CS 543: Computer Graphics

Fractals & Iterative Function Systems

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu

(with lots of help from Prof. Emmanuel Agu :-)

What are Fractals?

- Mathematical expressions
- Approach infinity in organized way
- Utilize recursion on computers
- Popularized by Benoit Mandelbrot (Yale University)
- Dimensionality
  - Line is one-dimensional
  - Plane is two-dimensional
  - Fractals fall somewhere in between
- Defined in terms of self-similarity

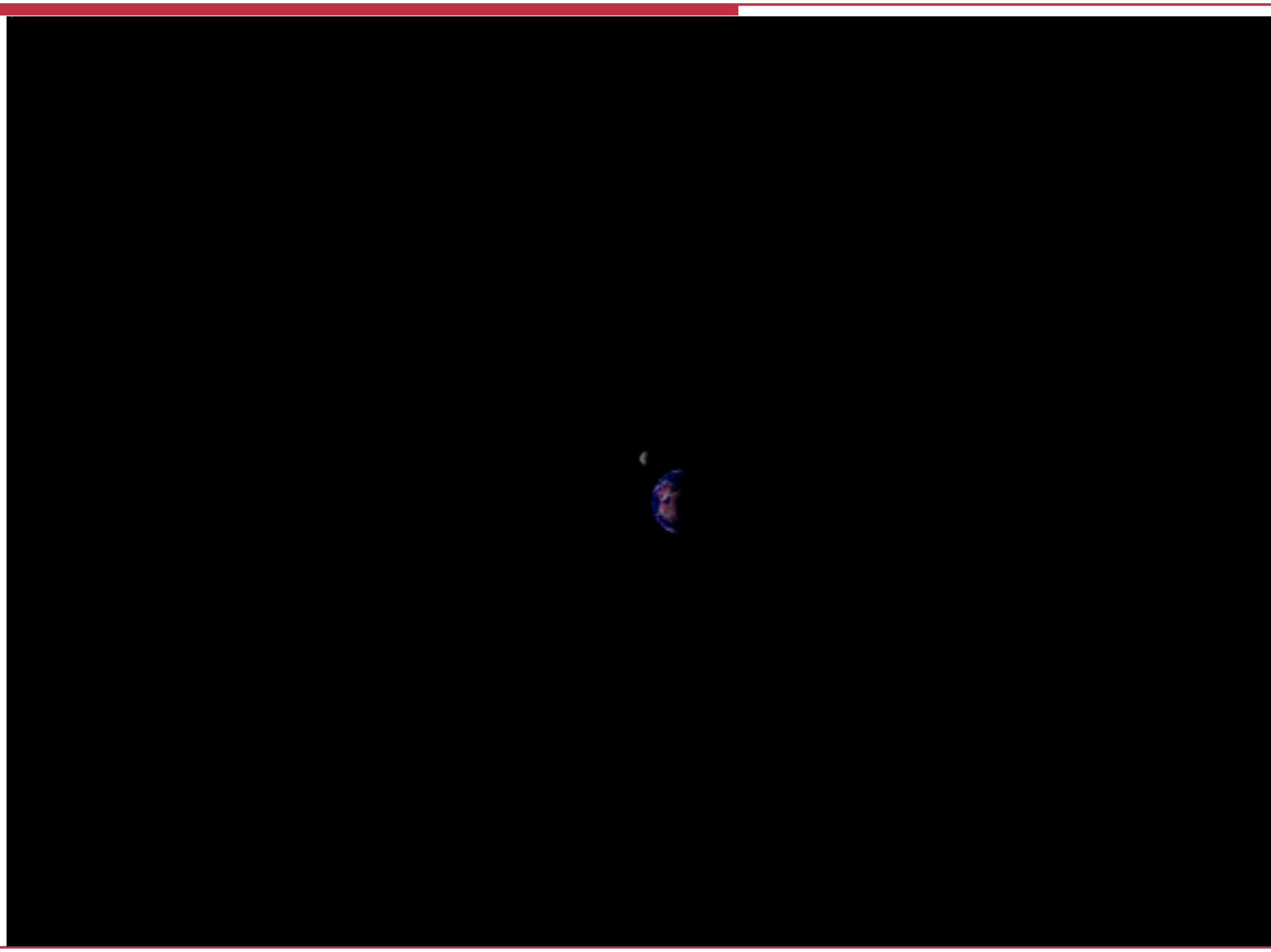
Self Similarity

- Level of detail remains the same as we zoom in
- Example
  - Surface roughness, or silhouette, of mountains is the same at many zoom levels
  - Difficult to determine scale
- Types or fractals
  - Exactly self-similar
  - Statistically self-similar

Examples of Fractals

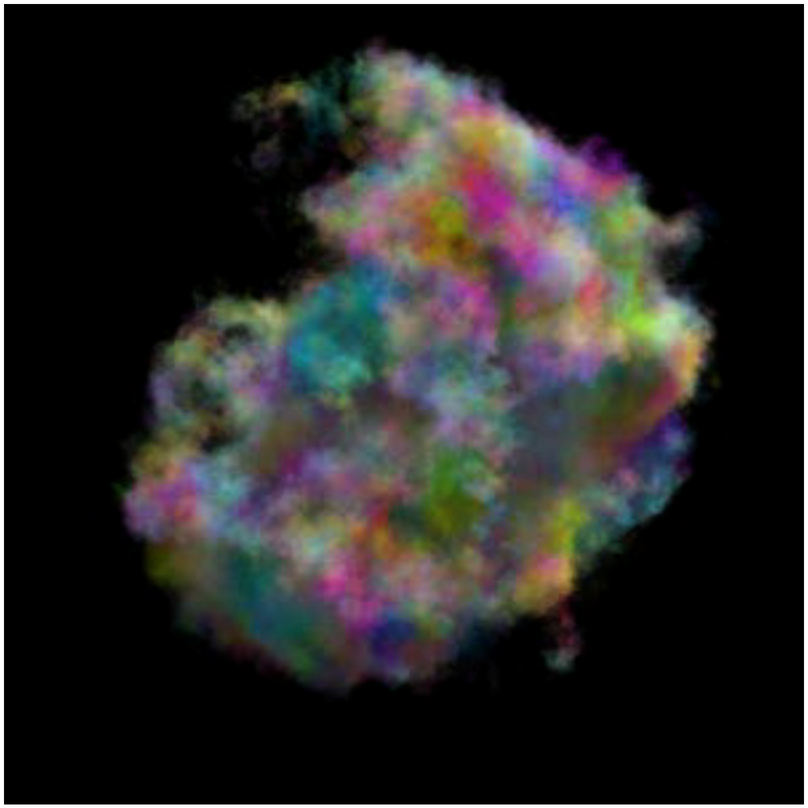
- Modeling mountains (terrain)
- Clouds
- Fire
- Branches of a tree
- Grass
- Coastlines
- Surface of a sponge
- Cracks in the pavement
- Designing antennae (www.fractenna.com)

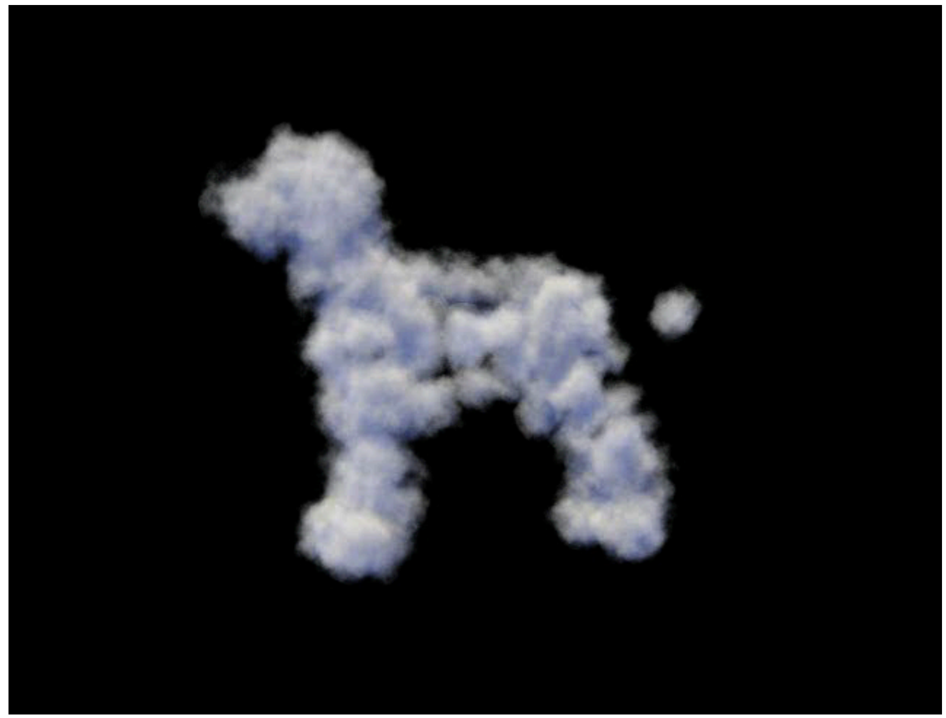
Examples of Fractals: Mountains



Images: www.kenmusgrave.com

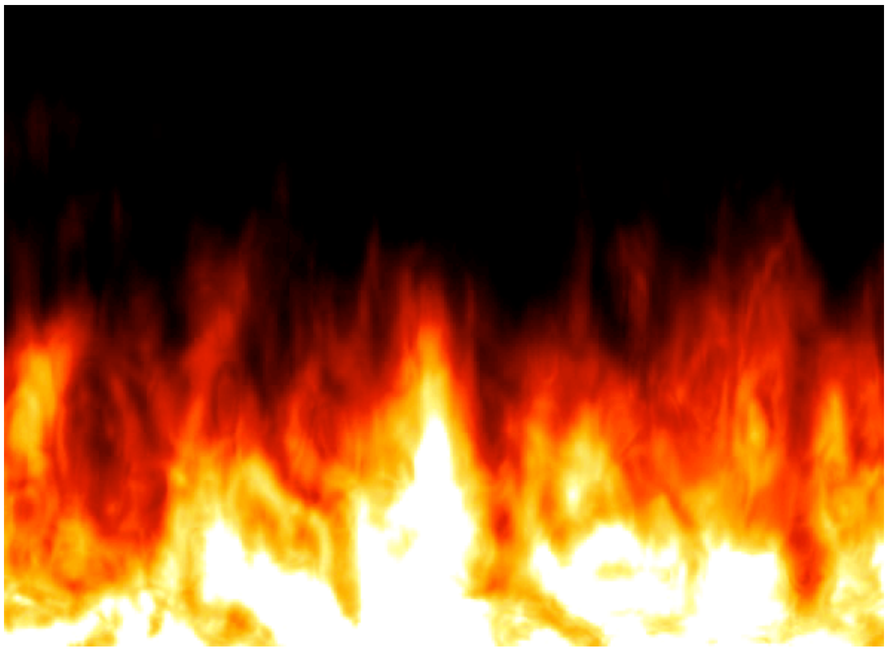
Examples of Fractals: Clouds





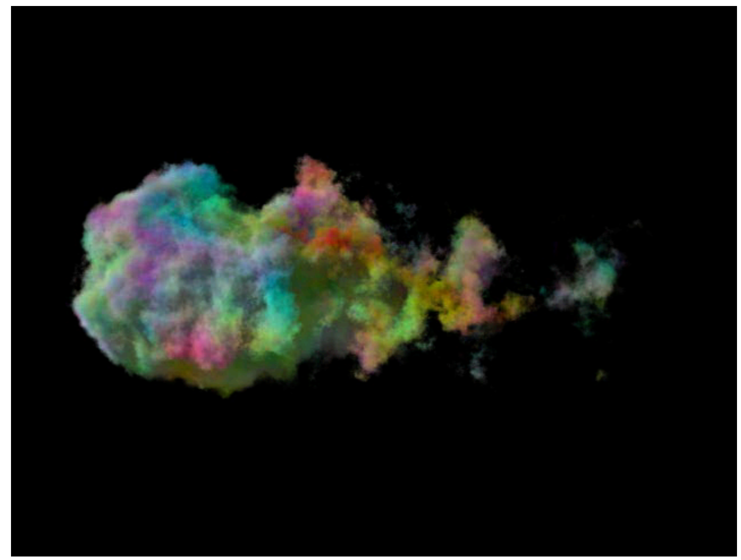
Images: www.kenmusgrave.com

Examples of Fractals: Fire



Images: www.kenmusgrave.com

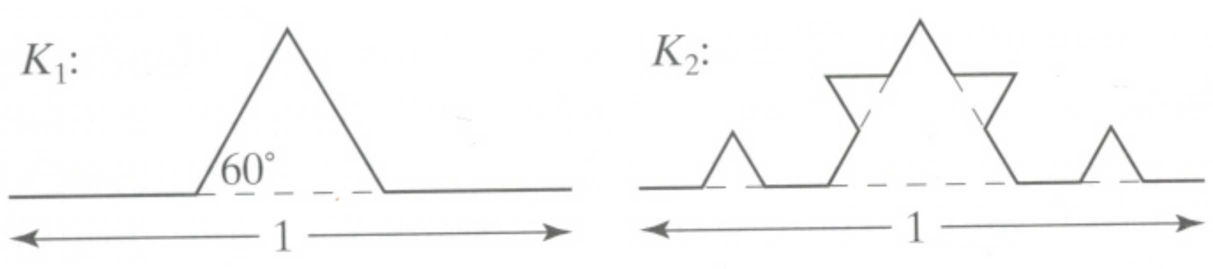
Examples of Fractals: Comets?



Images: www.kenmusgrave.com

Koch Curves

- Discovered in 1904 by Helge von Koch
- Start with straight line of length 1
- Recursively
  - Divide line into three equal parts
  - Replace middle section with triangular bump with sides of length 1/3
  - New length = 4/3

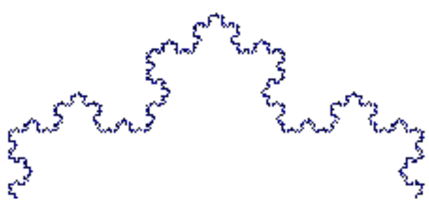


Koch Snowflake

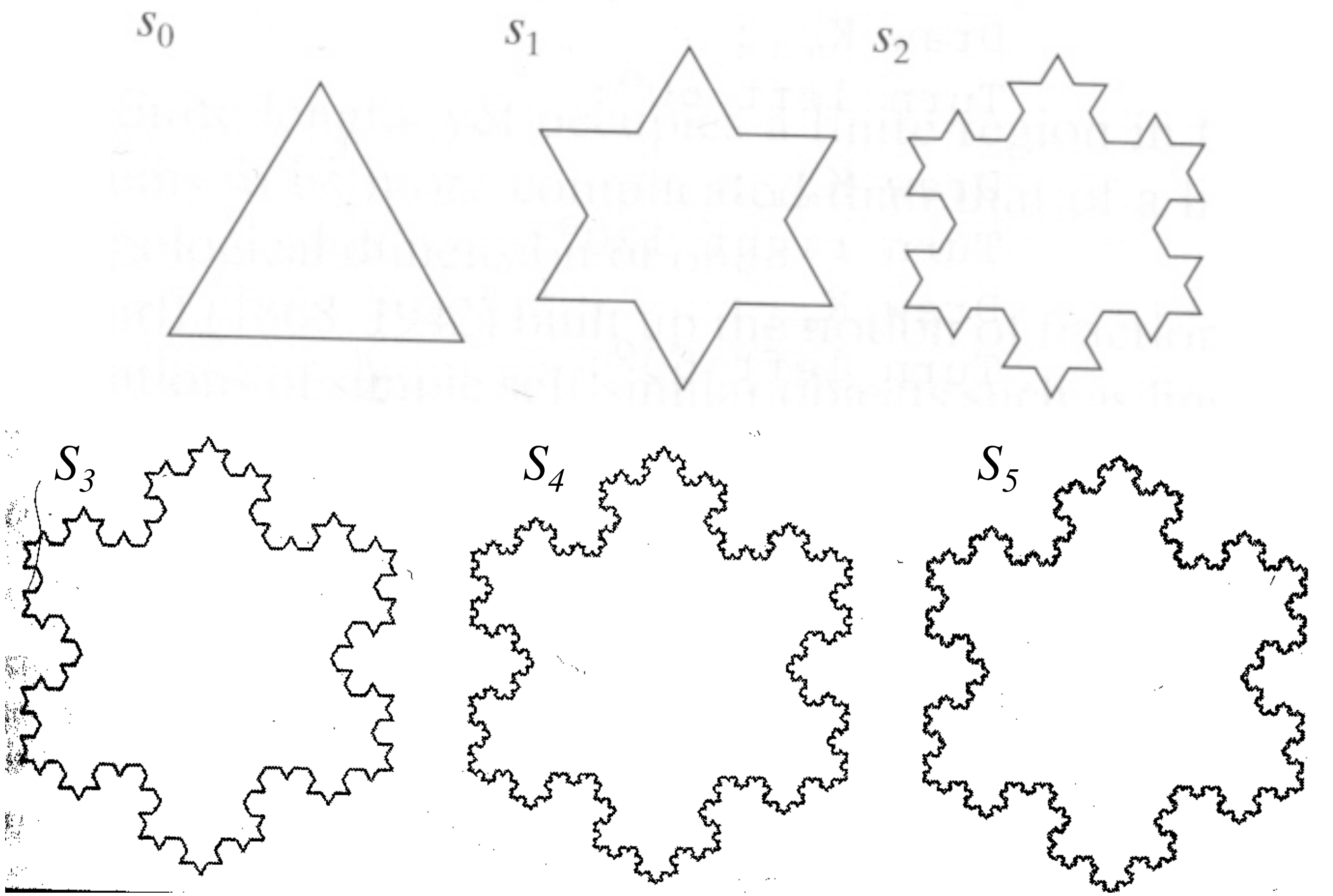
- Can form Koch snowflake by joining three Koch curves
- Perimeter of snowflake grows as:

$$P_i = 3\left(4/3\right)^i$$

  where P_i is the perimeter of the ith snowflake iteration
- However, area grows slowly as $S_\infty = 8/5$!
- Self similar
  - Zoom in on any portion
  - If n is large enough, shape is the same
  - On computer, smallest line segent > pixel spacing


www.jimloy.com

Koch Snowflake



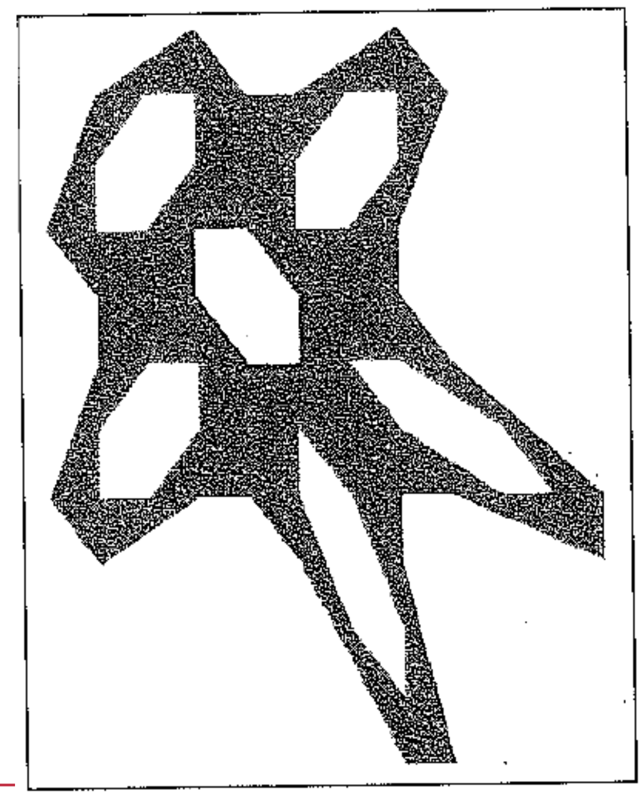
Psedocode to draw Koch Curve

```
if (n equals 0)  {
   draw straight line
} else  {
   Draw K_{n-1}
   Turn left 60°
   Draw K_{n-1}
   Turn right 120°
   Draw K_{n-1}
   Turn left 60°
   Draw K_{n-1}
}
```

Gingerbread Man

- Each new point **q** is formed from previous point **p** using the equation

$$q.x = M(1 + 2L) - p.y + |p.x - LM|;$$
$$q.y = p.x.$$



- For 640 x 480 display area,

  use M = 40   L = 3

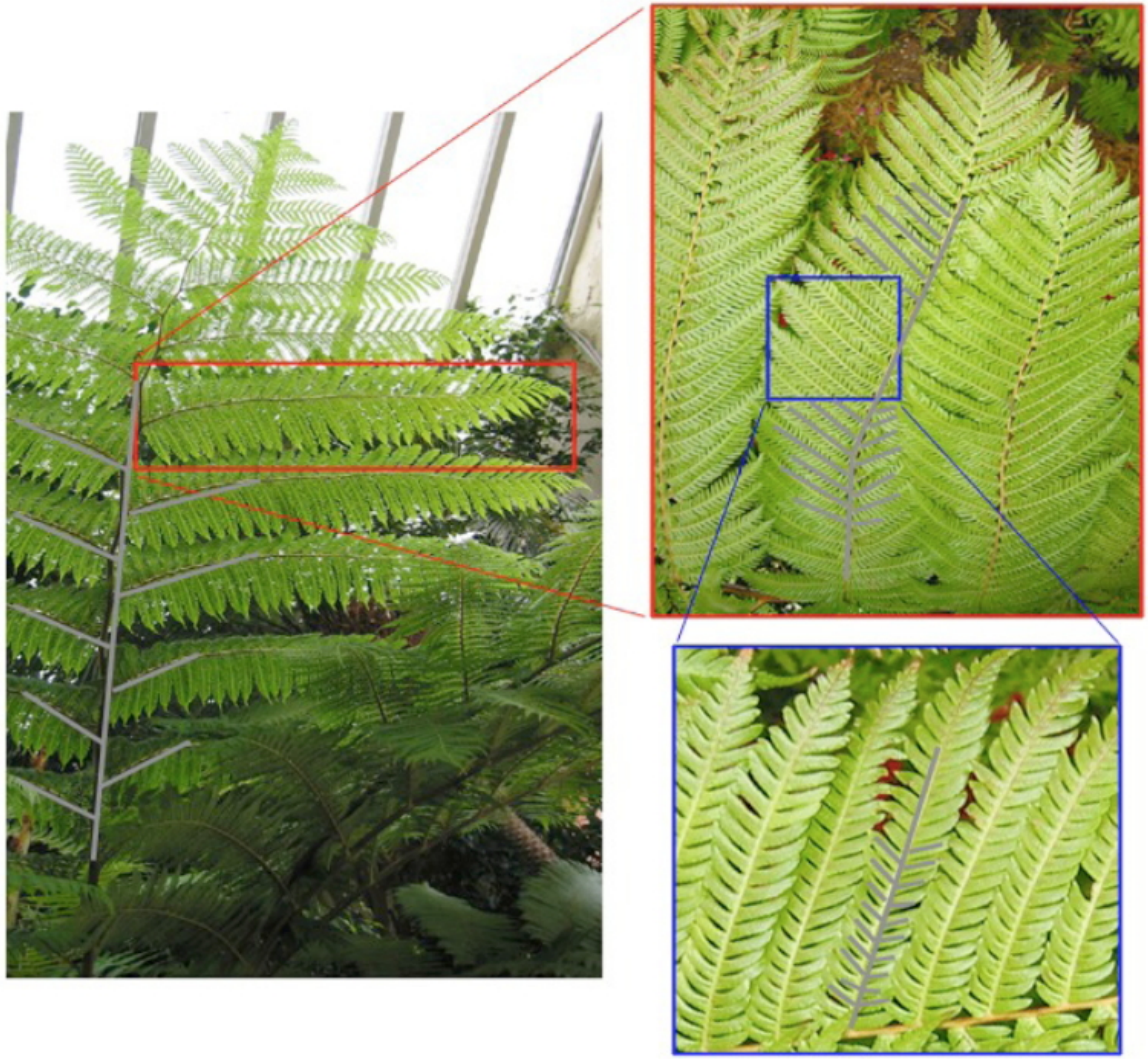
- A good starting point is

  (115, 121)

Iterated Function Systems (IFS)

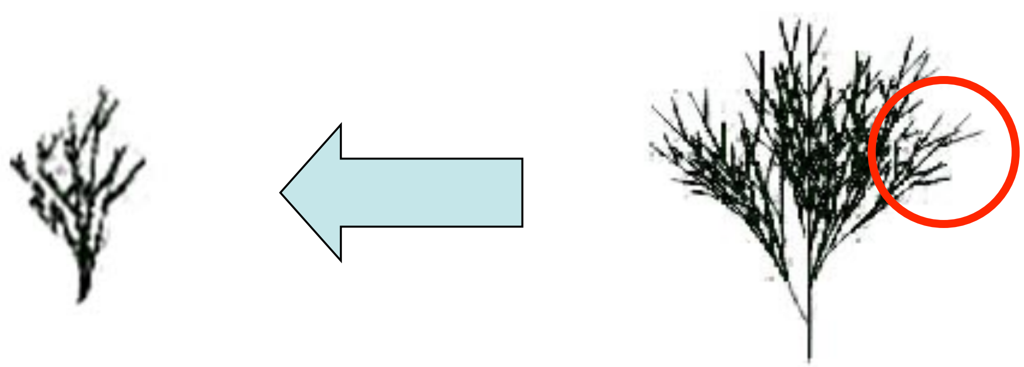
- Subdivide
- Recursively call a function
- Does result converge to an image? What image?
- IFS do converge to an image
- Examples
  - The Fern
  - The Mandelbrot set

Example: Ferns



Fractals and Self-Similarity

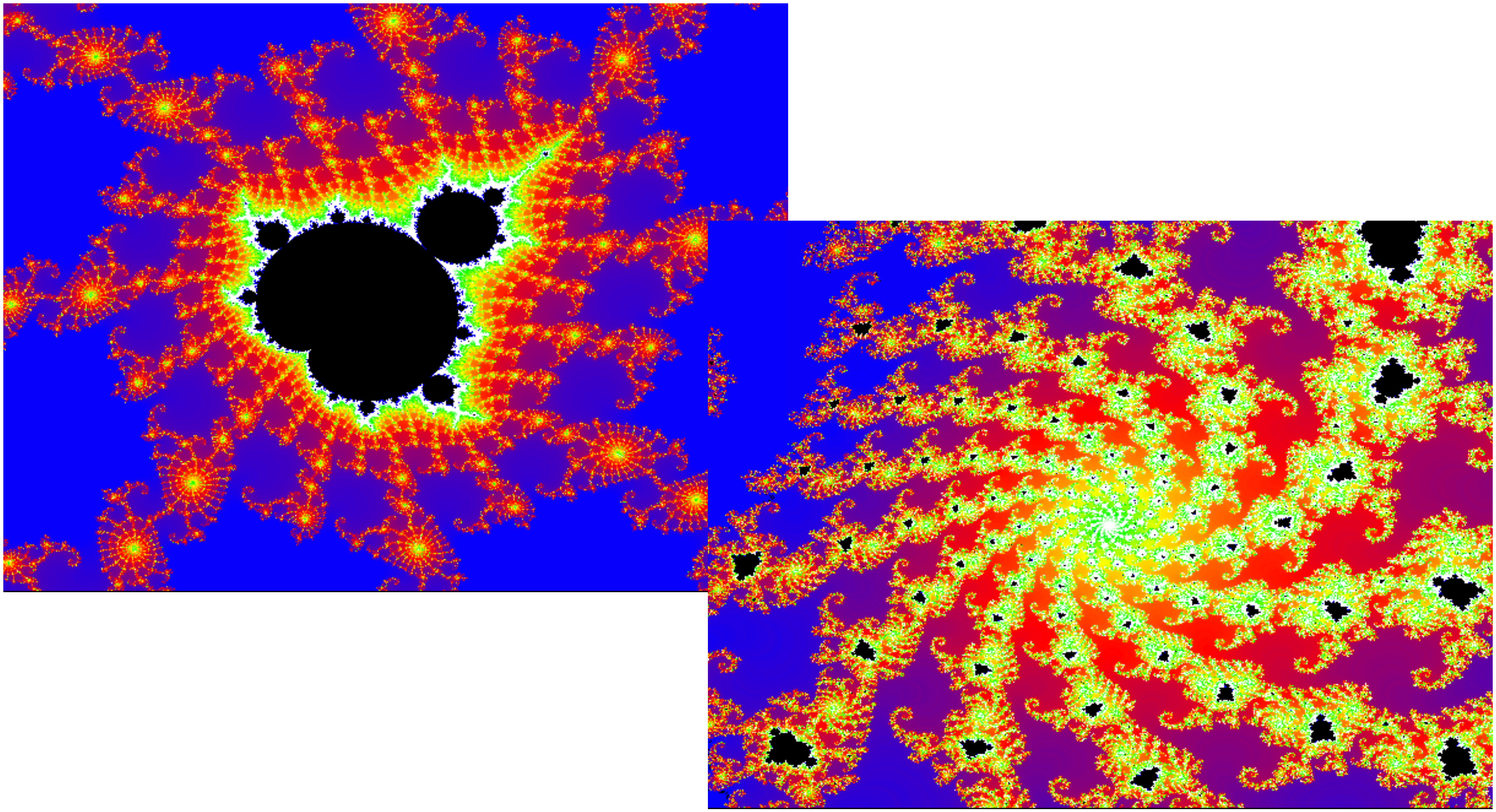
- **Exact Self-similarity**
  - Each small portion of the fractal is a reduced-scale replica of the whole (except for a possible rotation and shift).
- **Statistical Self-similarity**
  - The irregularities in the curve are statistically the same, no matter how many times the picture is enlarged.



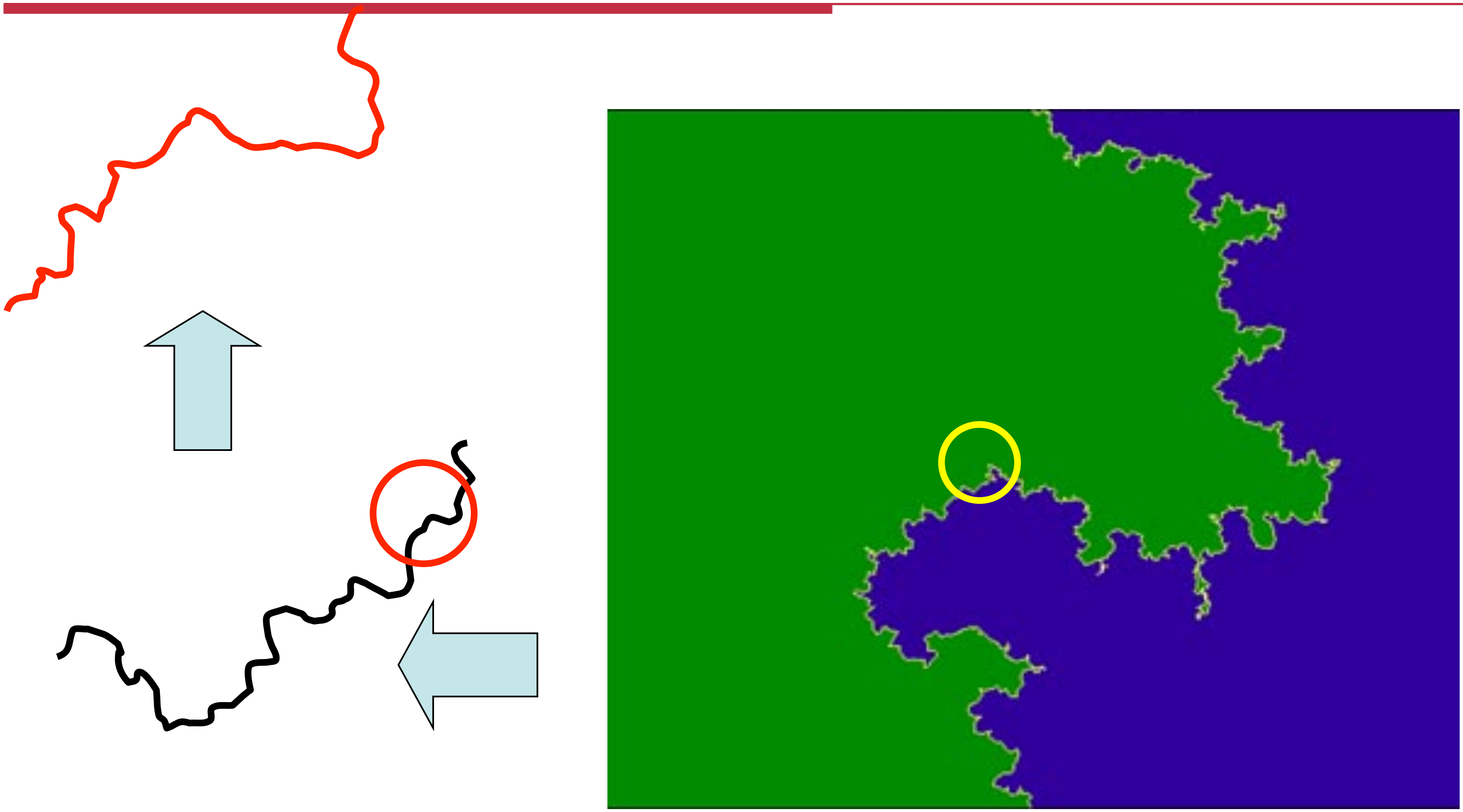
The Fern

- Any (sub) branch looks similar to any other (sub) branch
- Including ancestors and descendents

Mandelbrot Set

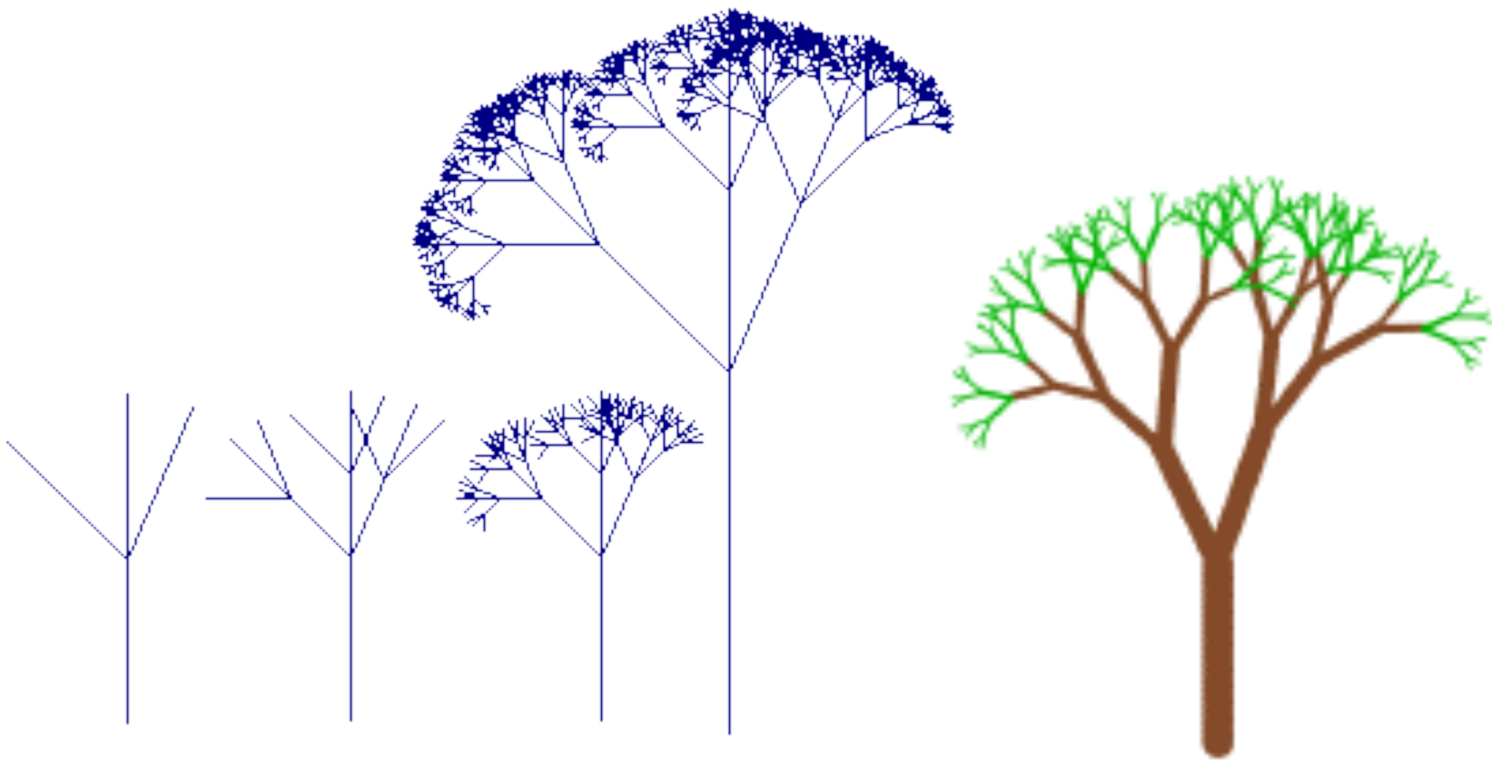


Fractal Coastline



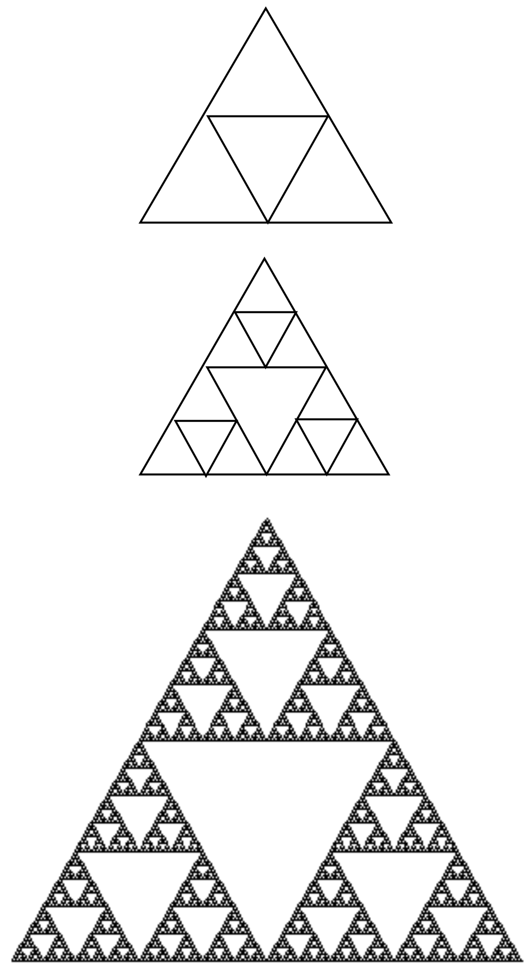
Examples of Fractals: Trees

Fractals appear “the same” at every scale.



Fractal Dimension – Eg. 2

The Sierpinski Triangle



$$D = \frac{\log N}{\log\left(\frac{1}{s}\right)}$$

$$N = 3,\ \ s = ½$$
$$\therefore D = 1.584$$

Space-Filling Curves

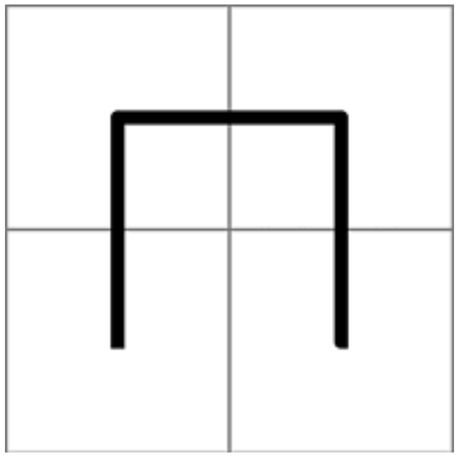
- There are fractal curves which completely fill up higher dimensional spaces such as squares or cubes.
- The space-filling curves are also known as Peano curves (Giuseppe Peano: 1858-1932).
- Space-filling curves in 2D have a fractal dimension 2.

You're not expected to be able to prove this.

Hilbert Curve

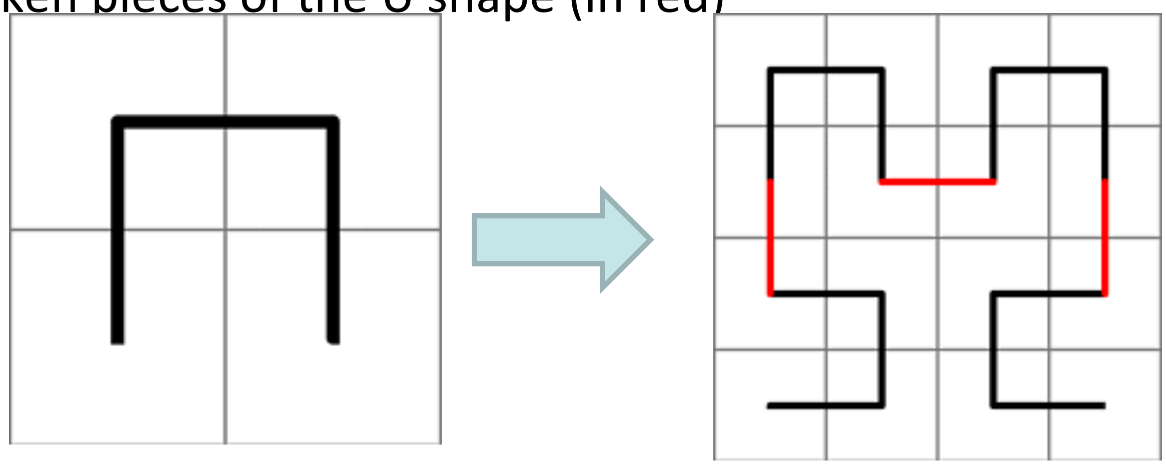
- Discovered by German Scientist, David Hilbert in late 1900s
- Space filling curve
- Drawn by connecting centers of 4 sub-squares, make up larger square.
- Iteration 0: 3 segments connect 4 centers in upside-down U

Iteration 0



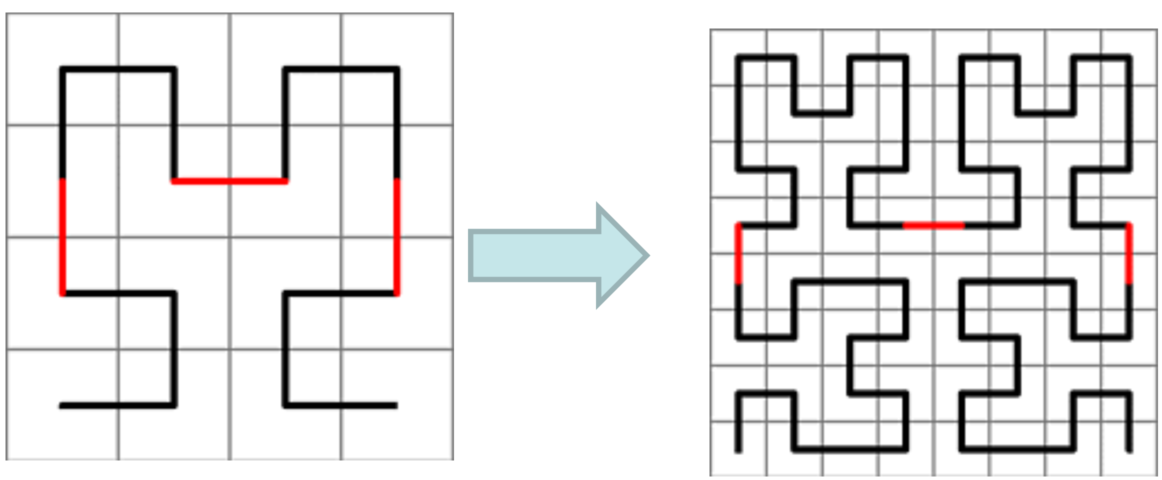
Hilbert Curve: Iteration 1

- Each of 4 squares divided into 4 more squares
- U shape shrunk to half its original size, copied into 4 sectors
- In top left, simply copied, top right: it's flipped vertically
- In the bottom left, rotated 90 degrees clockwise,
- Bottom right, rotated 90 degrees counter-clockwise.
- 4 pieces connected with 3 segments, each of which is same size as the shrunken pieces of the U shape (in red)

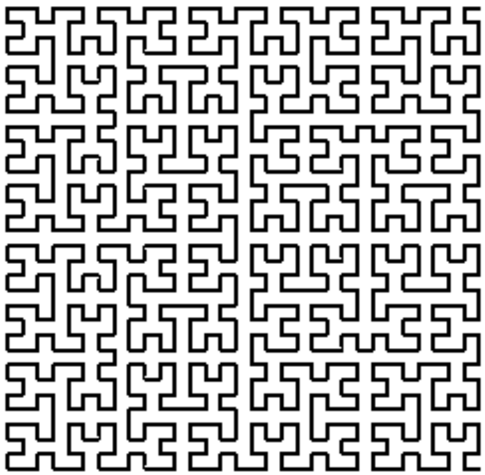


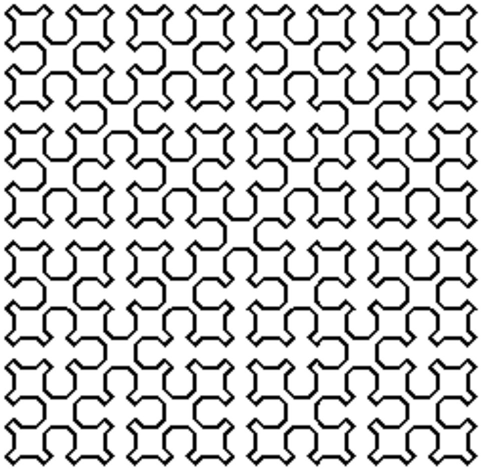
Hilbert Curve: Iteration 2

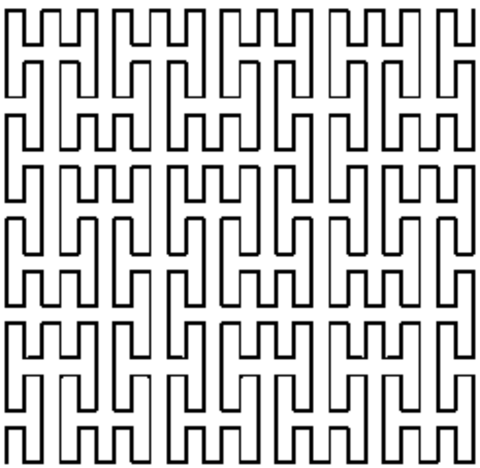
- Each of the 16 squares from iteration 1 divided into 4 squares
- Shape from iteration 1 shrunk and copied.
- 3 connecting segments (shown in red) are added to complete the curve.
- Implementation? Recursion is your friend!!

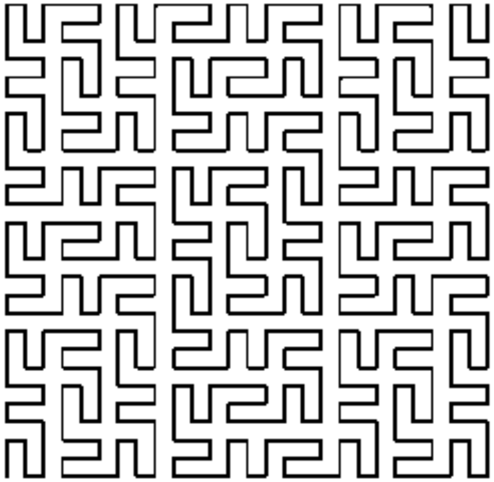


Space-Filling Curves

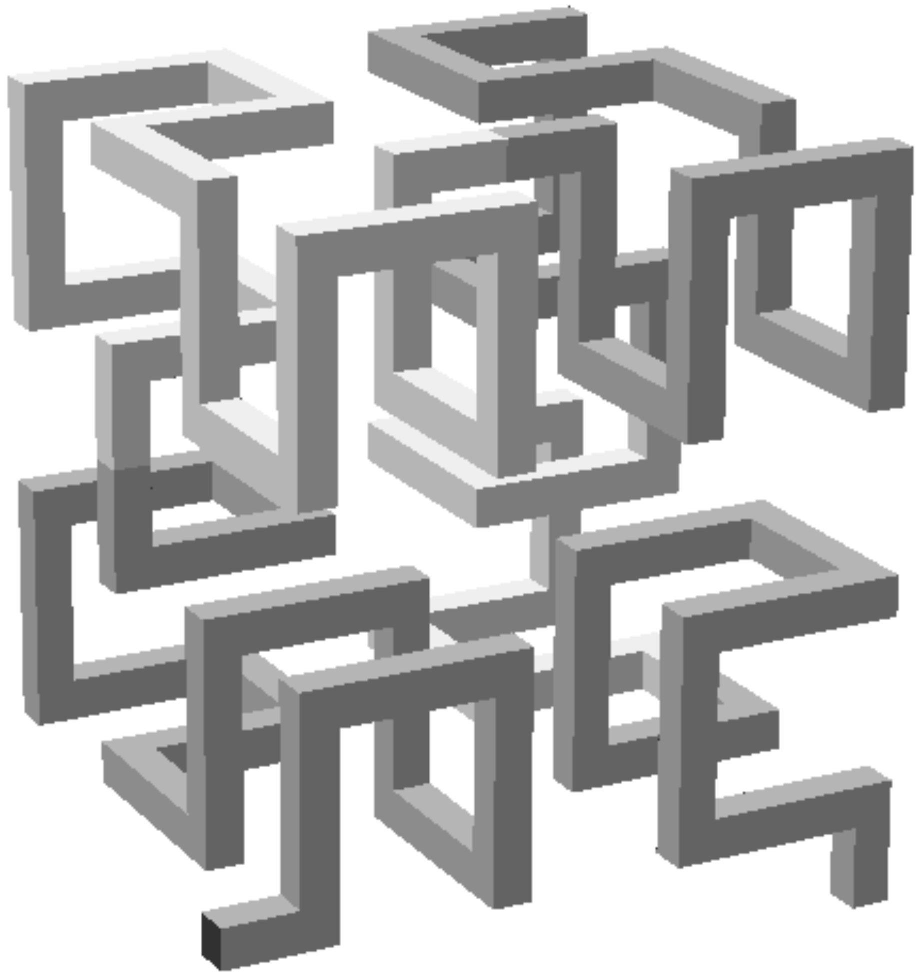








Space-Filling Curves in 3D



Generating Fractals

- Iterative/recursive subdivision techniques
- Grammar based systems (L-Systems)
  - Suitable for turtle graphics/vector devices
- Iterated Functions Systems (IFS)
  - Suitable for raster devices

L-Systems
(“Lindenmayer Systems”)

- A grammar-based model for generating simple fractal curves
  - Devised by biologist Aristid Lindenmayer for modeling cell growth
  - Particularly suited for rendering line drawings of fractal curves using turtle graphics
- Consists of a start string (*axiom*) and a set of *replacement rules*
  - At each iteration all replacement rules are applied to the string in parallel
- Common symbols:
  - F Move forward one unit in the current direction.
  - \+ Turn right through an angle *A*.
  - \- Turn left through an angle *A*.

The Koch Curve

Axiom: F (the zeroth order Koch curve)

Rule: F $\rightarrow$ F-F++F-F

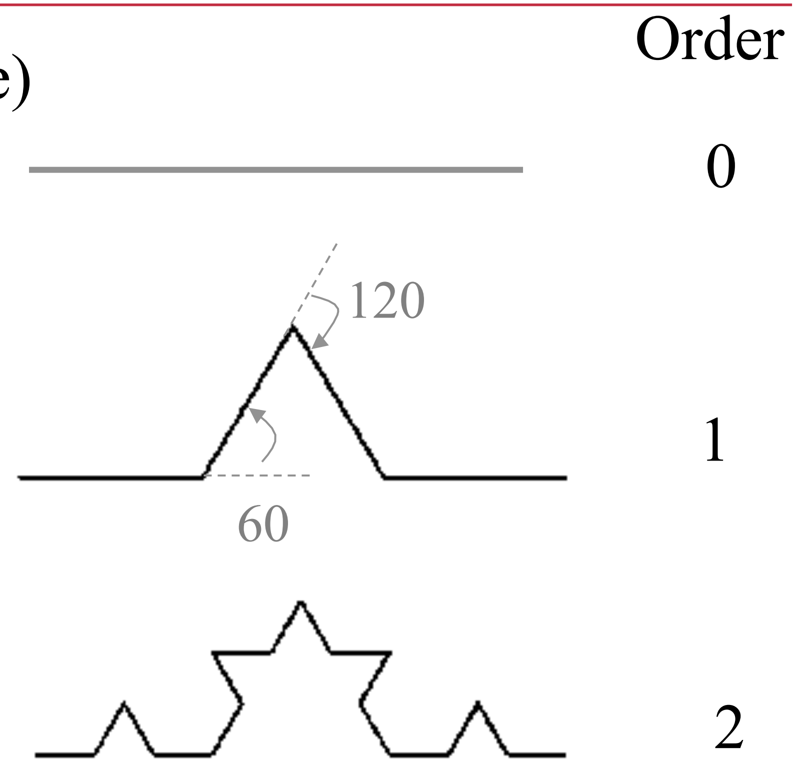
Angle: 60°

First order:

F-F++F-F

Second order:

F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F



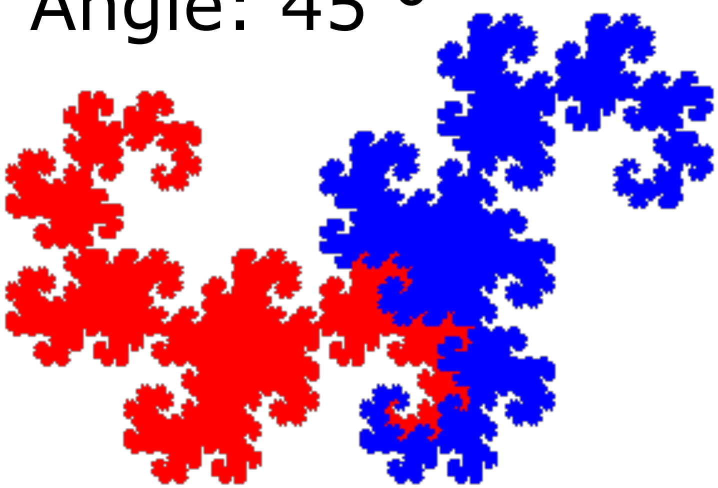
The Dragon Curve

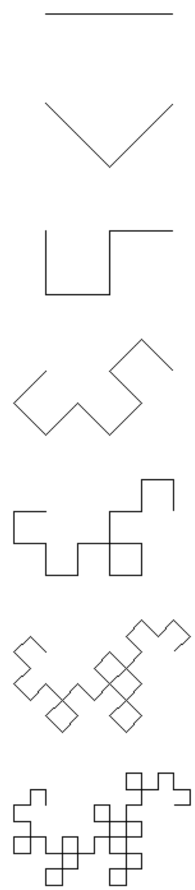
Axiom: FX

Rules:

- $F \rightarrow \varnothing$
- $X \rightarrow +FX--FY+$
- $Y \rightarrow -FX++FY-$

Angle: 45 °





At each step, replace a straight segment with a right angled elbow.

Alternate right and left elbows.

FX and FY are "embryonic" right and left elbows respectively.

L-System code

```
import turtle
turtle.speed(0) # Max speed (still horribly slow)

def draw(start, rules, angle, step, maxDepth):
    for char in start:
        if maxDepth == 0:
            if   char == 'F': turtle.forward(step)
            elif char == '-': turtle.left(angle)
            elif char == '+': turtle.right(angle)
        else:
            if char in rules:  # rules is a dictionary
                char = rules[char]
            draw(char, rules, angle, step, maxDepth-1)
# Dragon example:
draw("FX",{'F':"",'X':"+FX--FY+",'Y':"-FX++FY-"}, 45, 5, 10)
```

Generalized Grammars

- The grammar rules in L-systems can be further generalized to provide the capability of drawing branchlike figures, rather than just continuous curves.
- The symbol **[** is used to store the current state of the turtle (position and direction) in a stack for later use.
- The symbol **]** is used to perform a pop operation on the stack to restore the turtle's state to a previously stored value.

Generalized Grammars

Fractal bush:

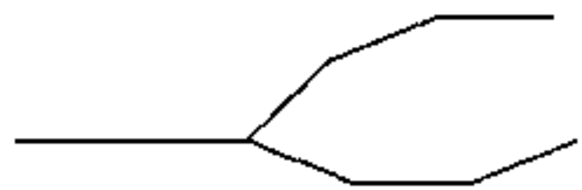
S → F

F → FF-[-F+F+F]+[+F-F-F]

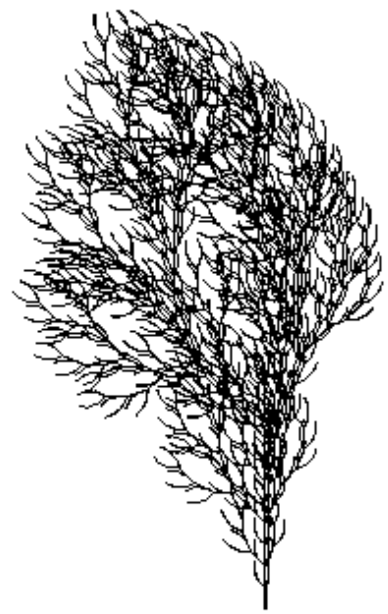
(A = 22 degs.)

Zero order bush

F



First order bush



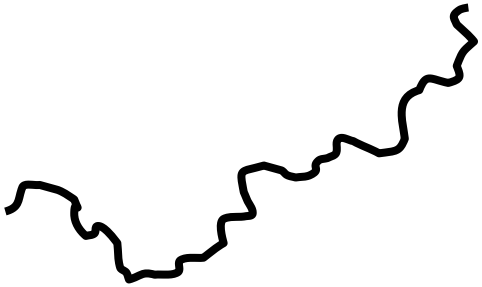
Fourth order bush
(with 90 deg. rotation)

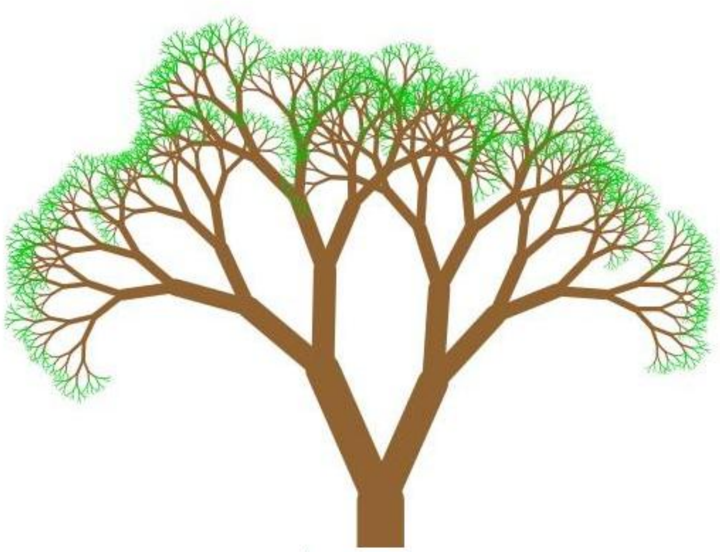
Random Fractals

- Natural objects do not contain identical scaled down copies within themselves and so are not exact fractals.
- Practically every example observed involves what appears to be some element of randomness, perhaps due to the interactions of very many small parts of the process.
- Almost all algorithms for generating fractal landscapes effectively add random irregularities to the surface at smaller and smaller scales.

Random Fractals

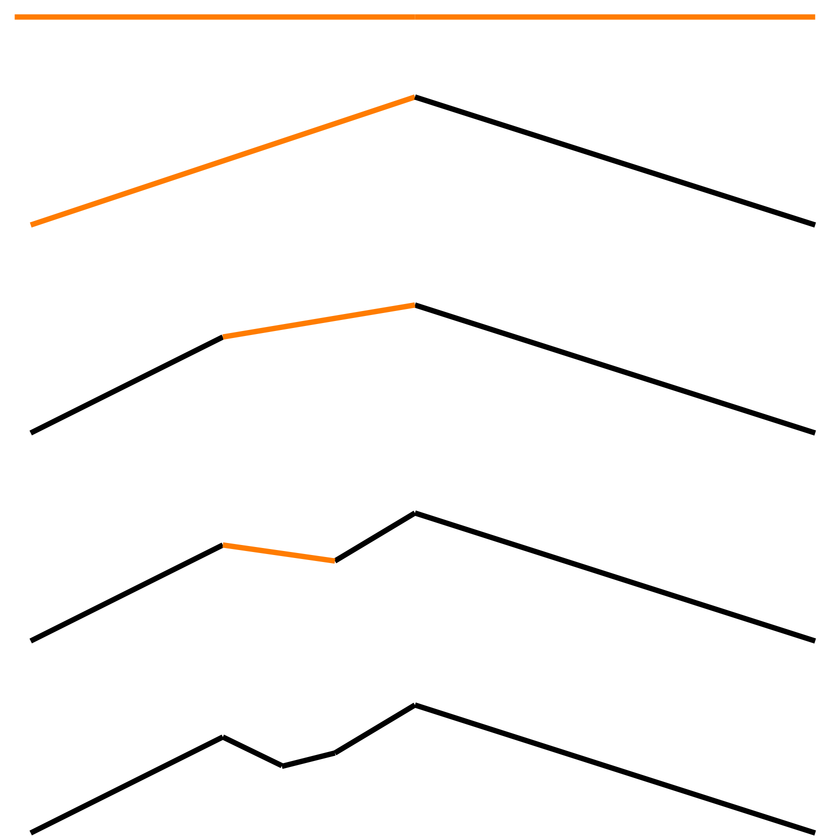
- Random fractals are
  - randomly generated curves that exhibit self-similarity, or
  - deterministic fractals modified using random variables
- Random fractals are used to model many natural shapes such as trees, clouds, and mountains.





IFS Example: Generating Fractal Terrain (2D)

1. Choose a random-number range
2. Start with a line
3. Find the midpoint
4. Displace it in *y* by a random amount
5. Reduce the range of your random numbers
   - Controls roughness
6. Recurse on both new segments

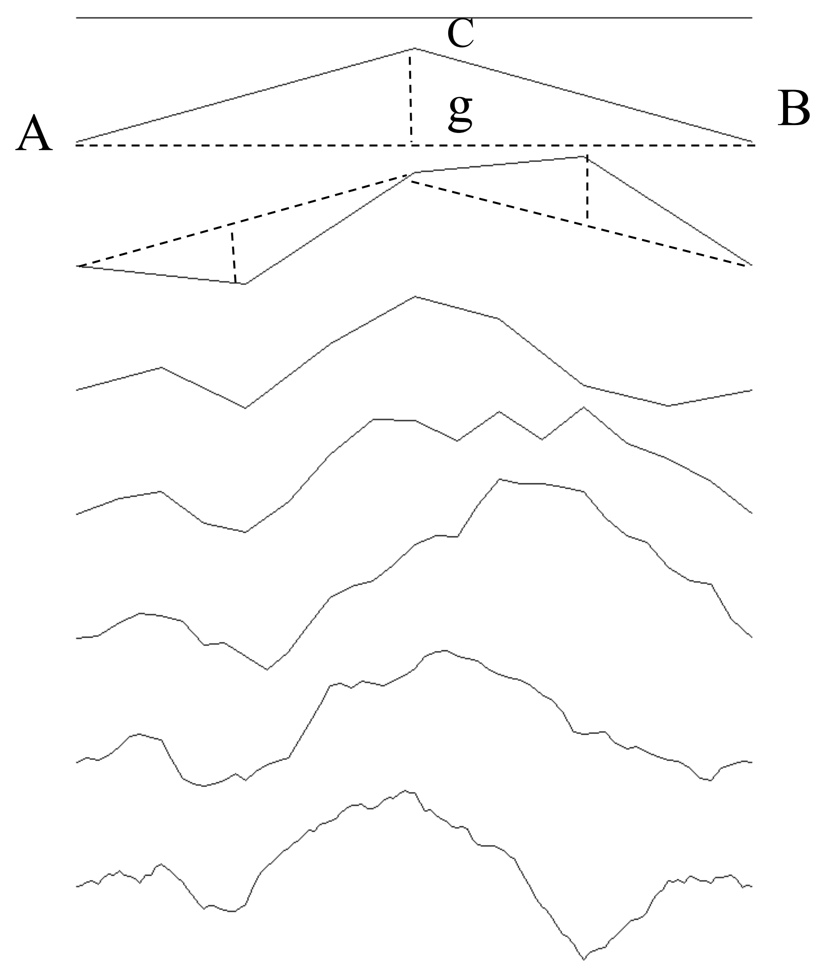


Random Midpoint Displacement Algorithm (2D)

- Subdivide a line segment into two parts, by displacing the midpoint by a random amount "g". *i.e.*, y-coordinate of C is

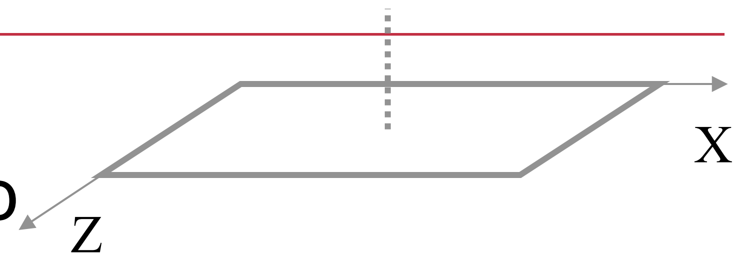
$$y_C = (y_A + y_B)/2 + g$$

  - Generate *g* using a Gaussian random variable with zero mean (allowing negative values) and standard deviation s.
- Recurse on each new part
  - At each level of recursion, the standard deviation is scaled by a factor $(1/2)^H$
    - H is a constant between 0 and 1
    - H = 1 in the example on the right



Midpoint Displacement Algorithm (3D)

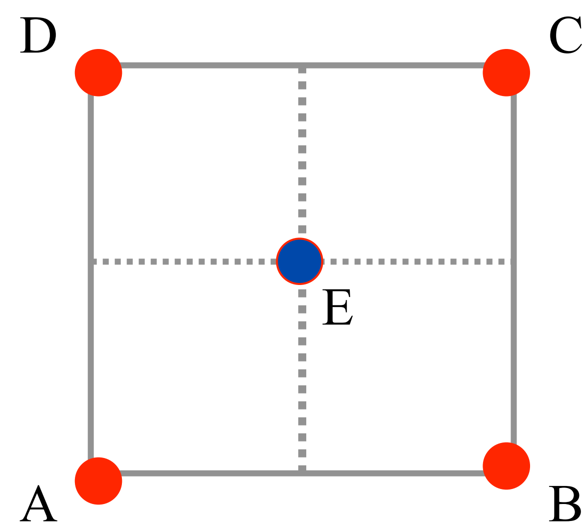
Square-Step:



Subdivide a ground square into four parts, by displacing the midpoint by a Gaussian random variable *g* with mean 0, std dev *s*.

i.e., Compute y-coordinate of E as

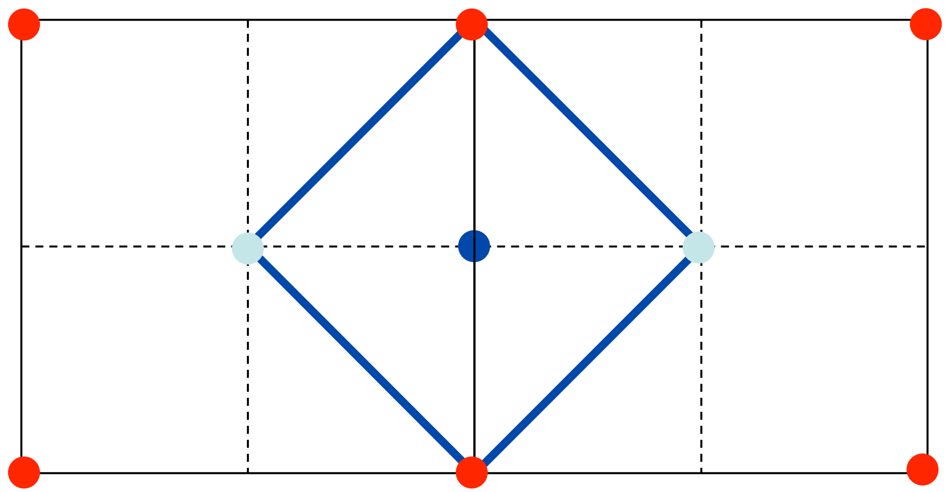
$y_E = (y_A + y_B + y_C + y_D)/4 + g$



Do that for all squares in the grid (only 1 square for the first iteration). Then ...

Diamond step

- To get back to a regular grid, we now need new vertices at all the edge mid-points too.
- For this we use a *diamond step*:



Vertices before square step

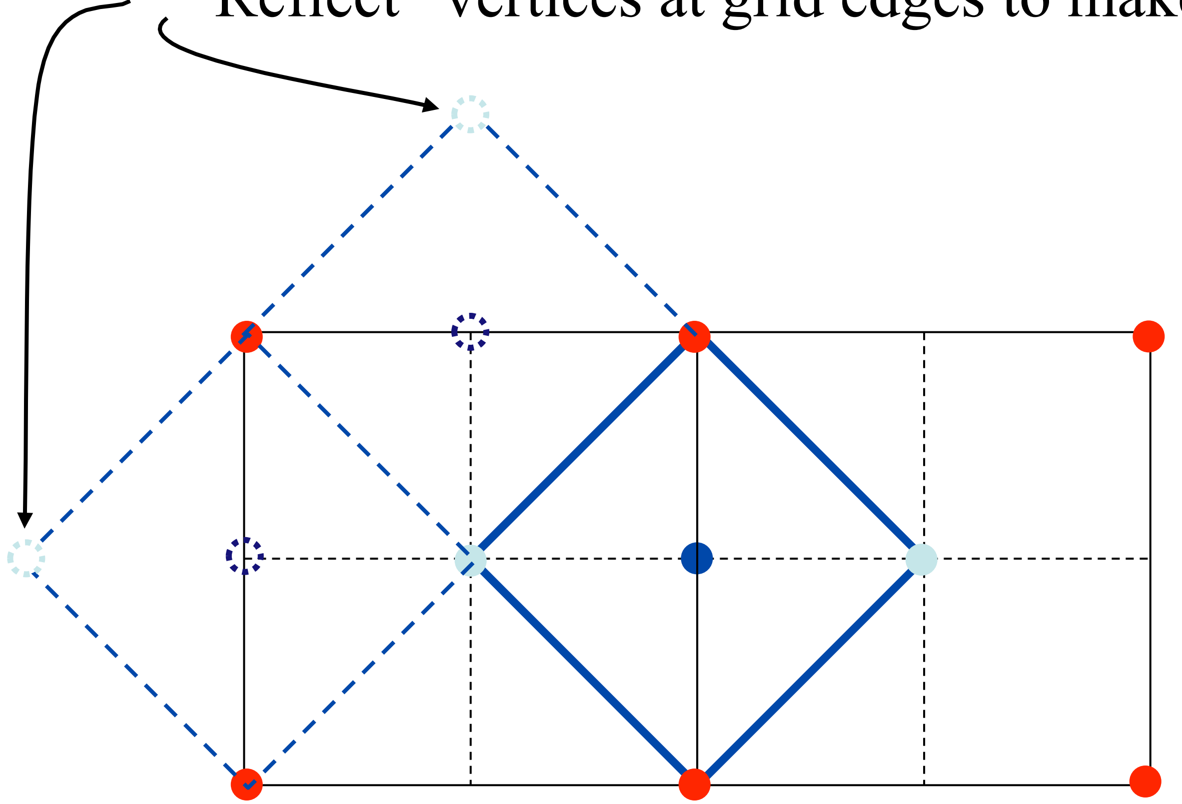
New vertices from square step

Vertex from diamond step
(on an old edge midpoint).
Computed as in square step but
using the 4 diamond vertices.

Do this for all edges (i.e., all possible diamonds).

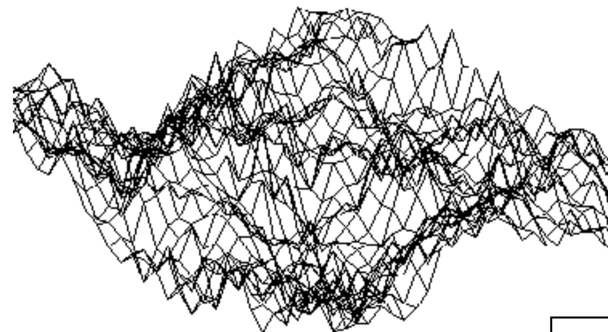
Diamond step (cont'd)

"Reflect" vertices at grid edges to make diamonds there.

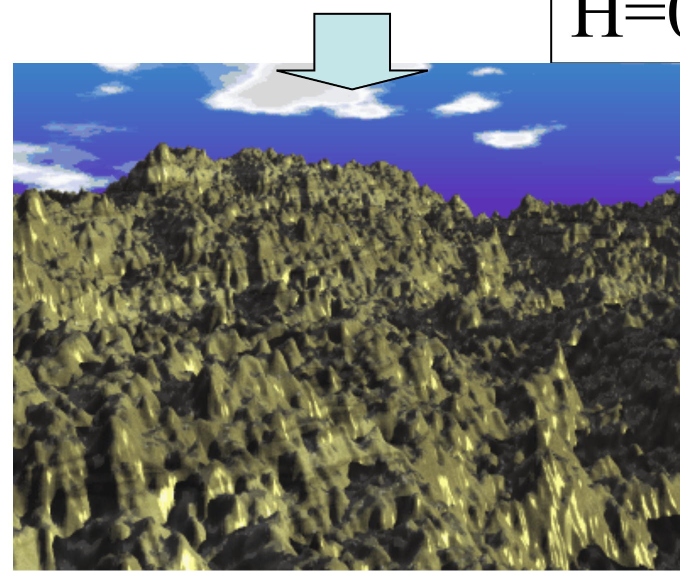


Diamond-Square Algorithm

The above two steps are repeated for the new mesh, after scaling the standard deviation of g by $(1/2)^H$. And so on …

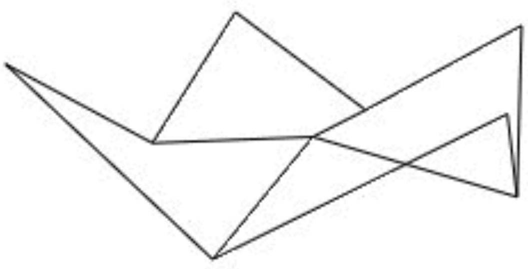


H=0.4

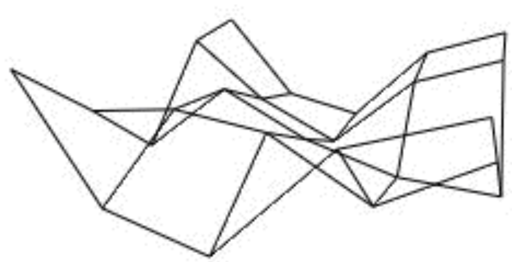


H=0.8

Diamond Step Process



1st pass



2nd pass

5th pass

Height Maps

- The 2D height map obtained using the diamond-square algorithm can be used to generate fractal clouds.
- Use the y value to generate opacity.

Useful Links

- Terragen – terrain generator
  - http://www.planetside.co.uk/terragen/
- Generating Random Fractal Terrain
  - http://www.gameprogrammer.com/fractal.html
- Lighthouse 3D OpenGL Terrain Tutorial
  - http://www.lighthouse3d.com/opengl/terrain/
- Book about Procedural Content Generation
  - Noor Shaker, Julian Togelius, Mark J. Nelson, ***Procedural Content Generation in Games: A Textbook and an Overview of Current Research*** (Springer), 2014.
- Book about Procedural Generation

  David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steve Worley. ***Texturing and Modeling: A Procedural Approach*** (The Morgan Kaufmann Series in Computer Graphics)

References

- Angel and Shreiner, Interactive Computer Graphics, 6th edition, Chapter 9
- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition, Appendix 4