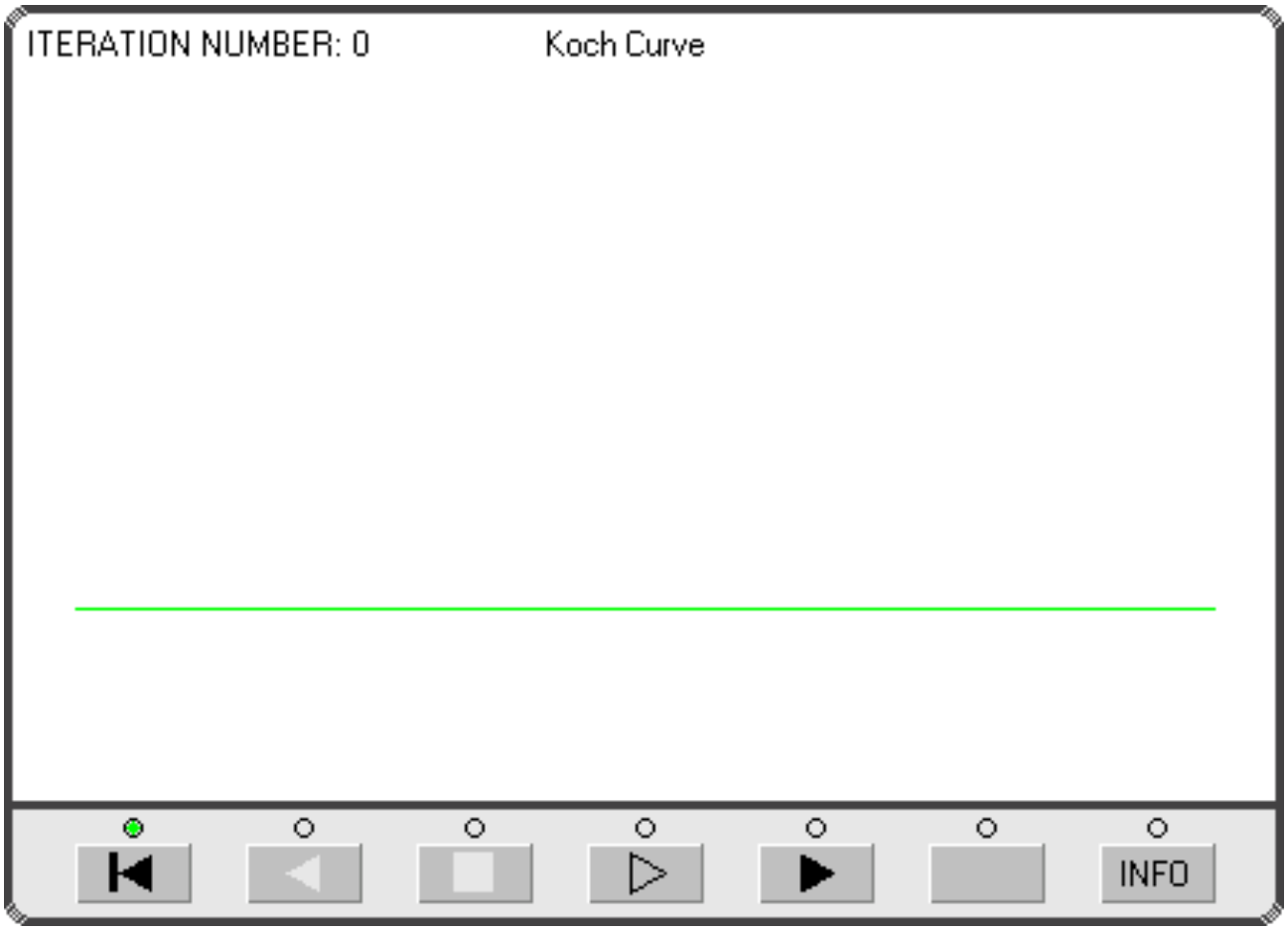
Length of the Fractals

- Lewis Richardson (1961) measured length of coastlines
- Total length increase as measurement stick reduced
- Does this mean coastlines are infinite in length?



Compiled from Gary W. Flake "The Computational Beauty of Nature"

Koch Curve



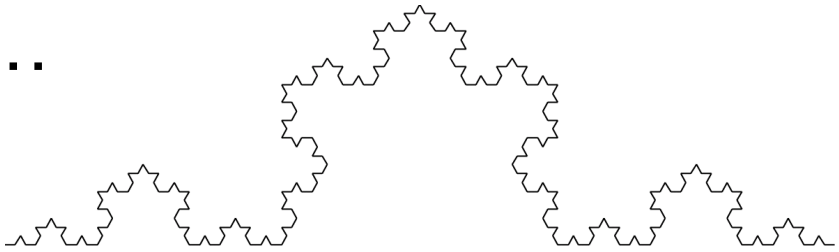
http://www.arcytech.org/java/fractals/

Properties of Koch Curve

- Has no tangent at any point
- Length of the curve at iteration n is $(4/3)^n$

Step	Num of segments	Length of segment	Total length
0	1	1	1
1	4	0.3333333	1.3333333
2	16	0.1111111	1.7777778
100	$1.04858*10^{60}$	$1.94033*10^{-48}$	$3.11798*10^{12}$

- Koch curve is 1.26186 … dimensional object



Compiled from Gary W. Flake "The Computational Beauty of Nature"

Fractal Dimensions

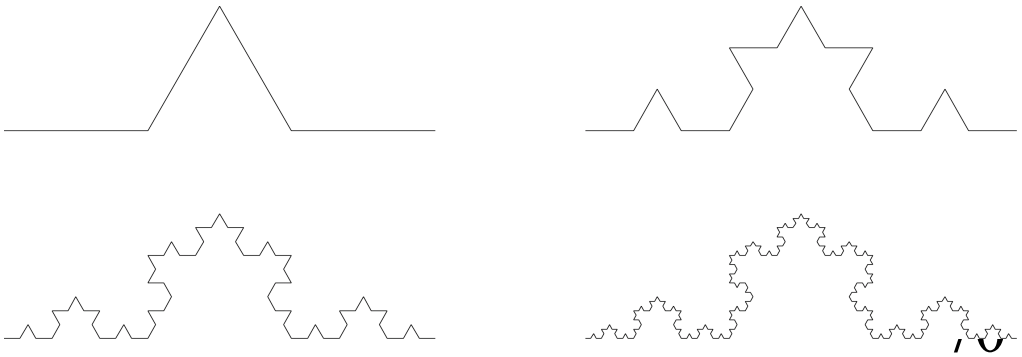
- 1D line of unit length divided into n segments looks like the whole line scaled by $1/n$
- 2D square of unit area divided into n segments looks like the whole square scaled by $1/n^{1/2}$
  - i.e. 4 segments, each one 1/2 the size of the original
- Divide an object into n segments where each segment is scaled by $1/s$
- $s = n^{1/d}$ $d = \log(n)/\log(s)$ $2 = \log(4)/\log(2)$
- The square is divided into 4 parts and each part is 1/2 the size of the original
- $d = \log(4)/\log(2) = 2$

Fractal Dimensions

- $s = n^{1/d}$ $\quad d = \log(n)/\log(s)$
- Koch curve divides line into 4 parts, each part is 1/3 the size of the original
- Therefore the dimension of the Koch curve is:

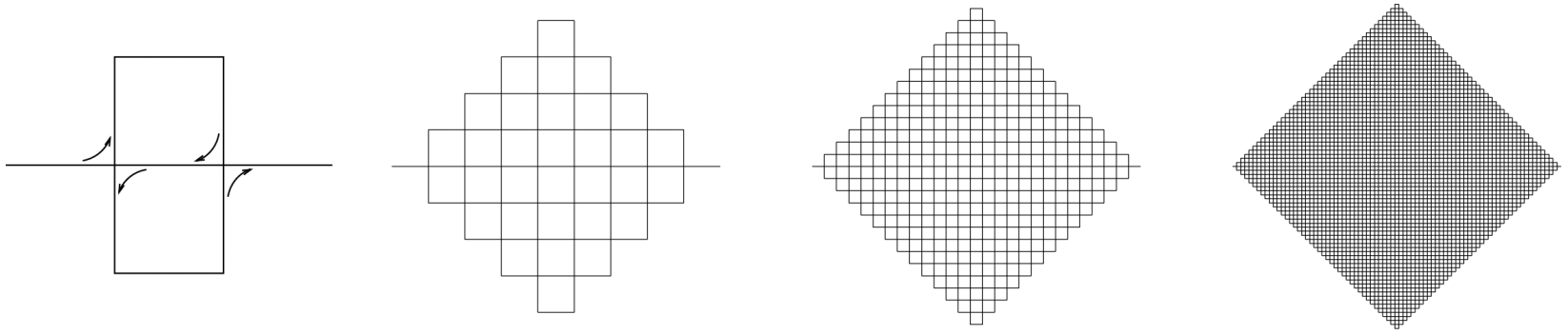
$$4^{\frac{1}{d}} = 3, \quad n^{1/d} = s$$

$$d = \frac{\log(4)}{\log(3)} = \frac{\log(n)}{\log(s)} = 1.26186 \ \ldots$$



Space Filling Fractals

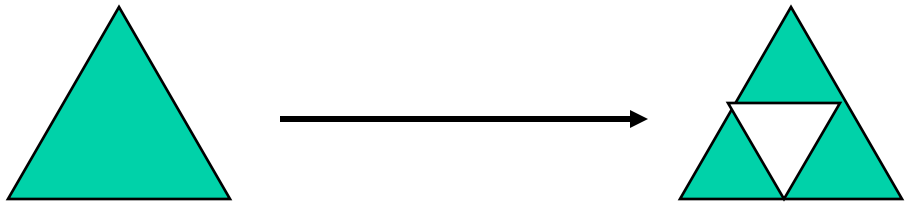
- Peano curve (1890)
- Fills a 2D region
- Fractal dimension - 2
- $s = n^{1/d}$ 9 segments 1/3 original size
  $d = \log(9)/\log(3) = 2$



Construction of Peano Curve

Sierpinski Gasket

Rule based:



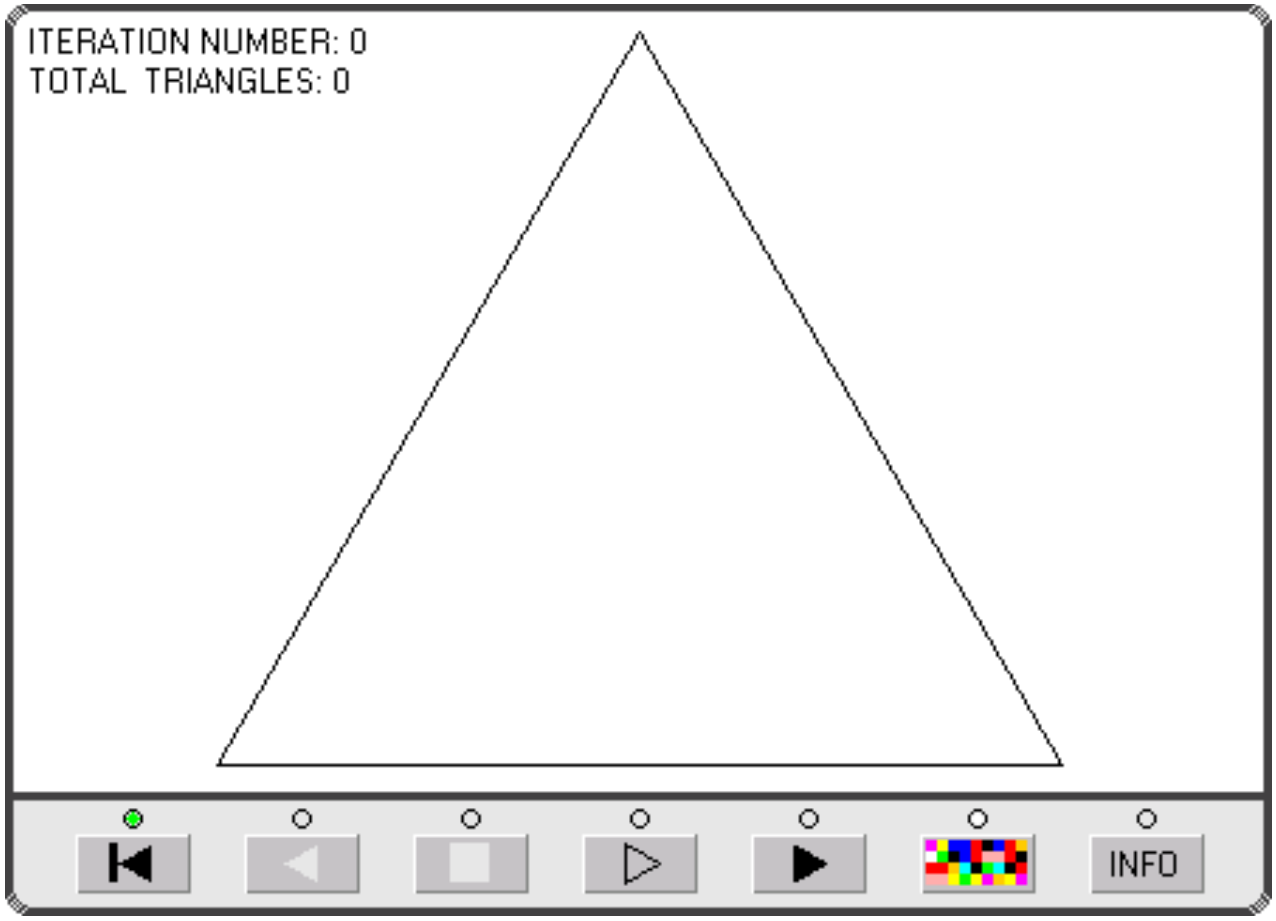
Repeat n times. As n $\rightarrow \infty$

Area $\rightarrow 0$

Perimeter $\rightarrow \infty$

Not a normal geometric object

Sierpinski Gasket



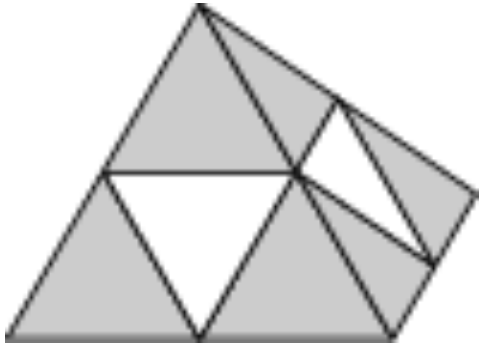
http://www.arcytech.org/java/fractals/

Examples

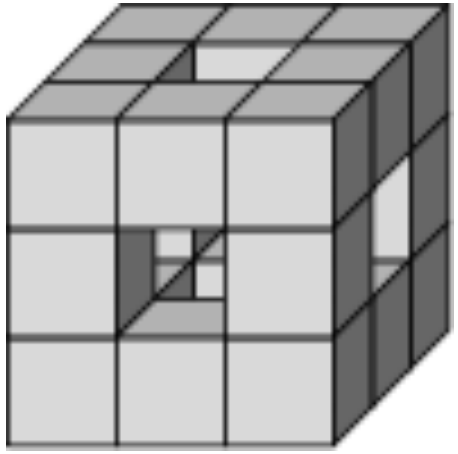
- Koch Curve
  - Scale by 3 each time
  - Create 4 new objects
  - d = ln 4 / ln 3 = 1.26186
- Sierpinski gasket
  - Scale by
  - Create 3 new objects
  - d = ln 3 / ln 4 = 1.58496

Volumetric Examples

$d = \ln 4 / \ln 2 = 2$

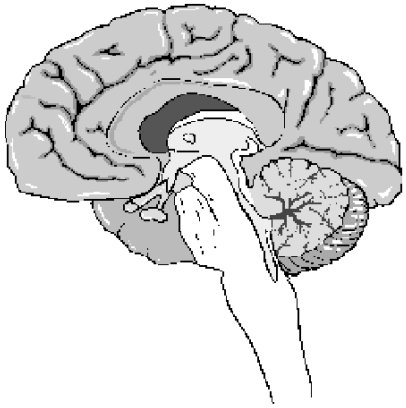


$D = \ln 20 / \ln 3 = 2.72683$

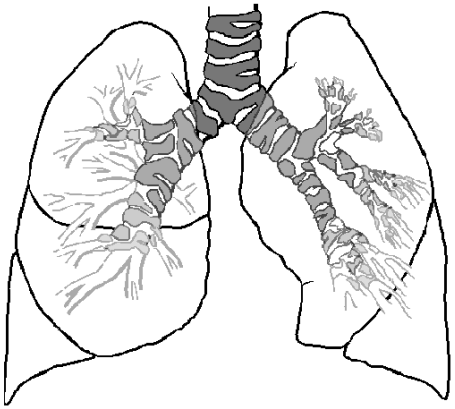


Naturally Occurring Fractals

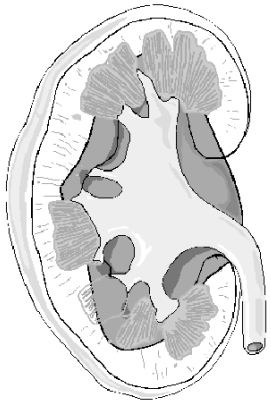
- Fractals often occur in nature
- All natural fractals are *grown*
- We can model natural objects with fractals



Brain



Lungs



Kidney

Compiled from Gary W. Flake "The Computational Beauty of Nature"

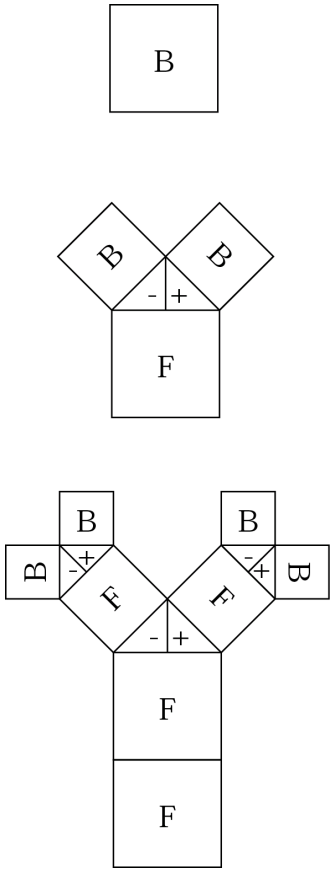
Production Systems

- In 1968 A. Lindenmayer developed a formalism describing plant growth
- Called L-system
- L-system consists of:
  - Seed cell (*axiom*)
  - Production rules

Axiom: B

Rules: $B=>F[-B][+B]$

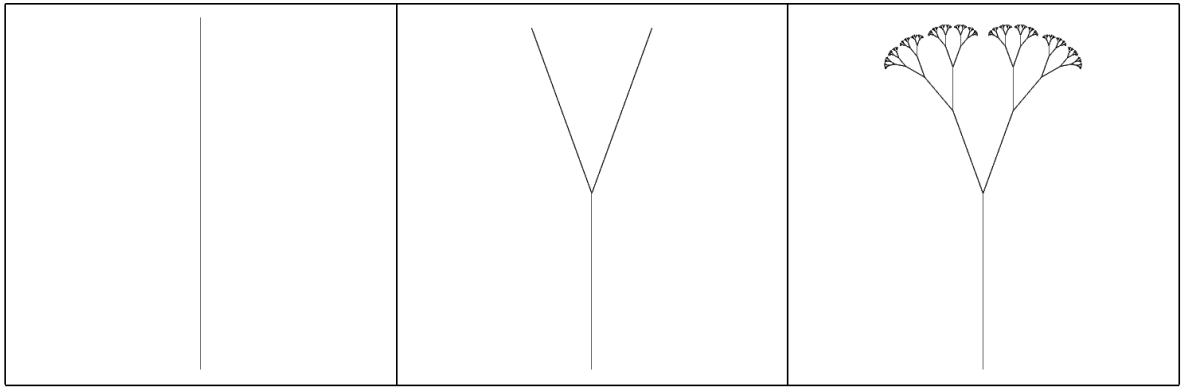
$F=>FF$



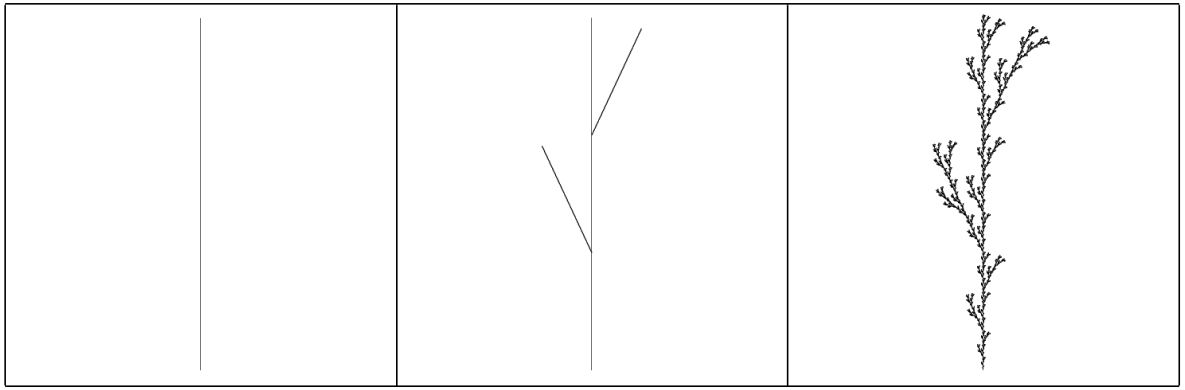
Compiled from Gary W. Flake "The Computational Beauty of Nature"

Plantlike Fractals 1

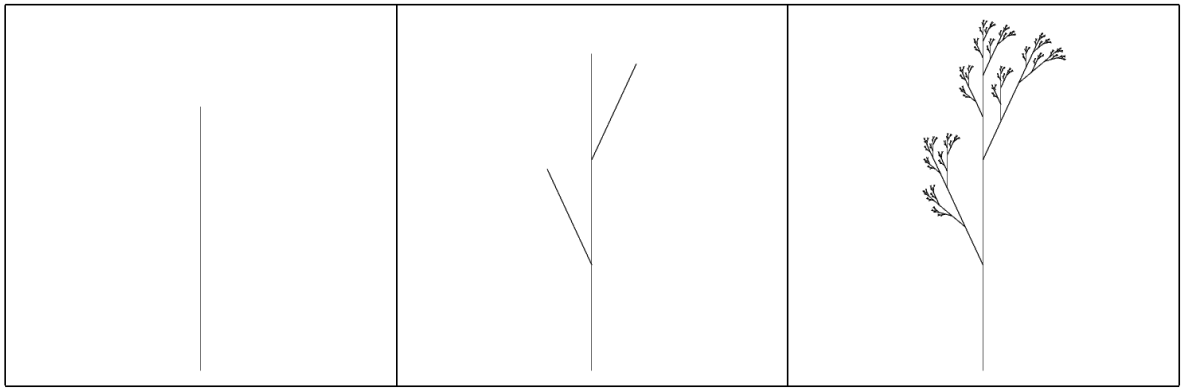
Twig **Angle:** 20 **Axiom:** F **Rule(s):** $F = |[-F][+F]$



Weed-1 **Angle:** 25 **Axiom:** F **Rule(s):** $F = F[-F]F[+F]F$



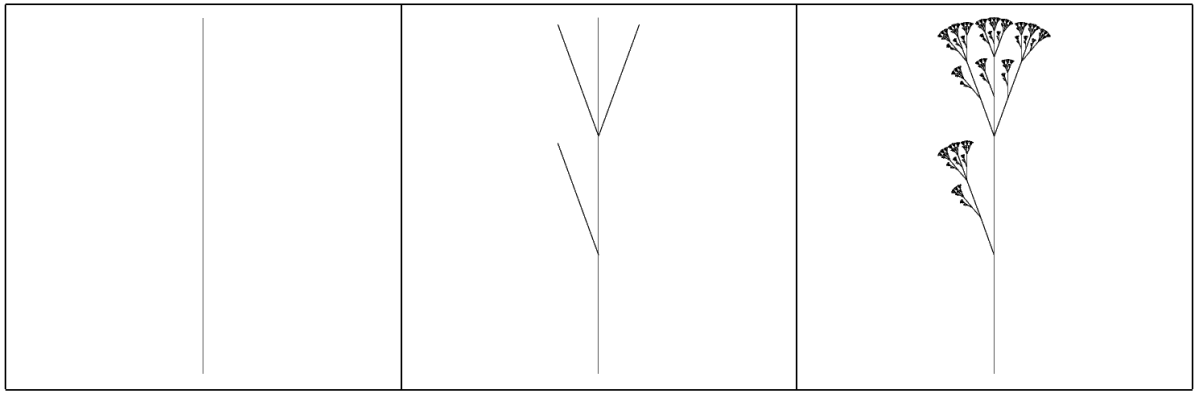
Weed-2 **Angle:** 25 **Axiom:** F **Rule(s):** $F = |[-F]|[+F]F$



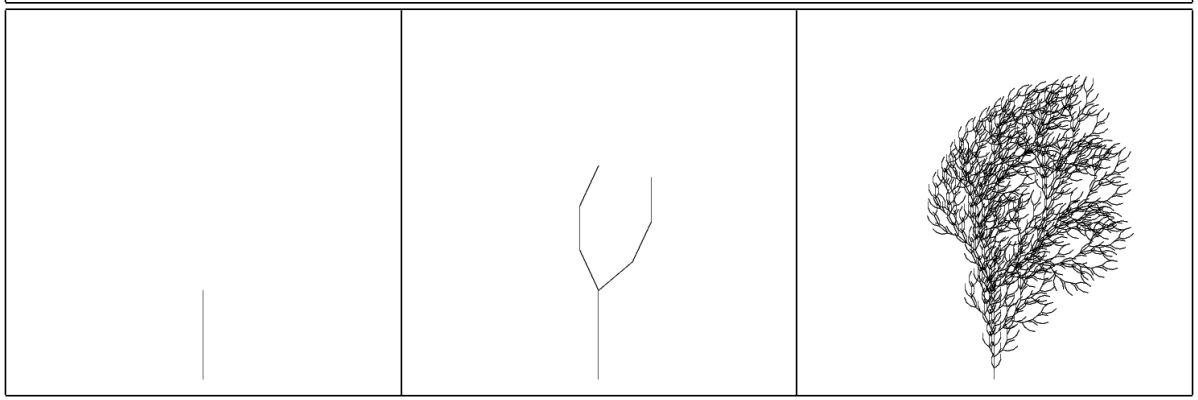
Compiled from Gary W. Flake "The Computational Beauty of Nature"

Plantlike Fractals 2

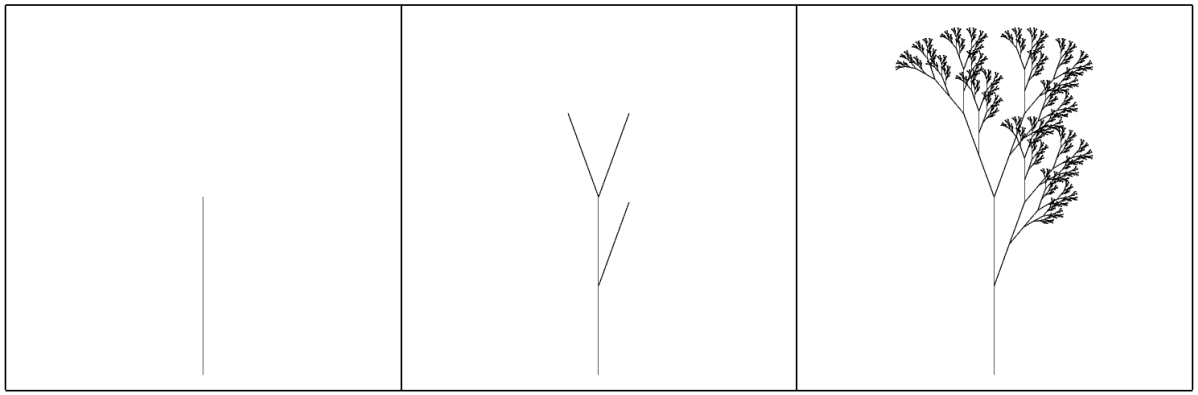
Weed-3 **Angle:** 20 **Axiom:** F **Rule(s):** $F = |[-F]|[+F][-F]F$



Bush-1 **Angle:** 25 **Axiom:** F **Rule(s):** $F = FF + [+F - F - F] - [-F + F + F]$



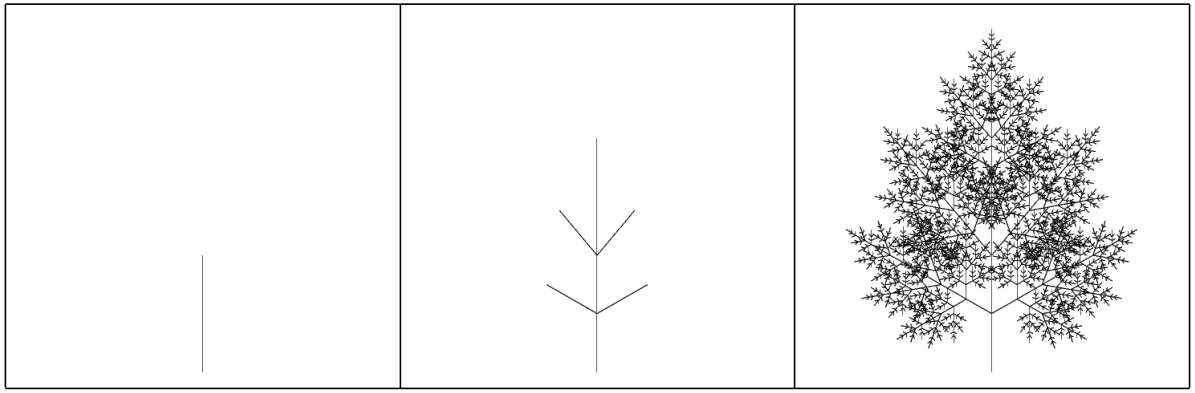
Bush-2 **Angle:** 20 **Axiom:** F **Rule(s):** $F = |[+F]|[-F] + F$



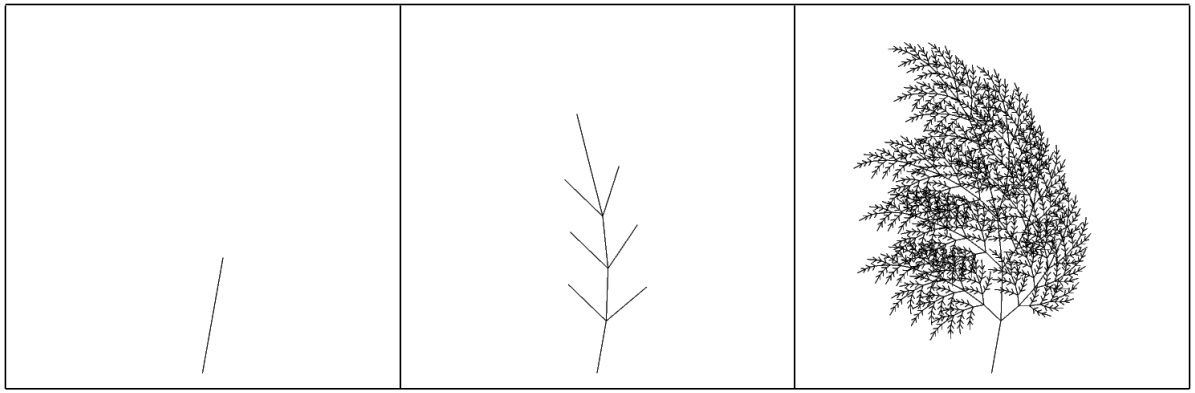
Compiled from Gary W. Flake "The Computational Beauty of Nature"

Plantlike Fractals 3

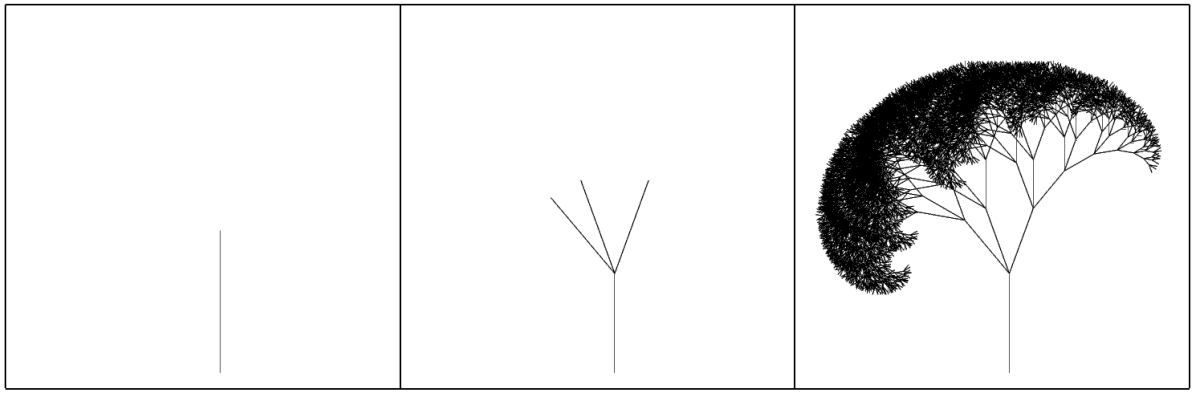
Tree-1 **Angle:** 20 **Axiom:** F **Rule(s):** $F = |[3-F][3+F]|[--F][++F]|F$



Tree-2 **Angle:** 8 **Axiom:** F **Rule(s):** $F = |[5+F][7-F]-|[4+F][6-F]-|[3+F][5-F]-|F$



Tree-3 **Angle:** 20 **Axiom:** F **Rule(s):** $F = |[--F][+F]-F$



Compiled from Gary W. Flake "The Computational Beauty of Nature"

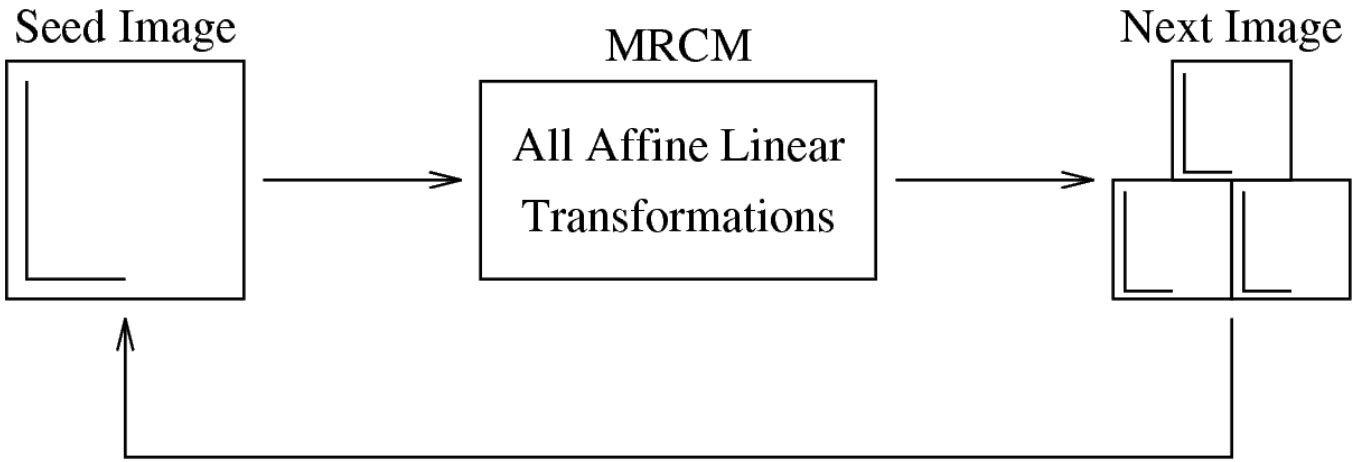
Affine Transformation Fractals

- Fractal often described to contain miniature versions of itself.
- We can use affine transformations to describe where these miniatures should be placed
  - Translation
  - Rotation
  - Scaling
- Self-affine fractals may have different scaling factors in different dimensions

Compiled from Gary W. Flake "The Computational Beauty of Nature"

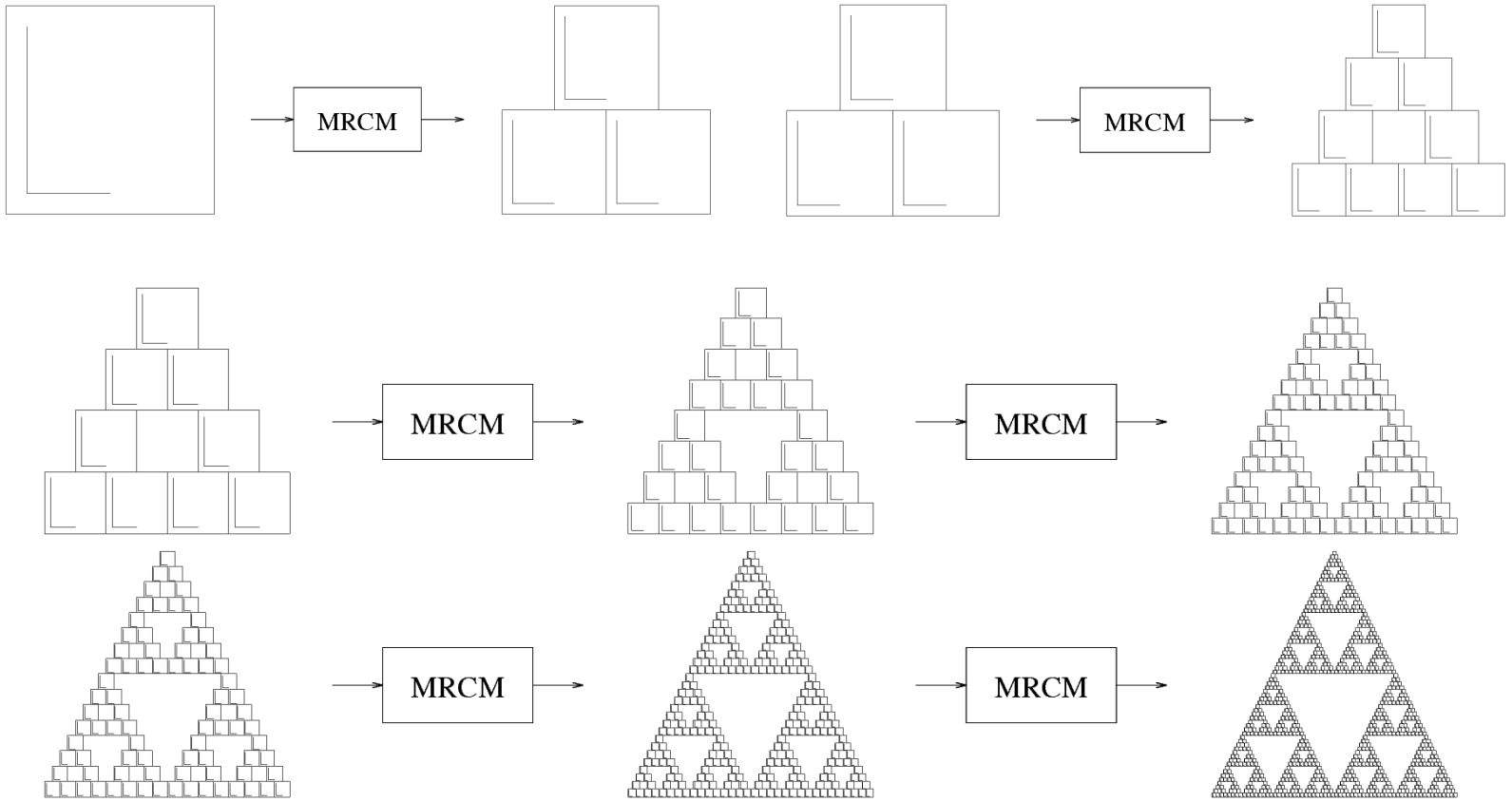
Multiple Reduction Copy Machine Algorithm

- Can be simulated with real copy machine
- Copy machine takes a seed image
- Creates several transformed copies of it as an output.
- Procedure is recursive



Compiled from Gary W. Flake "The Computational Beauty of Nature"

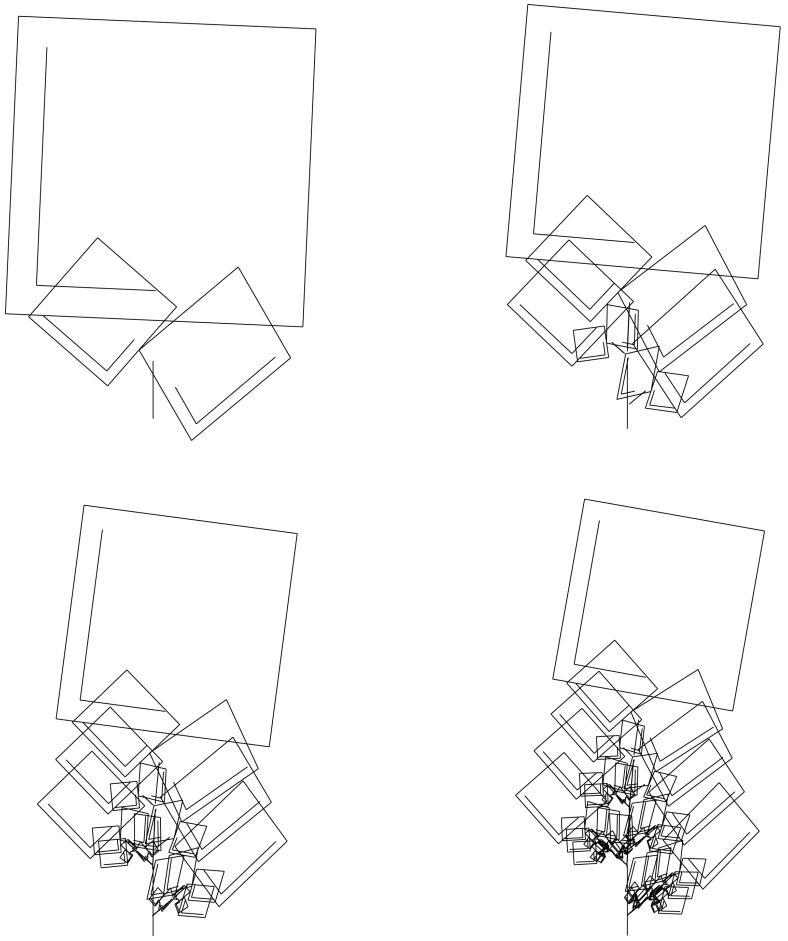
MRCM Examples



Compiled from Gary W. Flake "The Computational Beauty of Nature"

Problems with MRCM

- To get good image we must compute MRCM to large depth
- Computing time grows exponentially with depth
- Problem amplified when the reduction in size during one iteration is small

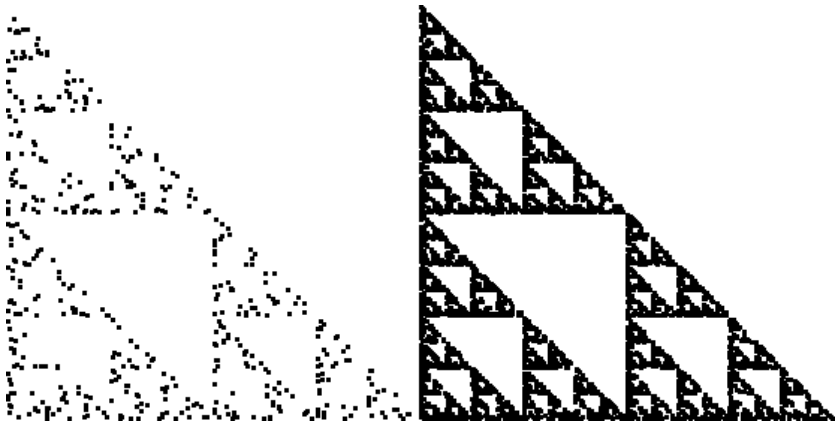


Fern Leaf

Compiled from Gary W. Flake "The Computational Beauty of Nature"

Iterated Functional System

- Observations:
  - Fractal consist entirely of points
  - Randomly and recursively apply affine transformation $L_i(p)$ to point p
  - Converges to a fractal

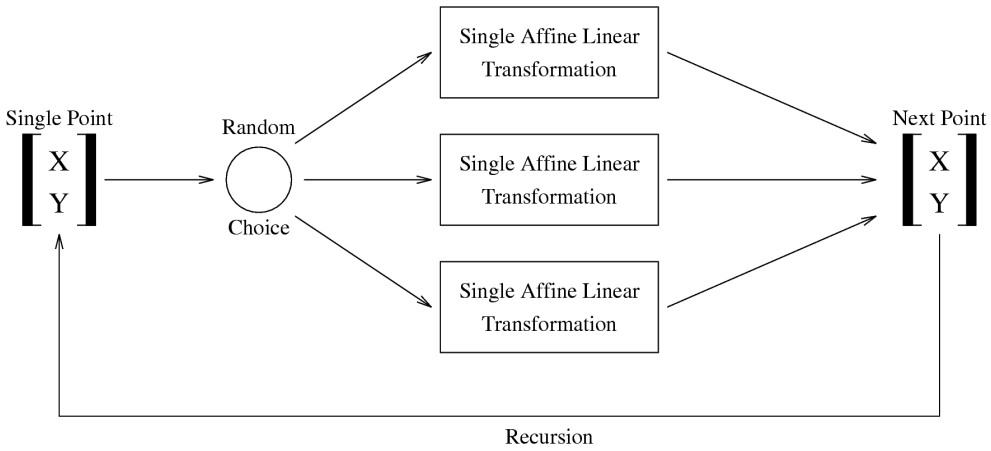


500 points 5000 points

Compiled from Gary W. Flake "The Computational Beauty of Nature"

IFS Algorithm

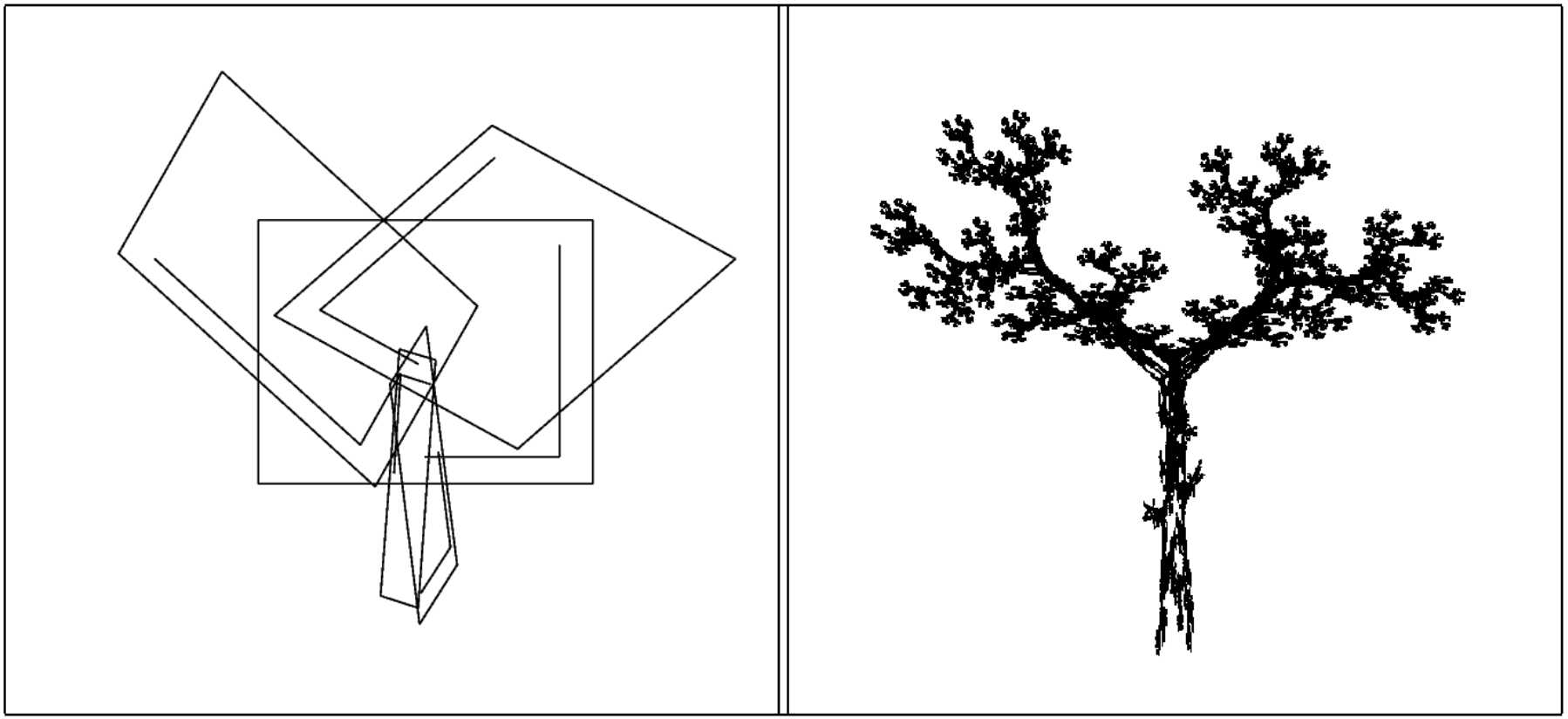
- Pick random point of the seed image
- Randomly pick one of the affine transformations
- Transform the point
- Continue recursively



Fern Leaf

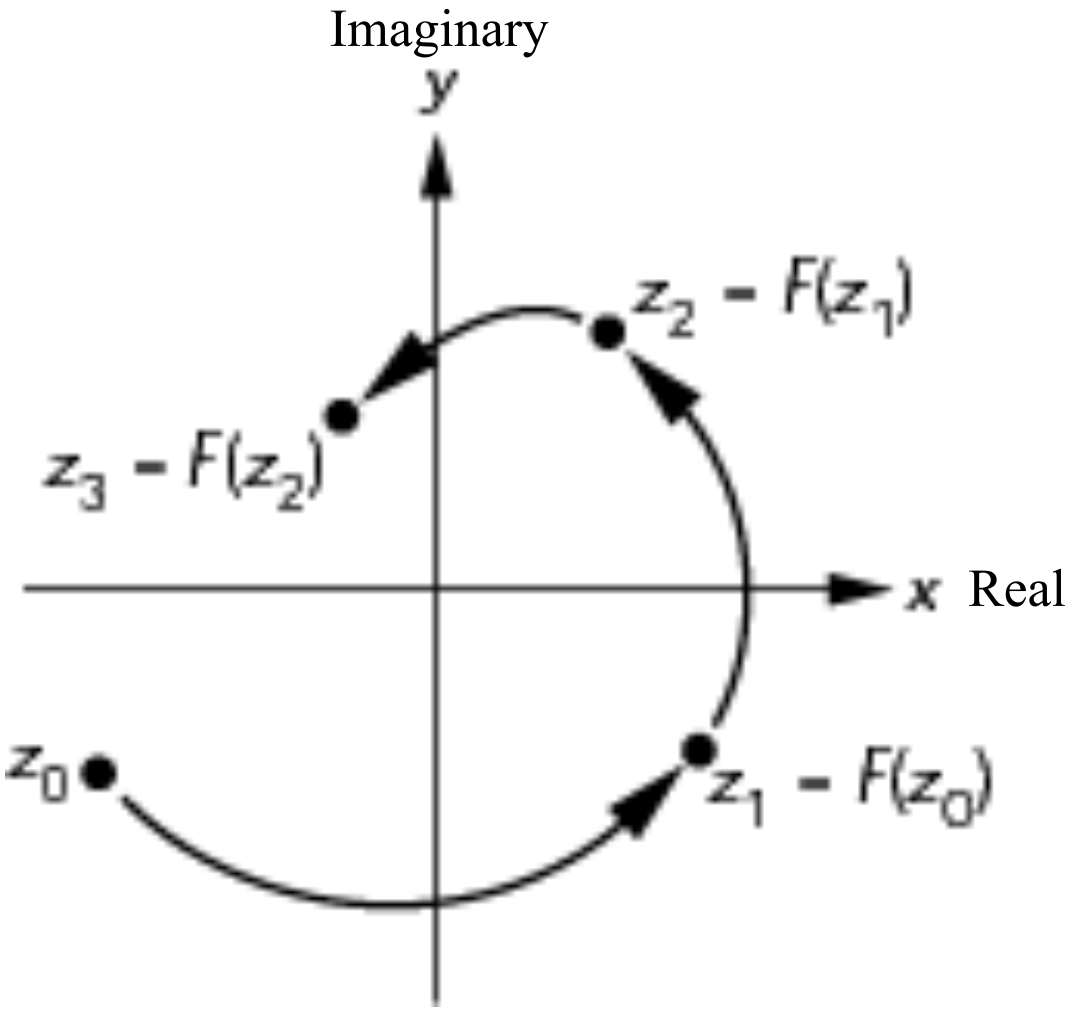
Compiled from Gary W. Flake "The Computational Beauty of Nature"

IFS Generated Tree



Compiled from Gary W. Flake "The Computational Beauty of Nature"

Iteration in the Complex Plane



Mandelbrot Set

- $x_n = x_{n-1}^2 + c$ for some complex number c
  - For some c, $x \rightarrow 0$ as $n \rightarrow \infty$ $\in$ Mandelbrot Set
  - For some c, $x \rightarrow \infty$ as $n \rightarrow \infty$
  - For others *neither*

For each c in the complex plane

$x_0 = 0$

for ($n = 1$ to n_{max}) {

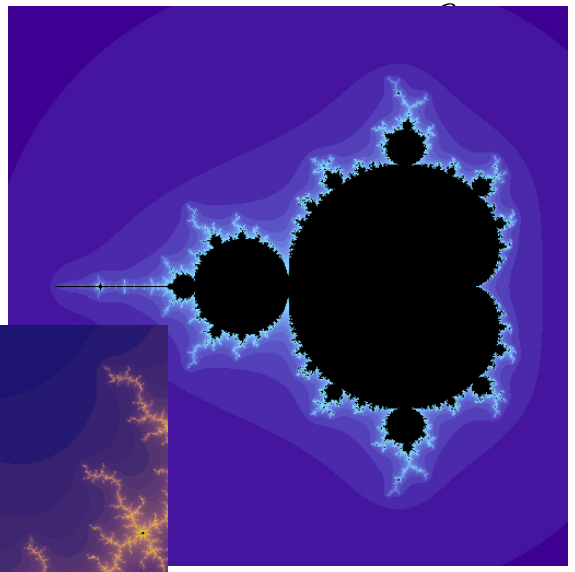
$x_n = x_n^2 + c$

if ($|x_n| > 2$) break

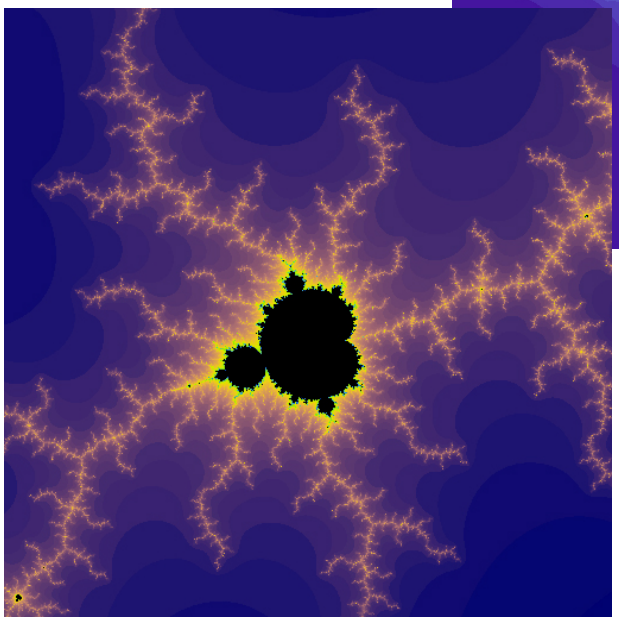
}

If ($n < n_{max}$) color c = white

else color c = black



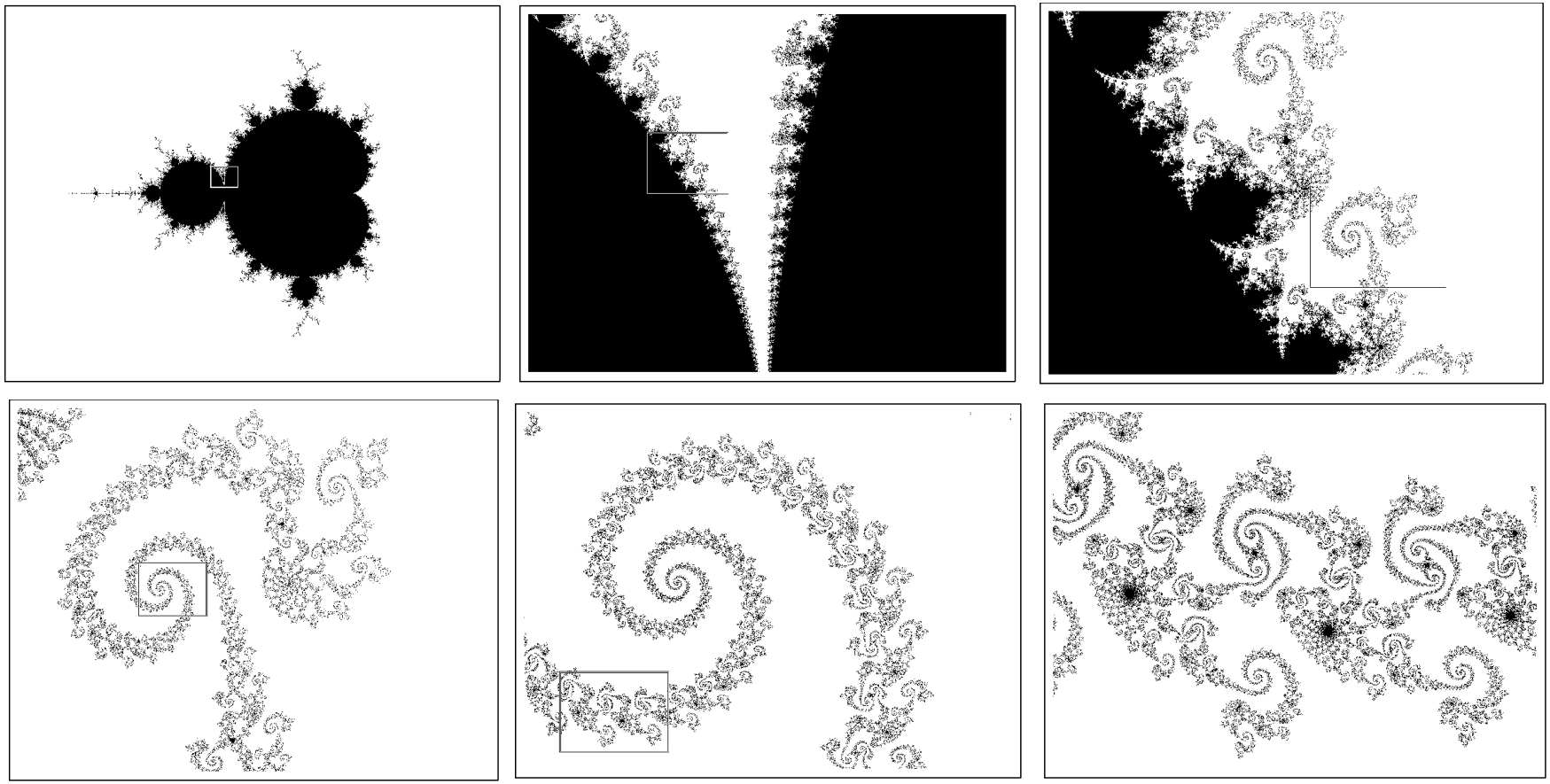
Mandelbrot Set



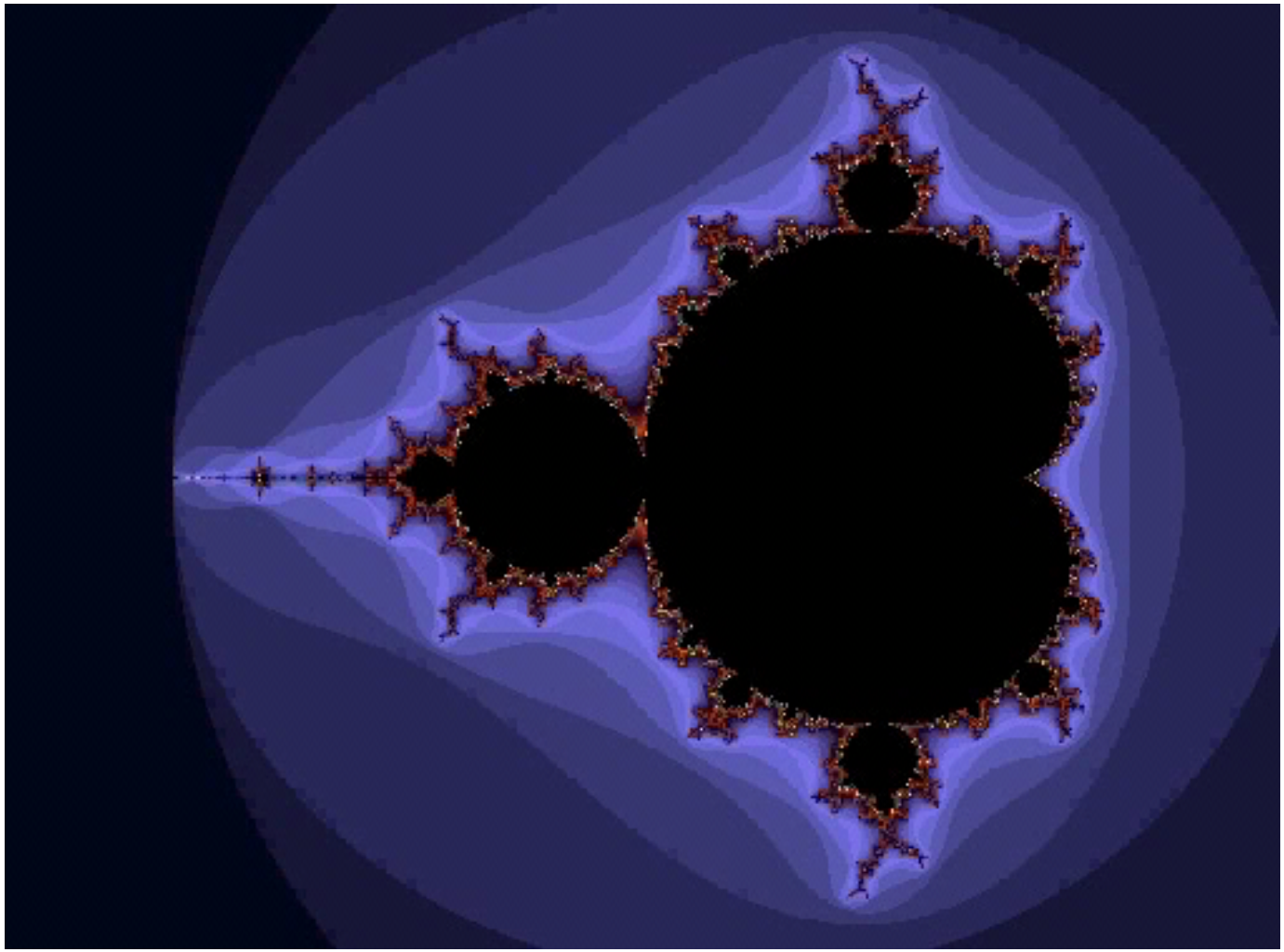
Magnified region

from http://www.astro.su.se/~alexis/fractals/

Mandelbrot Set

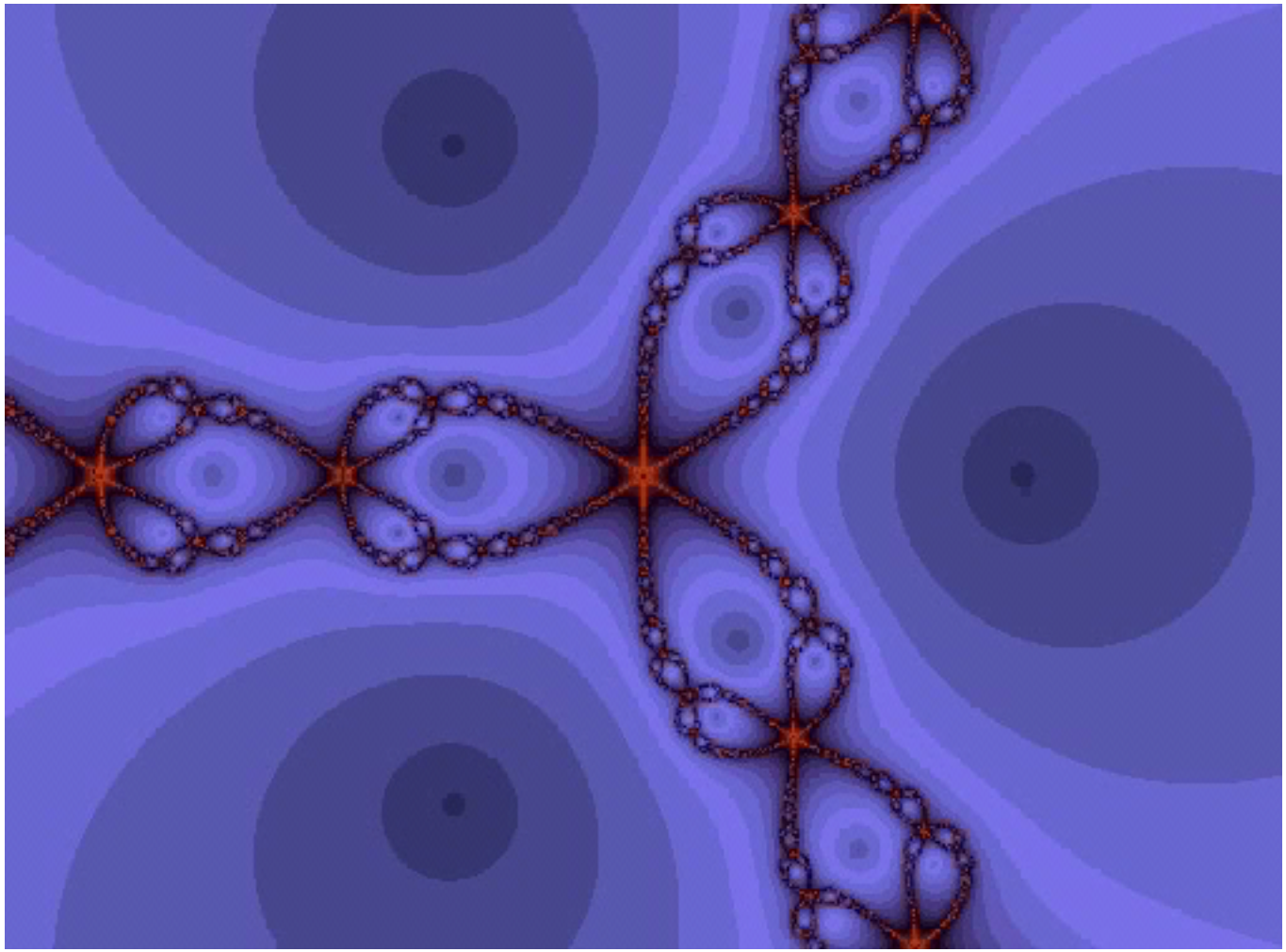


Mandelbrot Set



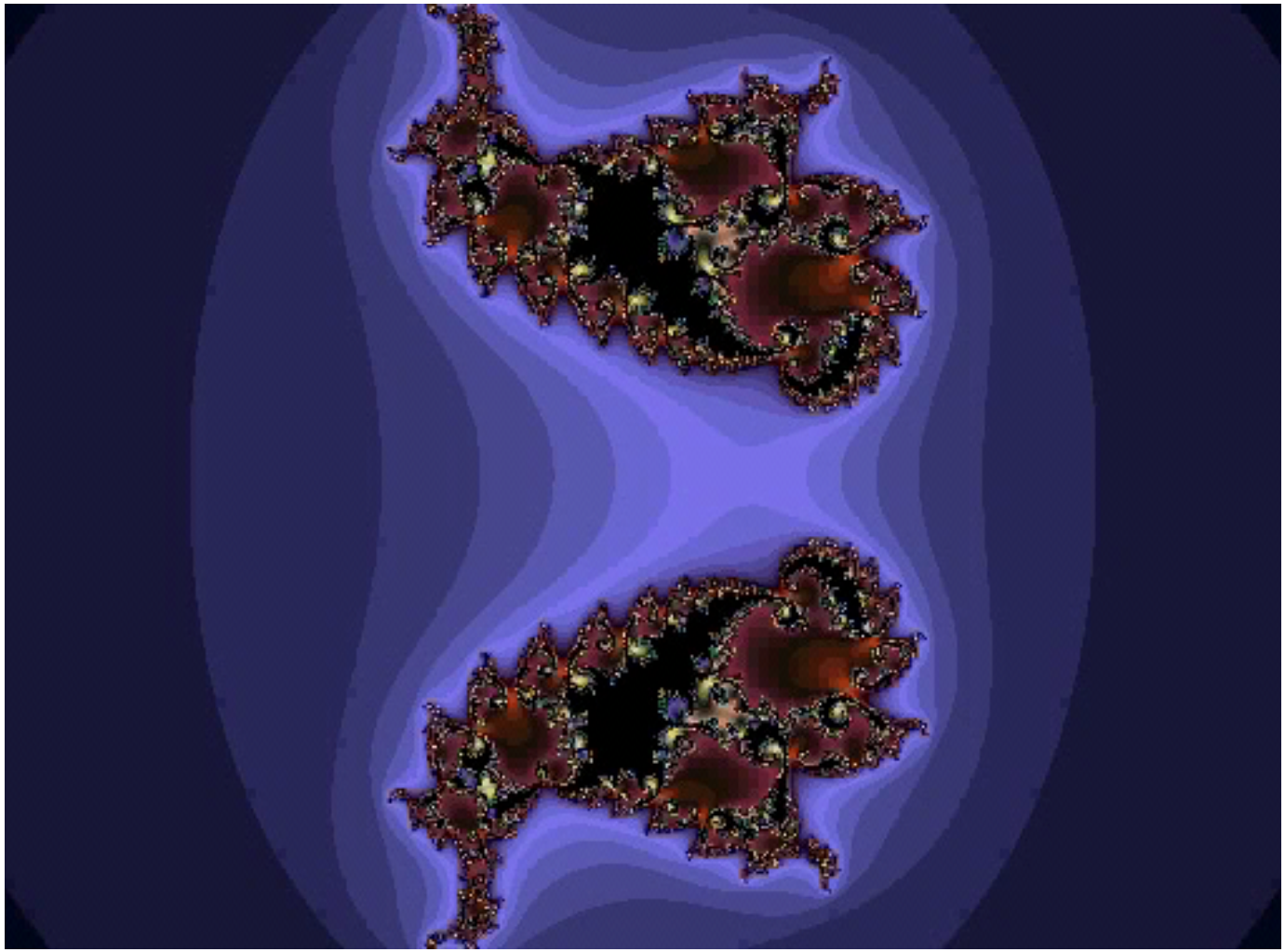
Newton Set

$f(x) = x^3 - 1$ $\quad x_{n+1} = x_n - f(x) / f'(x)$



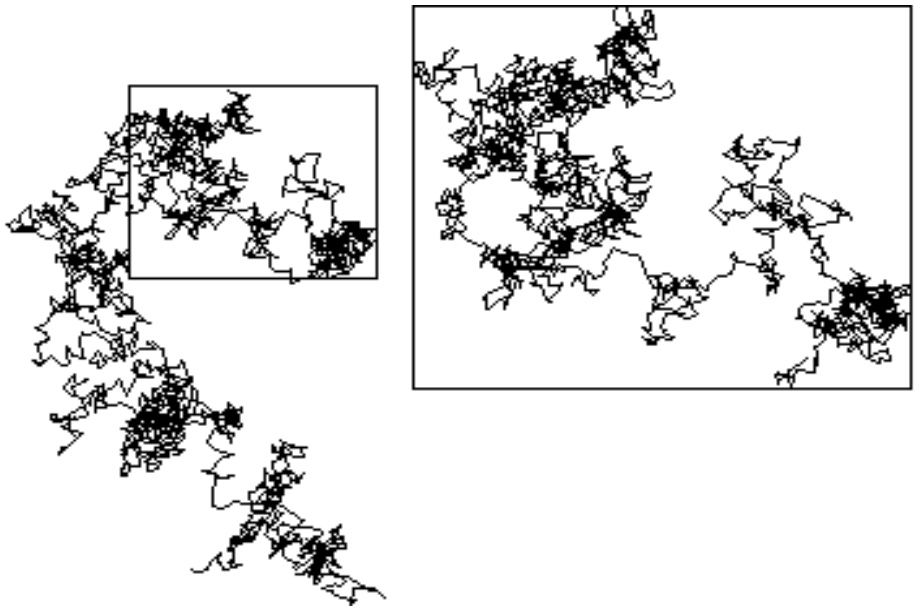
Phoenix Set

$$z_{n+1} = z_n^2 + \mathrm{Re}(c) + \mathrm{Im}(c) * z_{n-1}$$



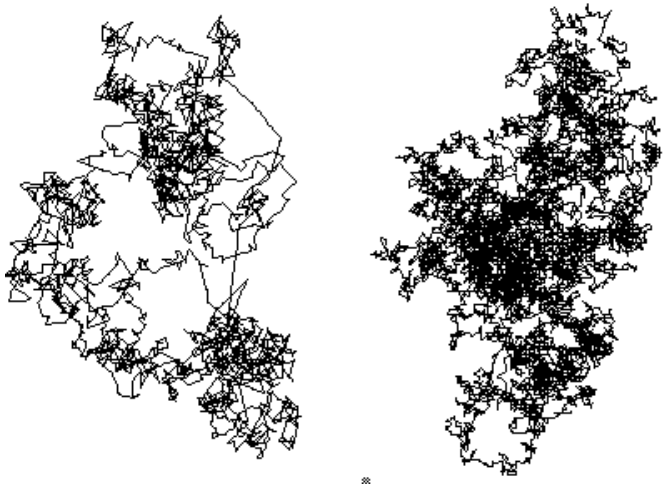
Brownian Motion

- The dance of pollen grains in a water drop (Robert Brown in 1827)
- Explained by Albert Einstein in 1905
- Self-similar motion
- Infinite length
- Used to simulate rivers mountains and other random natural phenomena



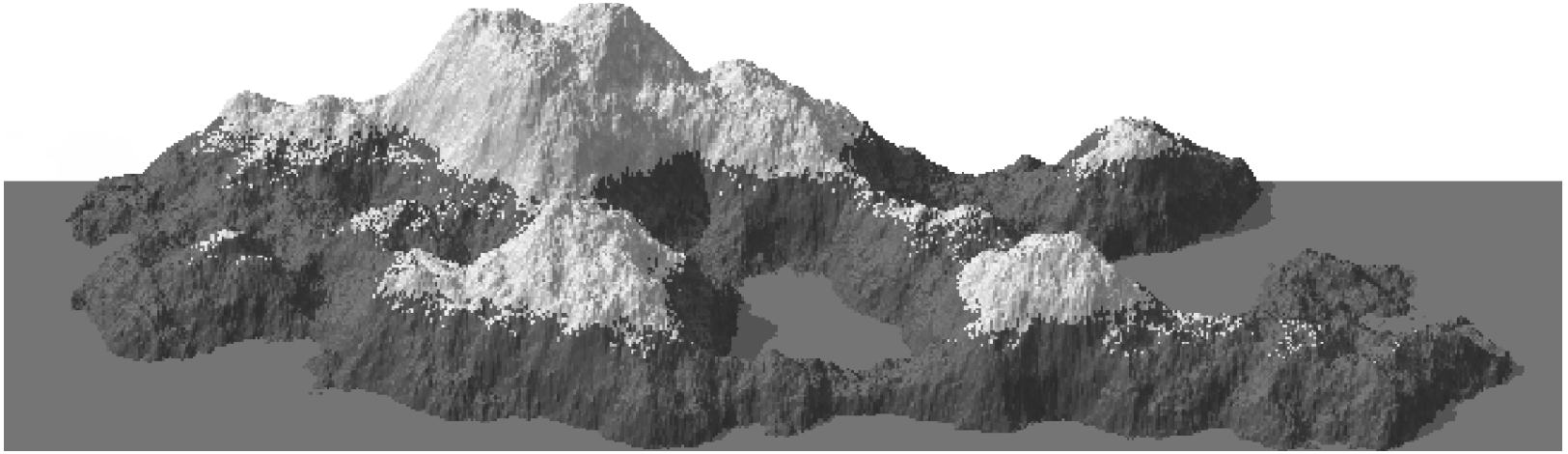
Fractional Brownian Motion

- Fractional BM (fBM) is a generalization of BM to include memory
  - Integral on progress of random walk
- fBM characterized by it's power spectrum
  - BM has $1/f^2$ power spectrum
  - fBM had $1/f^{\beta}$ power spectrum with $1.0 \leq \beta \leq 3.0$
- Think of β as controlling terrain roughness



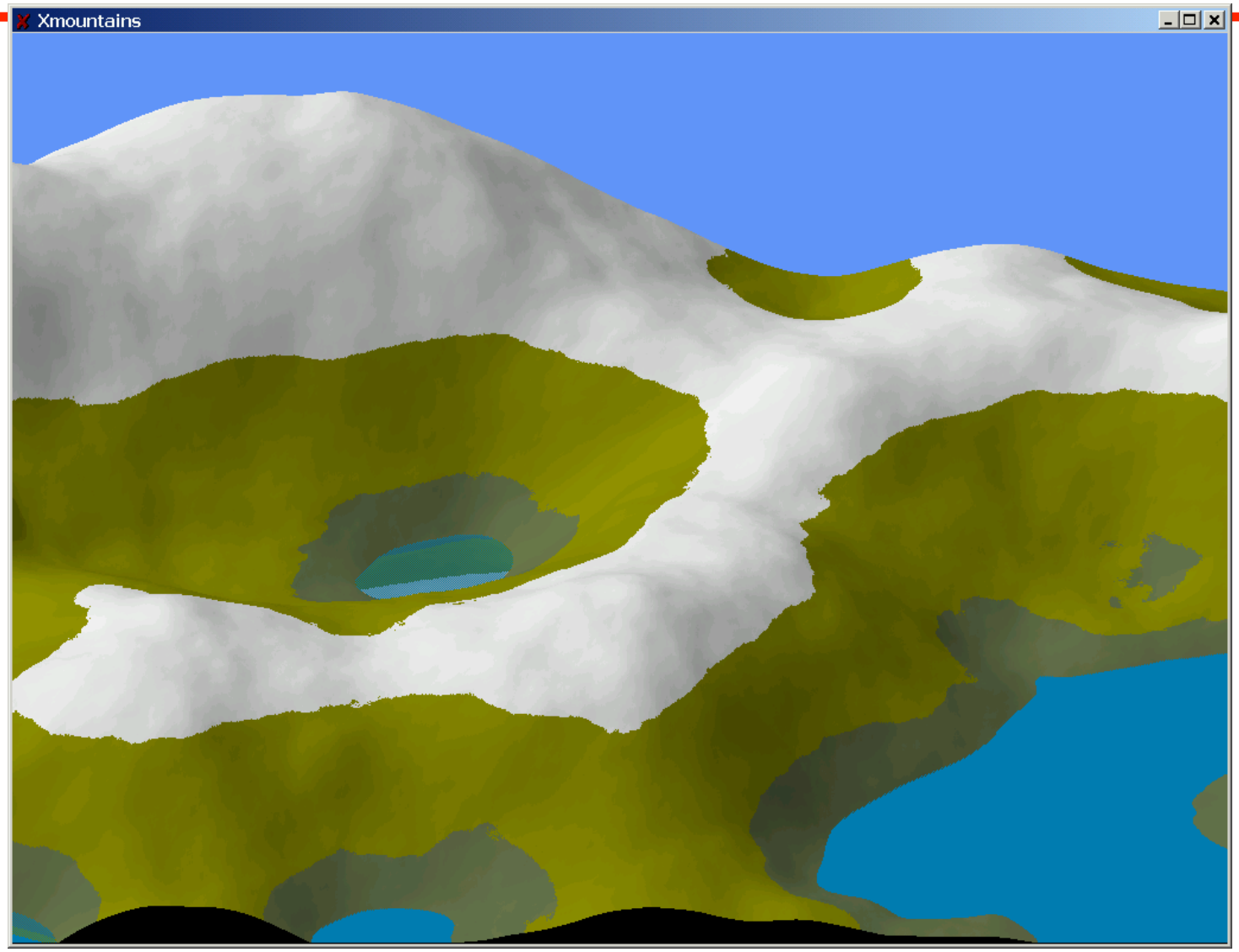
Voss & Saupe in "The Science of Fractal images"

XMOUNTAINS



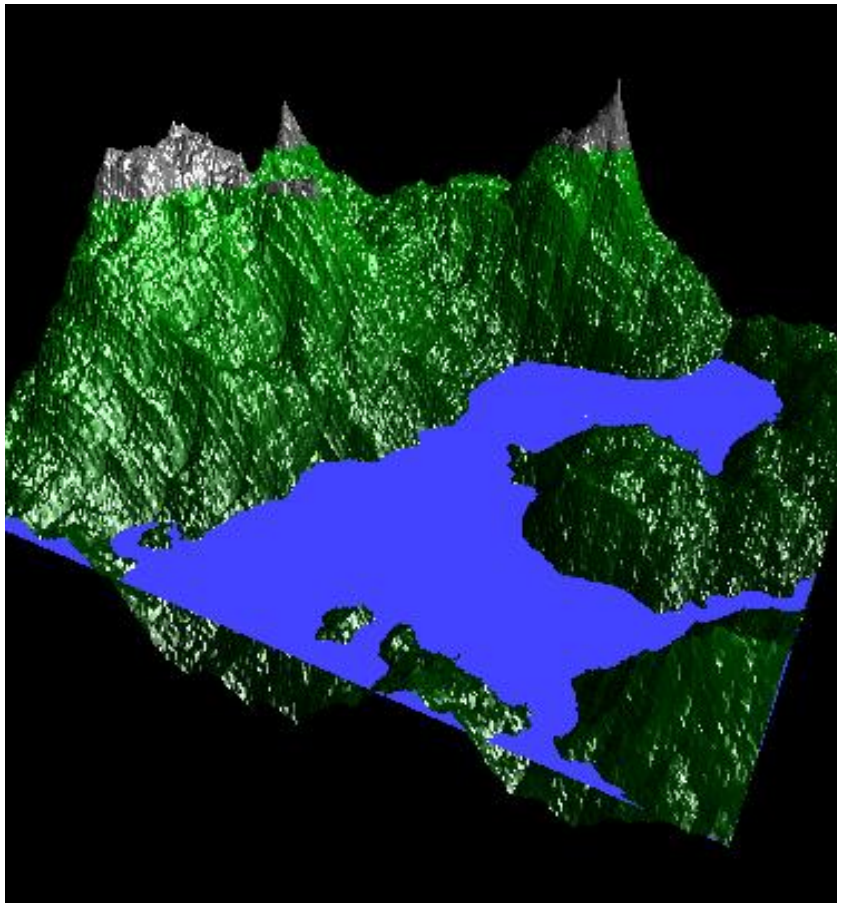
- Landscape generated by the `XMOUNTAINS` program by Steven Booth
- Realistic, self-similar image

XMOUNTAINS : Changing Fractal Dimension



Random Midpoint Displacement

- fBM computations are time consuming
- RMD are faster to compute but less realistic
- Basic idea:
  - Start with 2 (2D) 4 (3D) random points.
  - Compute the midpoint for each interval
  - Displace the midpoint at random but based on the difference between end points



RMD Algorithm

- Generate points a and b at random

$y_{mid} = (1/2)(y(a)+y(b))+r$

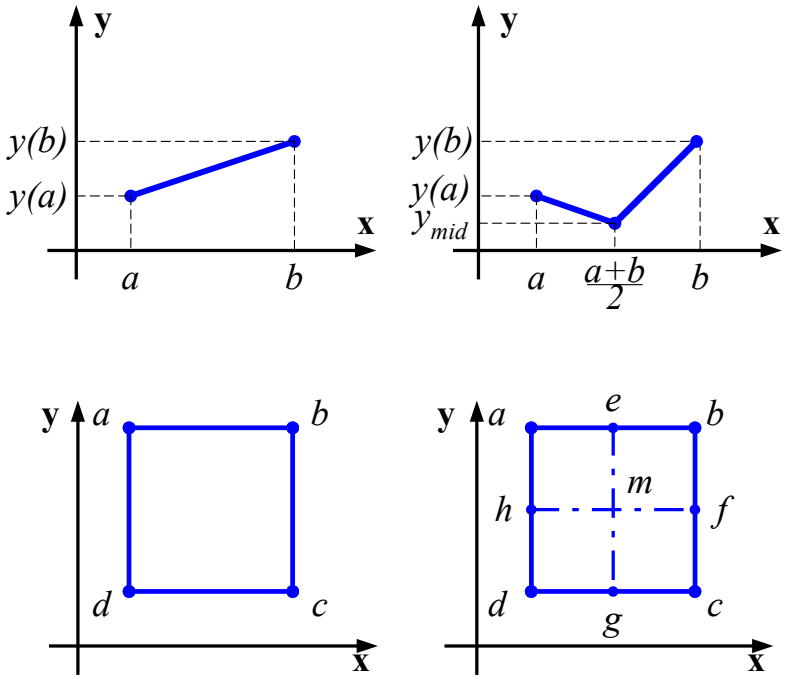
$r = s\cdot(|b-a|)\cdot r_g()$

s - is a surface roughness parameter

$r_g()$- returns a Gaussian random value (mean = 0, variance = 1)

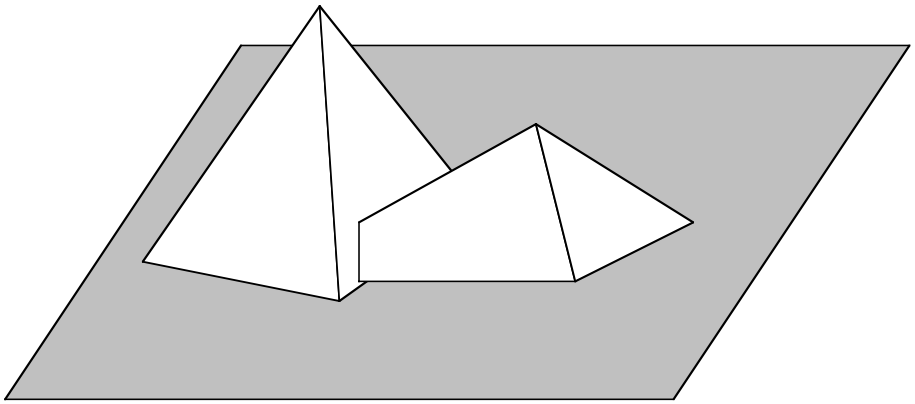
In 3D version:

$y_m = (1/4)(y(a)+y(b)+y(c)+y(d))+r$



Controlling the Topography

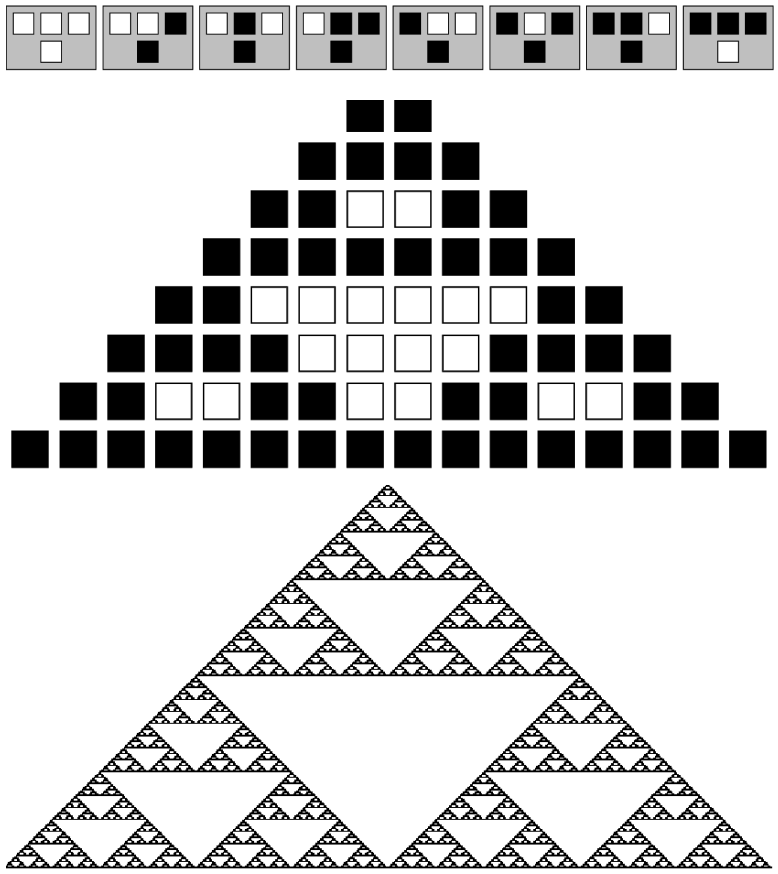
- To model real environments we want to control the placement of peaks and valleys
- We can do it by setting up *Control Surfaces*.
- Calculate random elevations based on:
  - Elevation of control surface
  - Average elevation of the midpoint



Fractal Mountains
by Ken Musgrave

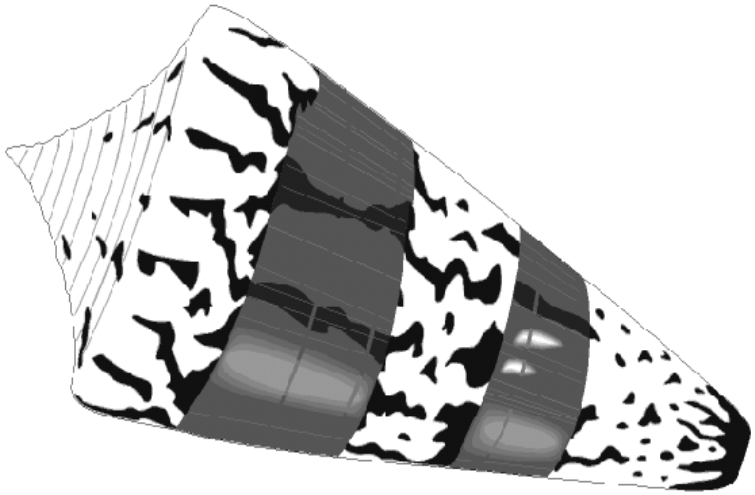
Cellular Automata

- Invented by John von Neumann in 1940s
- Thoroughly studied by Wolfram
- Mechanism to study reproduction
- Dynamic system
  - Discrete in space
  - Discrete in time



Compiled from Gary W. Flake "The Computational Beauty of Nature"

Cellular Automata in Nature



- Process which creates seashells has been linked to a one-dimensional CA

Compiled from Gary W. Flake "The Computational Beauty of Nature"

Conus Textile Seashell

Source: wikipedia.org