Binghamton University

EngiNet™

State University of New York

Thomas J. Watson

School of Engineering
and Applied Science

EngiNet™

WARNING

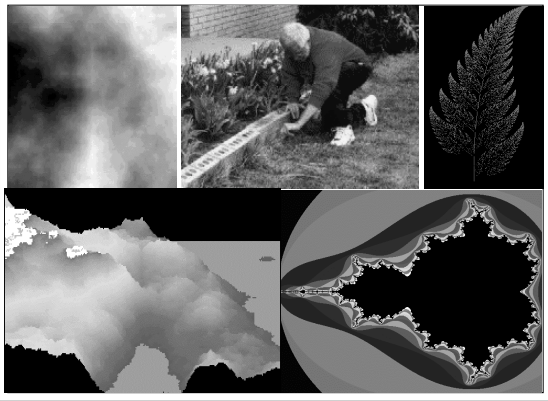
CS 460/560

Computer Graphics

Professor Richard Eckert

Lecture # 25

May 9, 2001



Lecture # 25

- Fractals

Fractals

- Mathematical Constructs
  - "Fractional" Dimension
- Lead to many techniques that can be used to model "natural" objects
- Lots of fractal generation programs
  - Example: FRACTINT, See: http://spanky.triumf.ca/www/fractint/fractint.html

Fractals

- Beautiful designs of infinite structure and complexity
- Qualities of Fractals:
  - Fractional dimension
  - Self similarity
  - Complex structure at all scales
  - Chaotic dynamical behavior
  - Simple generation algorithms
  - Capable of describing enormous range of natural objects

Some Objects Representable by Fractals

- Mountains
- Clouds
- Snow flakes
- Fog
- Frost patterns
- Fire
- River basins
- Sea coasts

- Explosions and fireworks
- Plants
- Island formations
- Galaxies
- Arteries and veins
- Cells
- Rivers
- Stock market fluctuations
- Weather systems
- Many More!!

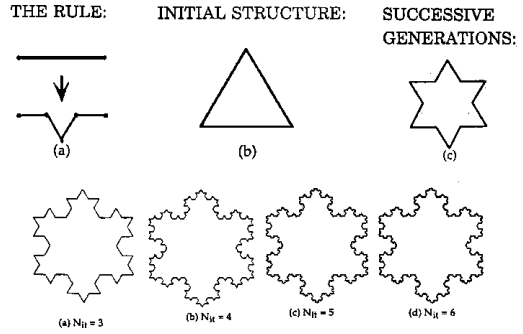
Types of Fractal-Generation Algorithms

- Linear Replacement Mapping
- Iterated Function Systems
- Random Midpoint Displacement
- Plasmas
- Escape-time algorithms
- Complex plane mapping
- Recursive, grammar-based systems
- Particle Systems

Linear Replacement Mapping

1. Define initial structure in terms of line endpoints
2. Define a replacement mapping
   - rule that replaces each line with a refined set of lines
   - defines next generation of the structure
   - inherently recursive
3. Iterate the refinement until desired level achieved

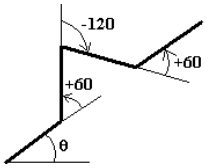
Example: Koch Snowflake



Implementing a Koch Curve

Assume recursive function Koch(len,theta,n)
(len = length, theta = angle of line, n=recursion level)
To get next generation curve from a line segment, make 4 calls:

```
Koch (len/3, theta, n-1);
theta += 60;
Koch (len/3, theta, n-1);
theta -= 120;
Koch (len/3, theta, n-1);
theta += 60;
Koch (len/3, theta, n-1);
```



Base case: At lowest (n=0) level of recursion, so draw line:
LineTo (len*cos(theta), len*sin(theta));

Using the Koch() function

- 1. Assign a value to n and initial position (x0,y0)
- 2. Make a call to MoveTo(x0, y0)
- 3. Assign an initial len, and theta
- 4. Make the call Koch (len, theta, n)

Iterated Function Systems

- Define a set of contractive affine transformation matrices

$$M_i = \begin{vmatrix} a_i & b_i & e_i \\ c_i & d_i & f_i \\ 0 & 0 & 1 \end{vmatrix}$$

Generate new points P'=(x',y') from old P=(x,y):

$$P' = M_i * P$$

i.e.:

$$x' = a_i*x + b_i*y + e_i$$
$$y' = c_i*x + d_i*y + f_i$$

The IFS Algorithm

Choose the set of ai,bi,ci,di,ei,fi
Select "seed point" (x,y)
Repeat many times:
- Pick an i randomly
- Compute x',y' from x,y using ai,bi,ci,di,ei,fi
- Plot (x',y') on screen
- Set (x,y) to (x',y')

Example: An IFS Fern

Matrix elements:

i	ai	bi	ci	di	ei	fi
1	0.00	0.00	0.00	0.16	0.0	0.0
2	0.85	0.04	-0.04	0.85	0.0	1.6
3	0.20	-0.26	0.23	0.22	0.0	1.6
4	-0.15	0.28	0.26	0.24	0.0	0.44

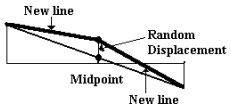
Result after 2000 iterations:

Result after 20,000 iterations:

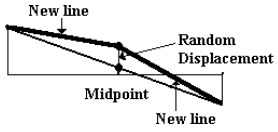
Result after 200,000 iterations

Random Midpoint Displacement

- Good for mountain silhouettes
- Recursive subdivision
- Start with a line segment
- Find midpoint (xm,ym)
- Displace ym by a random amount proportional to current length
- Repeat with each subdivision until sufficiently detailed
  - Repeat until we get to individual pixels
  - Store computed values of y in an array y[]



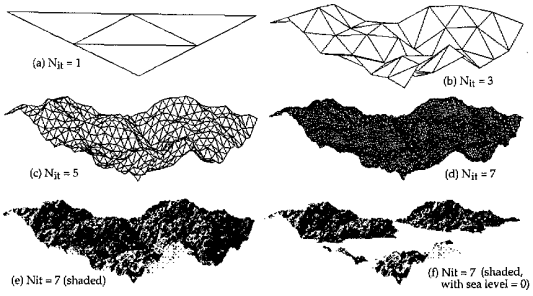
- Start endpoint coordinates: (x1,y1), (x2,y2)
- Assume we have a recursive procedure fracline(a,b)
  - Computes displaced midpoint line from x=a to x=b
  - Calls itself for each half of line
  - Repeat until y values for all pixels between endpoints are computed




```
int y[SCREEN_WIDTH);
float rug = 0.5;        // ruggedness factor
y[x1] = y1;  y[x2] = y2;  // line endpoints
fracline (x1,x2)        // fills y array values
for (x=x1; x<=x2; x++)
  SetPixel(x,y[x]);

fracline (a,b)
  {  if ((b-a) > 1)
     {  xmid = (a+b)/2;
        y[xmid] = (y[a]+y[b])/2 + rug*(b-a)*rand();
        fracline (a, xmid); fracline (xmid, b); }
  }
```

- Generalize to triangular surfaces in 3D
- Displace each triangle edge midpoint randomly in z
- --> Neat mountains!



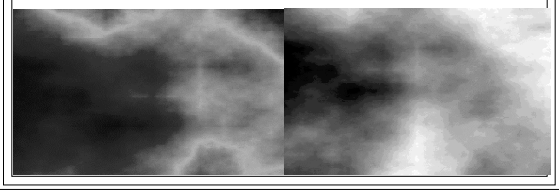
Plasmas

- Extension of random midpoint displacement
- Works with colors
- Great for generating clouds
- Easily generalized to give mountains

Plasma-generating Algorithm

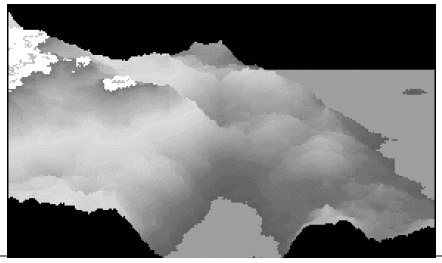
Set screen black
Set current rectangle to entire screen
Set each corner pixel of current rectangle to a random color
For each edge of current rectangle
  Compute color of midpoint P between edge's corner pixels by:
    1. Pick a random color C
    2. Compute weighting factor W proportional to distance between corner & P
    3. Set midpoint color to average of two corner colors and C weighted by W
Set center of current rectangle to average of 4 edge midpoint colors
Repeat recursively for each new rectangle determined by corner pixel and center pixel until all pixels are colored

- Key idea--at beginning, distances are large
  - So color of center pixel is mostly random
  - But as rectangles become smaller,
    random contribution is less...
    while neighbor pixel contribution is greater
  - So close points have similar colors
    - Like a cloud



Converting a Plasma to a Mountain

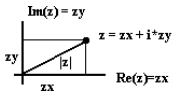
- Treat color code of each point as a height
- Plot the resulting surface
- (A cloud is a color-coded map of a mountain!)



Escape-Time Algorithms for Generating Fractals

- Give iterative rule for generating points in the complex plane
- Use "seed" points & determine if “orbit” of points generated by iterative rule is finite or escapes to infinity
- Map real (x) & imaginary (y) parts of each seed point to a pixel on screen
- Boundary between seed points whose orbits escape and those whose does not is often a very complex fractal

Example: Mandelbrot Set

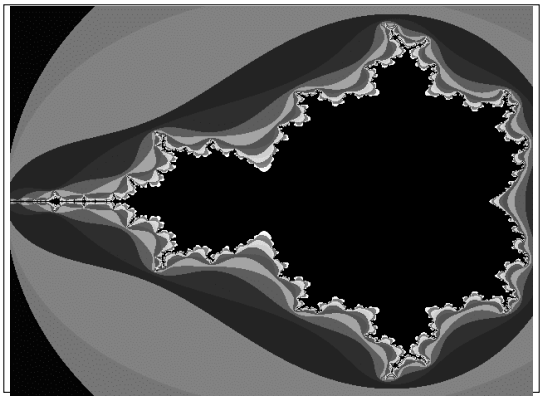


- Iteration rule: $z = z^2 + c$
- c is the seed point: $c = cx + i*cy$
- $z = zx + i*zy$ is each new complex point generated
- Start out with $z = (0,0)$
- By definition $z^2 = (zx^2 - zy^2, 2*zx*zy)$
- Square of radius of orbit: $|z|^2 = zx^2 + zy^2$
- If $|z| > 2$, orbit will escape to infinity
- Area of complex plane containing Mandelbrot set:
  $-2 < cx < 1.5$ and $-1.5 < cy < 0.5$

Mandelbrot Set Algorithm

- Simple algorithm to generate image of Mandelbrot set
- Points in Set are painted black
- Points outside set are painted white
- Can be generalized to paint in colors
  - Depending on how quickly outside points escape to infinity

```
Set N to some large maximum number of iterations
For y = 0 to SCREEN_HEIGHT
   For x = 0 to SCREEN_WIDTH
      Map (x,y) to (cx,cy)  // inverse 2D viewing transformation
      zx = 0; zy = 0; count = 0;
      While ( (zx*zx + zy*zy < 4) && (count < N) )
         count++;
         temp = zx*zx - zy*zy + cx;  // real part of new z
         zy = 2*zx*zy + cy;          // imaginary part of new z
         zx = temp;
      If (count < N)
         Setpixel(x,y,white);    // orbit escaped to infinity
      Else
         Setpixel(x,y,black);    // orbit did not escape in N iterations
```



Grammar-Based Systems (Lindemayer, L-Systems)

- Objects represented by strings of letters
  - Need an "Alphabet"
    - used to compose strings
  - Need an initial word ("Axiom")
    - successive generations of string derived from it
- "Productions" specify how new generations of objects are obtained
  - Give rewriting rules
    - applied in parallel to each letter in string

L-Systems in Computer Graphics

- Interpret each letter as a movement on screen (turtle graphics)
- Example alphabet with interpretation:
  - F: Go forward (trace a line)
  - +: Turn left by a given angle
  - -: Turn right by a given angle
  - many other possible movements

L-System for a Koch Curve

Alphabet:

F, +, -

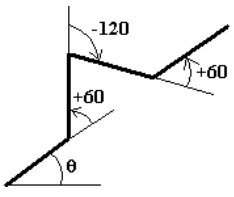
Take angle as 60

Axiom:

F

Production:

F -> F + F - - F + F



Deriving the System

F -> F + F - - F + F

Next iteration

(F+F--F+F) + (F+F--F+F) - - (F+F--F+F) + (F+F--F+F)

Successive iterations generate the Koch Curve

L-Systems can be extended in many ways

- Bracketed L-Systems
  - Good for modeling plants
  - Anything inside brackets is a branch
  - "[" means push onto stack (start branch)
  - "]" means pop from stack (end branch)
- Stochastic L-Systems
  - Apply productions probabilistically
- Lots of other variations

Particle Systems

- See
  - http://www.particlesystems.com
  - Dan Kutz's Particle Factory Simulator

Particle Systems

- Collections of particles that evolve over time
- Used to model systems whose time behavior is unpredictable
- Evolution determined by applying laws of physics to each particle
- Probabilistic effects easily included

Particles can:

- Be born and die
- Generate new particles
- Change their attributes
  - color, mass, etc.
- Move according to specified laws of motion
- Interact with their environment
- Interact with each other

Particles can model:

- Fire
- Clouds
- Fog
- Explosions
- Moving water
- Flocking birds
- Lots of other systems