IMGD 4000
Technical Game Development II
Procedural Content Generation

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Human Interaction in Virtual Environments (HIVE) Lab
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu

Procedural Content Generation

- The algorithmic creation of game content with limited or indirect user input[1]

  or

- Computer software that can create game content on its own, or together with one or many human players or designers[1]

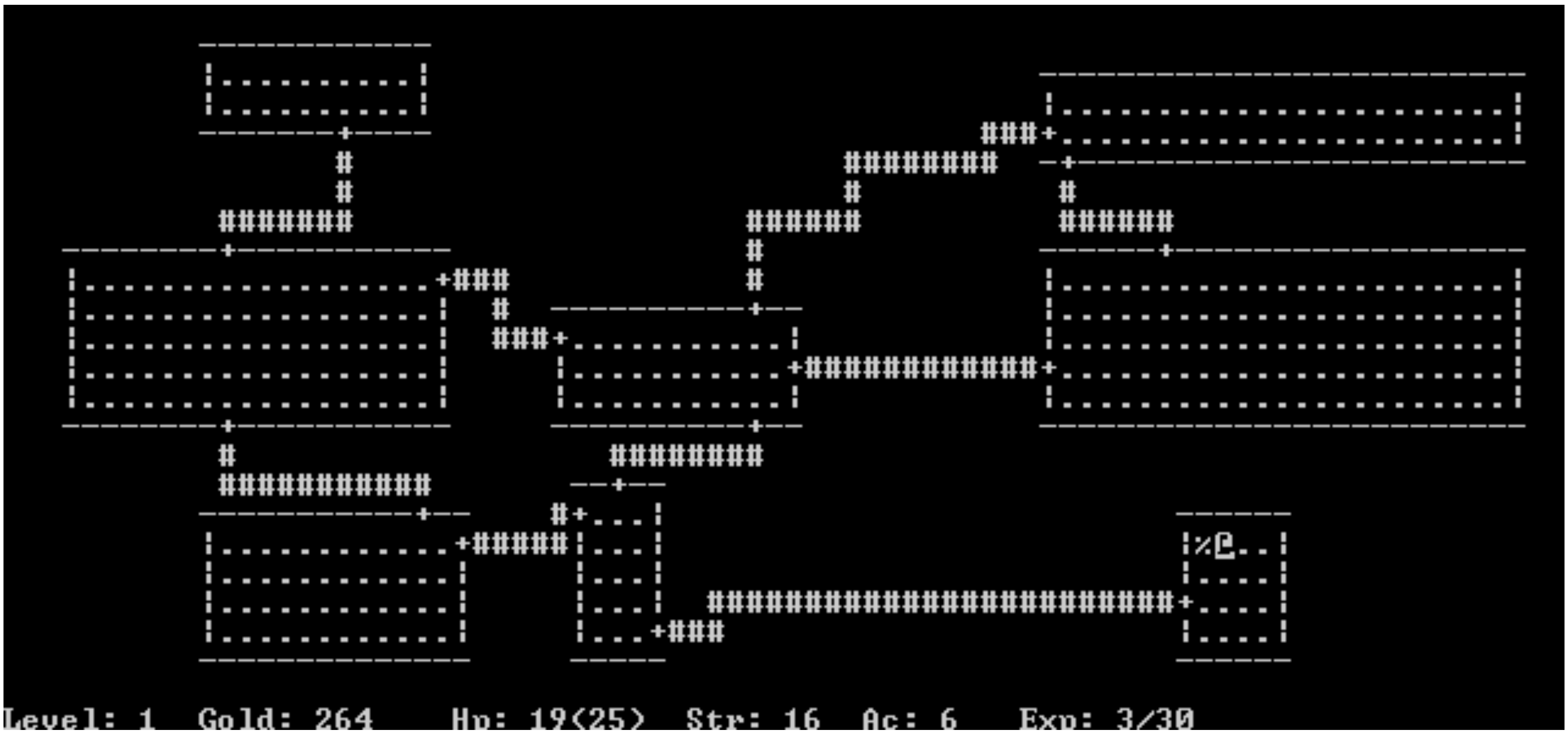
[1]Togelius, J., Kastbjerg, E., Schedl, D., Yannakakis, G.N., What is procedural content generation?: Mario on the borderline. *Proc. of the 2nd Workshop on Procedural Content Generation in Games* (2011)

Game Content?

- Levels, tracks, maps, terrains, dungeons, puzzles, buildings, trees, grass, fire, plots, descriptions, scenarios, dialogue, quests, characters, rules, boards, parameters, camera viewpoint, dynamics, weapons, clothing, vehicles, personalities...
- Wow! Just about *anything*!
  - Except NPC behavior (this is AI)
  - More on this later!

History: Runtime Level Generation

- *Rogue* (1980)



History: Runtime Level Generation

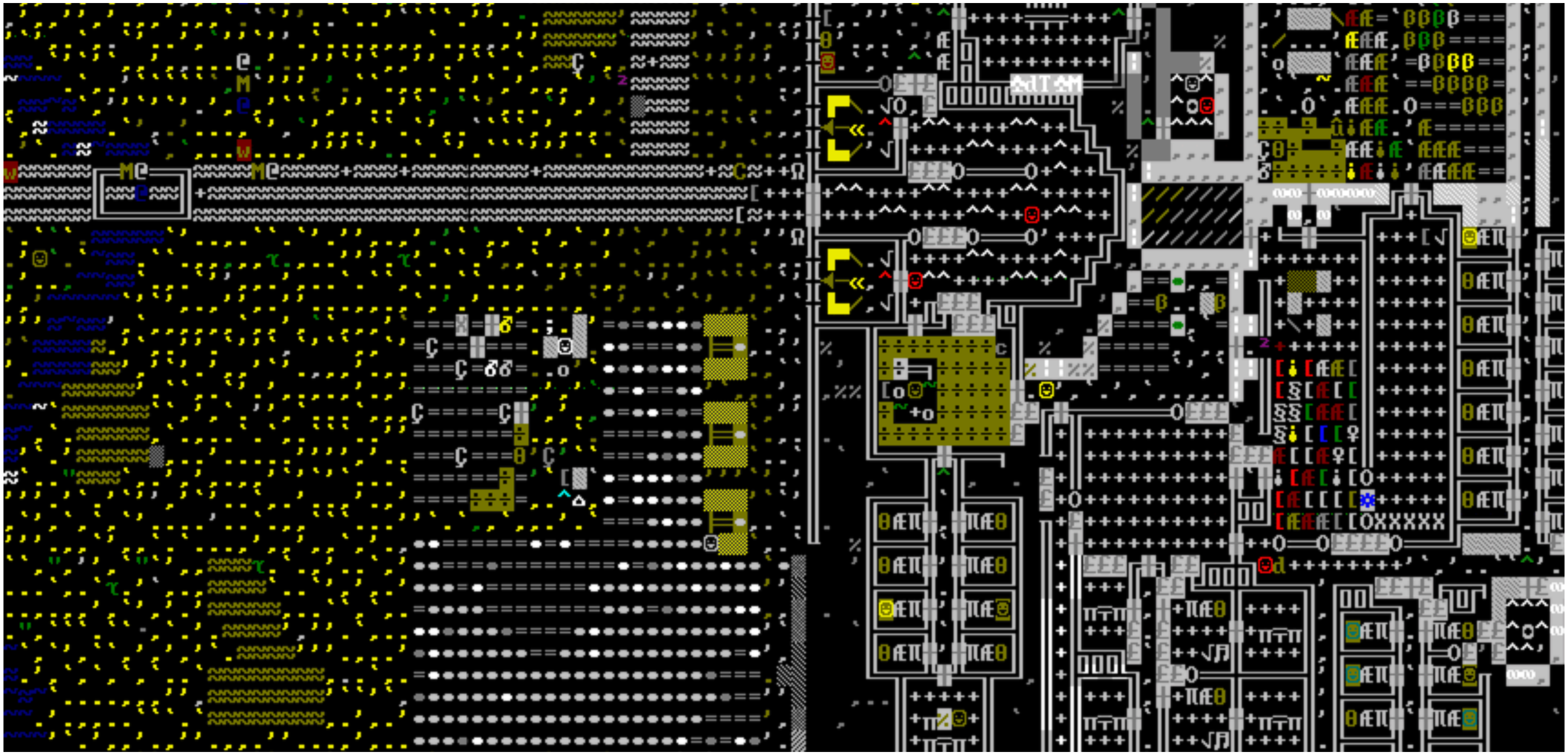
- *Tribal Trouble* (2005)

History: Runtime Level Generation

- *Civilization IV* (2005)

History: Runtime Level Generation

- *Dwarf Fortress* (2007)

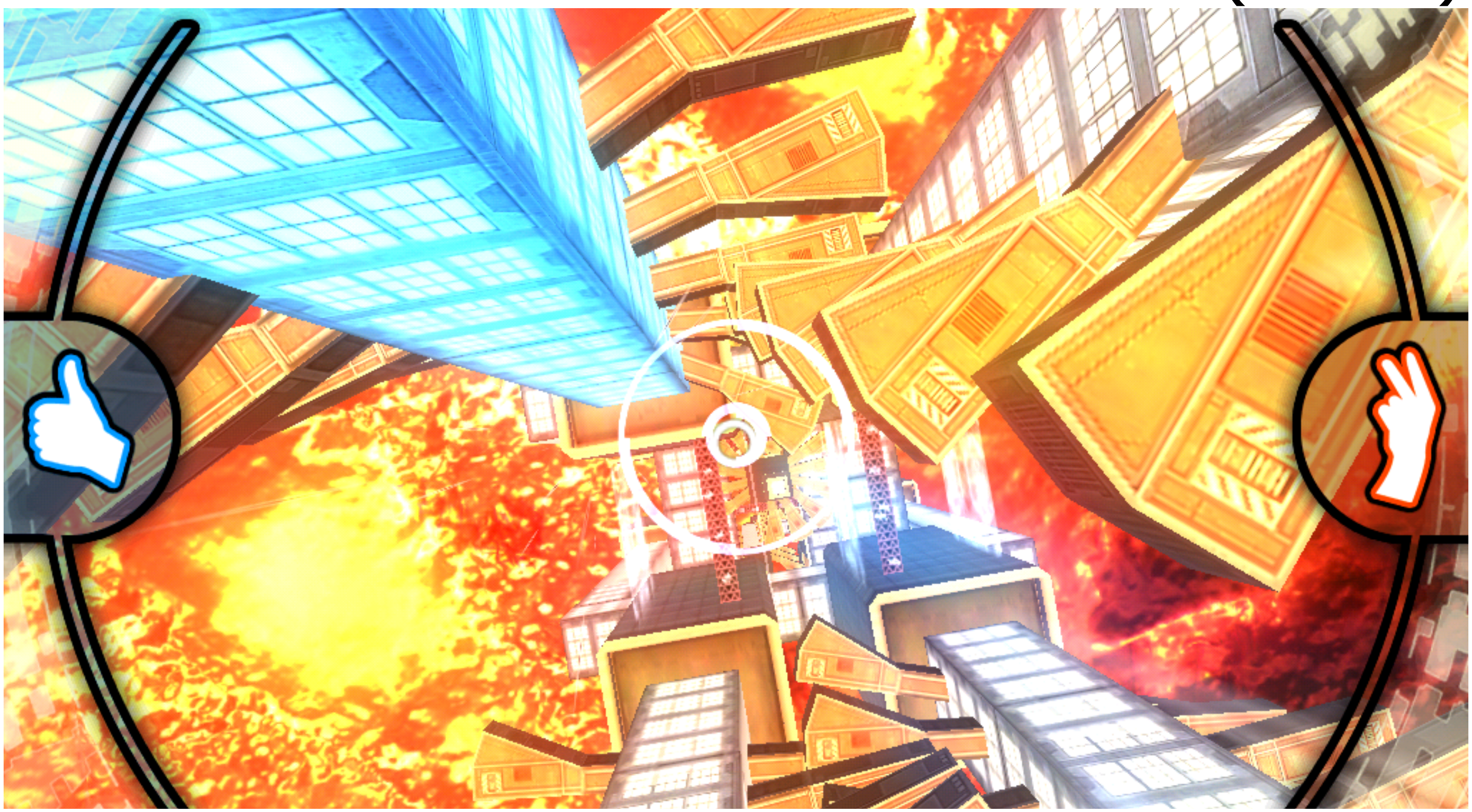


History: Runtime Level Generation

- *Diablo* (2008)

History: Runtime Level Generation

- *AaaaaAAaaaAAAaaAAAAaAAAAA* (2009)



History: Foliage Generation

- SpeedTree (*Oblivion*, 2009)

Terrain Generation: Can be Based on Physics

WPI

Terrain Generation using Procedural Models based on Hydrology

ACM Transactions on Graphics (Proceedings of SIGGRAPH), 2013

Jean-David Génevaux — *Université Lyon 2 - LIRIS*
Éric Galin — *Université Lyon 2 - LIRIS*
Éric Guérin — *INSA Lyon - LIRIS*
Adrien Peytavie — *Université Lyon 1 – LIRIS*
Bedřich Beneš — *Purdue University*

The Future?

- Can we drastically cut game development costs by creating content automatically from designers' intentions?
- Can we create games that adapt their game worlds to the preferences of the player?
- Can we create endless games?
- Can the computer circumvent or augment limited human creativity and create new types of games?

Procedural Dungeon Generation

- In general
  - PCG > Randomness
- Can think of approaches as
  - Online vs. Offline
  - Necessary vs. Optional
  - Random seed vs. Parameter vectors
  - Stochastic vs. Deterministic
  - Constructive vs. Generate-and-test

Online vs. Offline

- Online
  - As the game is being played
  - What could be the downside of this?
  - What is the upside?
- Offline
  - During development/building of the game
  - What could be the downside of this?
  - What is the upside?

Necessary vs. Optional

- Necessary content
  - Content the player needs to pass in order to progress
  - Move the story along, solve a puzzle, etc.
- Optional content
  - Can be discarded, or bypassed, or exchanged for something else
  - Background things, like terrain, forest, non-essential characters, etc.

Stochastic vs. Deterministic

- Deterministic
  - Given the same starting conditions, always creates the same content
- Stochastic
  - The above is not the case

Random Seeds vs. Parameter Vectors

- Also known as Dimensions of Control
- Can we specify the shape of the content in some meaningful way?

Constructive vs. Generate-and-test

- Constructive
  - Generate the content once, and be done with it
- Generate-and-test
  - Generate, test for quality, tweak, and re-generate until the content is good enough

Search-based Paradigm

- A special case of generate-and-test
  - The test function returns a numeric fitness value (not just accept/reject)
  - The fitness value guides the generation of new candidate content items
- Usually implemented through evolutionary computation
  - Genetic Algorithms

Evolutionary Computation?

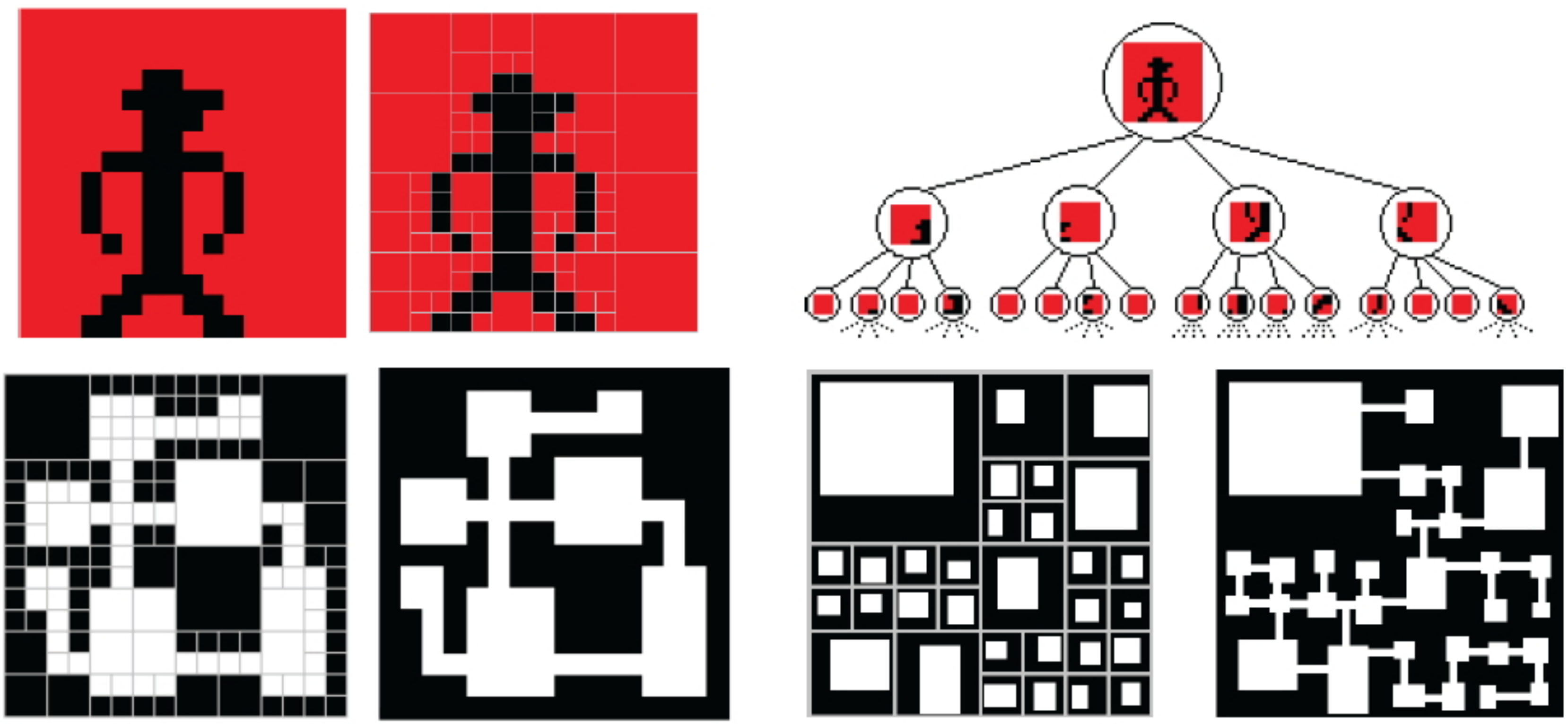
- Keep a population of candidates
- Measure the fitness of each candidate
- Remove the worst candidates
- Replace with copies of the best (least bad) candidates
- Mutate/crossover the copies
  - Can use all genetic operations (and some you can make up!)

Procedural Dungeon Generation

- In general
  - PCG > Randomness
- Space-Partitioning Algorithms
  - Macro approach
- Agent-Based Dungeon Growing
  - Micro approach

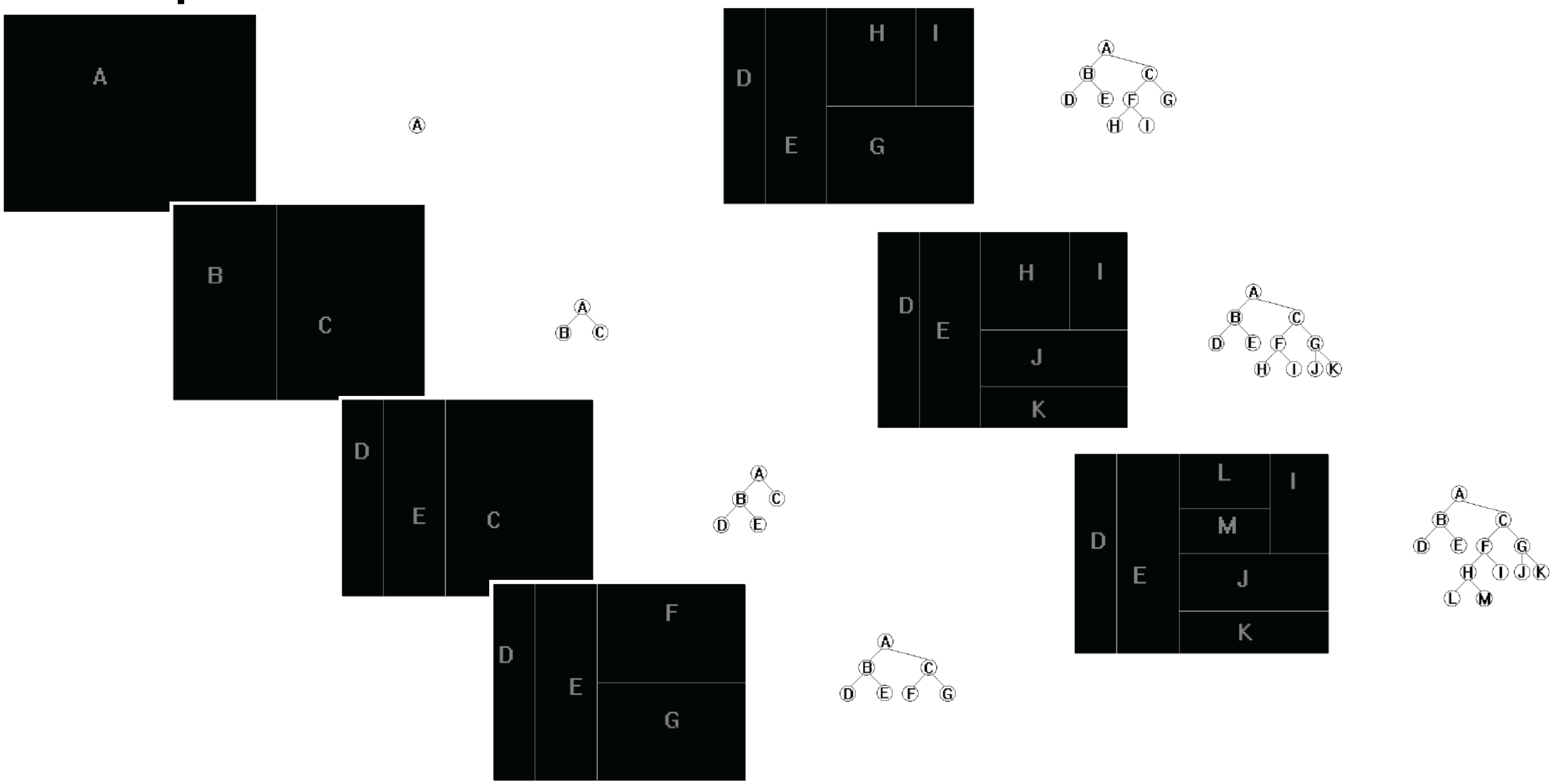
Space-Partitioning Approaches: Quad Trees

- Can partition the space, and choose how to fill each leaf



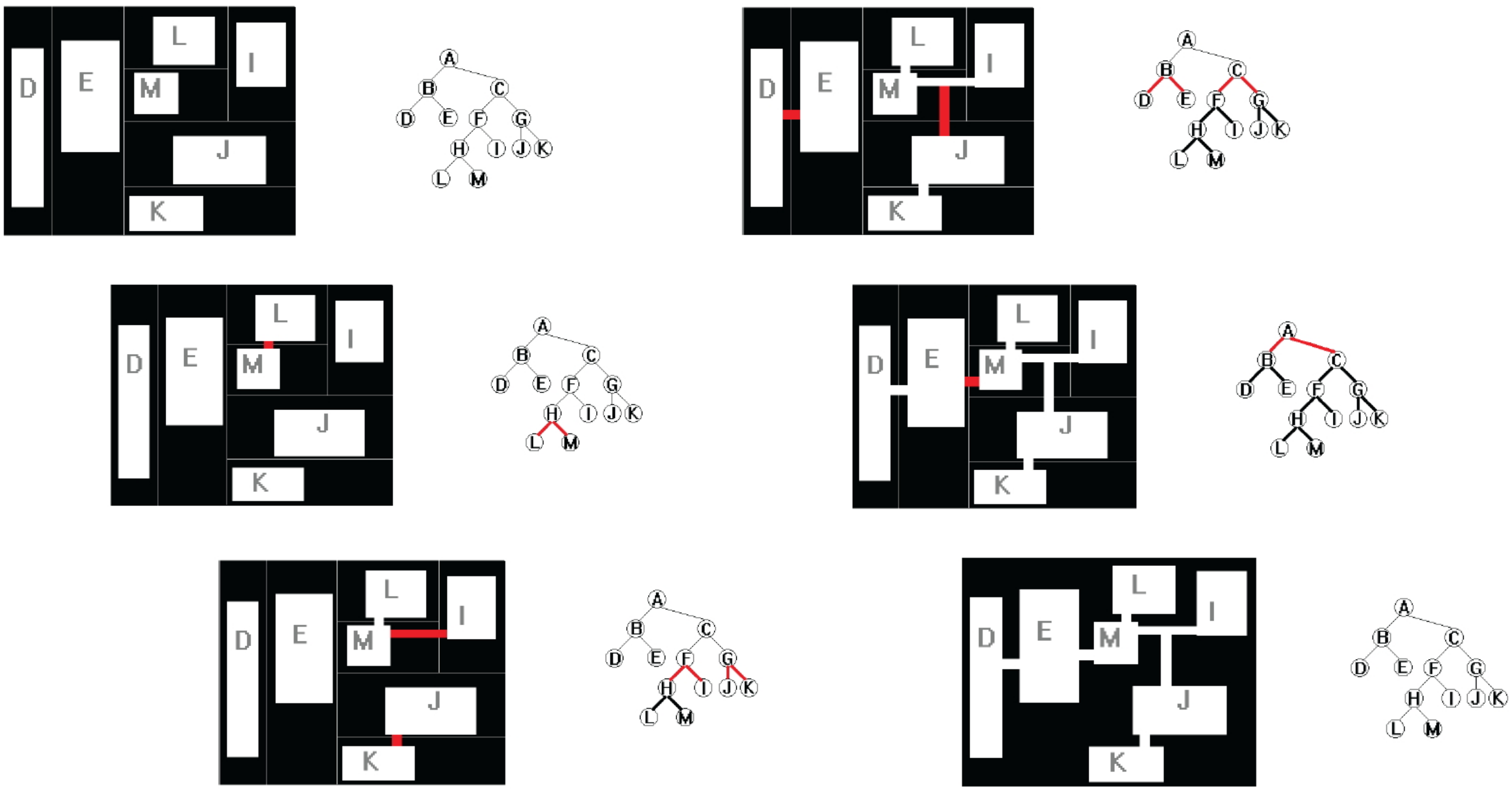
Space-Partitioning Approaches: K-D Trees

- Special case of BSP Trees



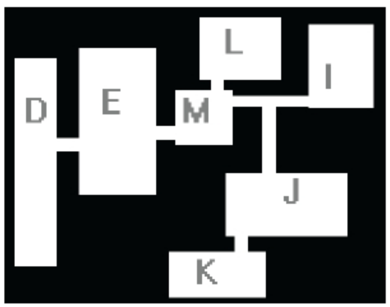
Space-Partitioning Approaches: K-D Trees

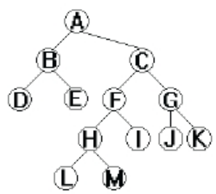
- Add rooms and corridors



Space-Partitioning Approaches: K-D Trees

- Add a theme to the resulting dungeon



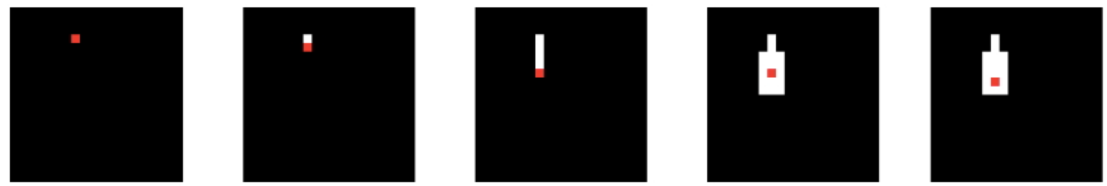


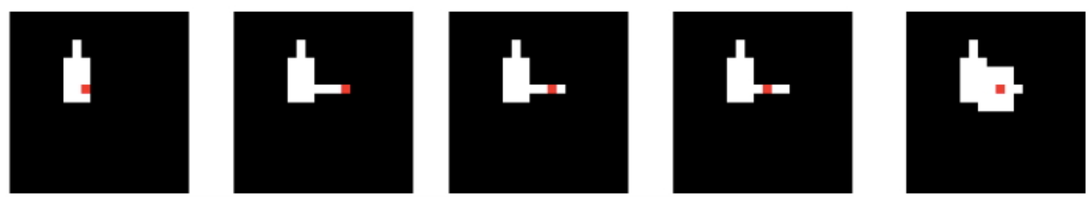
Agent-Based Dungeon Growing

- Agent chooses what to do based on different probabilities
  - Keep going, turn, build a room, etc.

Agent-Based Dungeon Growing: "Blind" Digger Code

```
1.  initialize chance of changing direction Pc=5
2.  initialize chance of adding room Pr=5
3.  place the digger at a dungeon tile and randomize its direction
4.  dig along that direction
5.  roll a random number Nc between 0 and 100
6.  if Nc below Pc:
7.      randomize the agent's direction
8.      set Pc=0
9.  else:
10.     set Pc=Pc+5
11. roll a random number Nr between 0 and 100
12. if Nr below Pr:
13.     randomize room width and room height between 3 and 7
14.     place room around current agent position
15.     set Pr=0
16. else:
17.     set Pr=Pr+5
18. if the dungeon is not large enough:
19.     go to step 4
```





Agent-Based Dungeon Growing: "Look Ahead" Digger Code

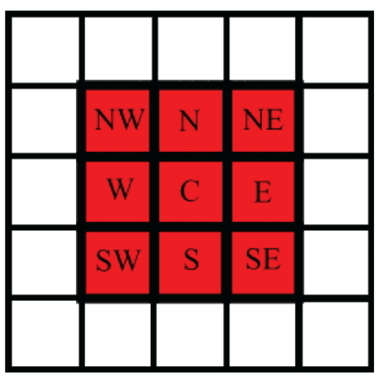
1. place the digger at a dungeon tile
2. set helper variables Fr=0 and Fc=0
3. for all possible room sizes:
4.   if a potential room will not intersect existing rooms:
5.   place the room
6.   Fr=1
7.   break from for loop

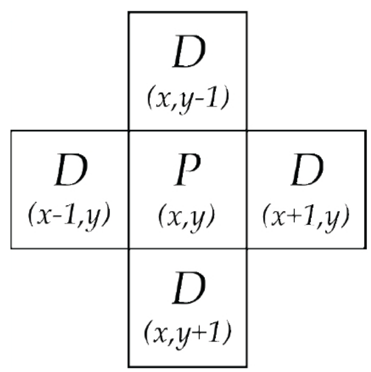
8. for all possible corridors of any direction and length 3 to 7:
9.   if a potential corridor will not intersect existing rooms:
10.   place the corridor
11.   Fc=1
12.   break from for loop
13. if Fr=1 or Fc=1:
14.   go to 2

Cellular Automata

- A discrete computational model
  - An *n*-dimensional grid
    - E.g., two-dimensional grid
  - A set of states
    - Simplest: ON/OFF
  - A set of transition rules
    - Decide what to do based on neighborhood

Moore Neighborhood





von Neumann Neighborhood

Cellular Automata

- Number of possible configurations of a neighborhood?
  - $Possible\_States^{Number\_of\_Cells}$
  - E.g., for a two-state automata and a Moore neighborhood of size 2,

    $2^{25} = 33{,}554{,}432$
  - Small neighborhoods usually use a lookup
    - Each neighborhood configuration leads to a state
  - Large neighborhoods usually use a proportion of cells of each state

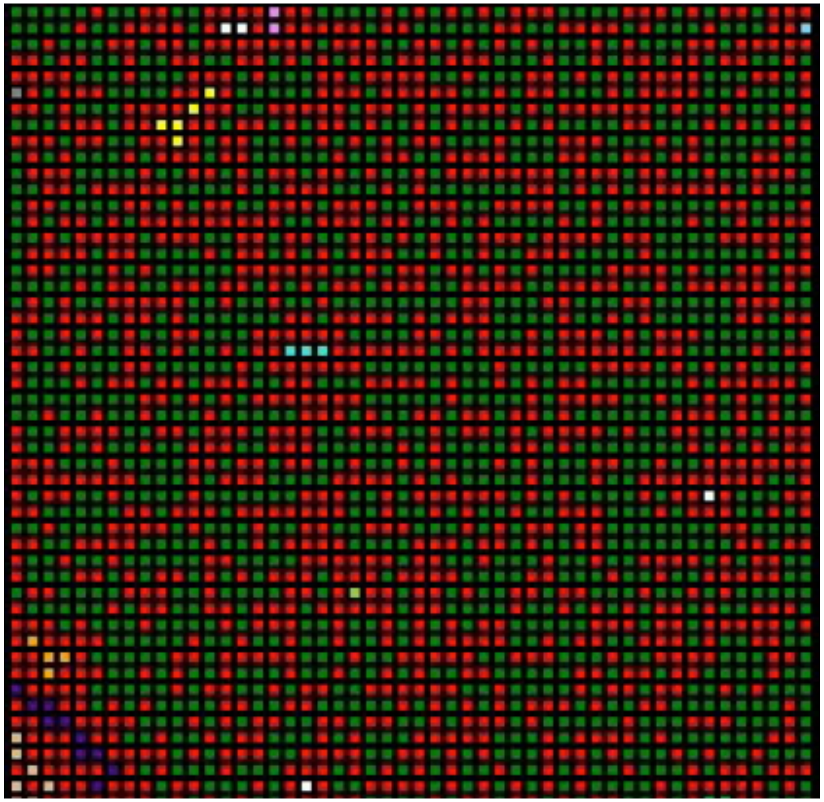
Example: Infinite Caves*

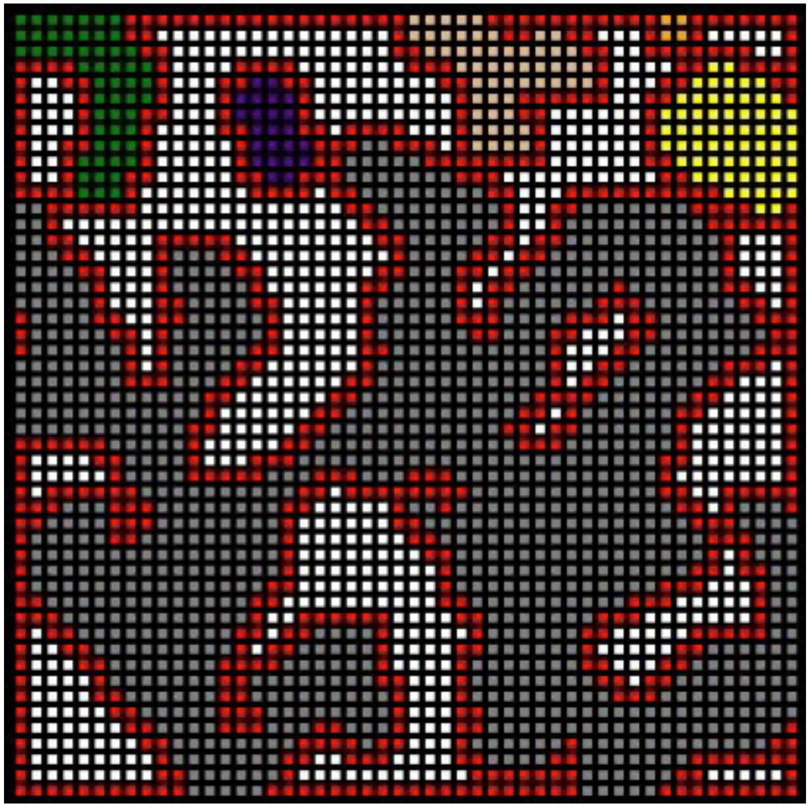
- Each room is a 50x50 grid, where each cell can be either *empty* or *rock* (2 states)
- Initially, each cell has a probability r (e.g., 0.5) that it is rock
  - Leads to relatively uniform rock distribution
- Apply a single rule to the grid for n (e.g., 2) steps
  - A cell turns into rock in the next step if at least T (e.g., 5) neighbors are rock, otherwise, it turns into free space
- For looks, rock cells that border empty space are designated as "walls", but function like rock

*Johnson, L., Yannakakis, G.N., Togelius, J.: Cellular Automata for Real-time Generation of Infinite Cave Levels. In: Proceedings of the ACM Foundations of Digital Games. ACM Press (2010)

Example: Infinite Caves*

- Random vs. Cooked

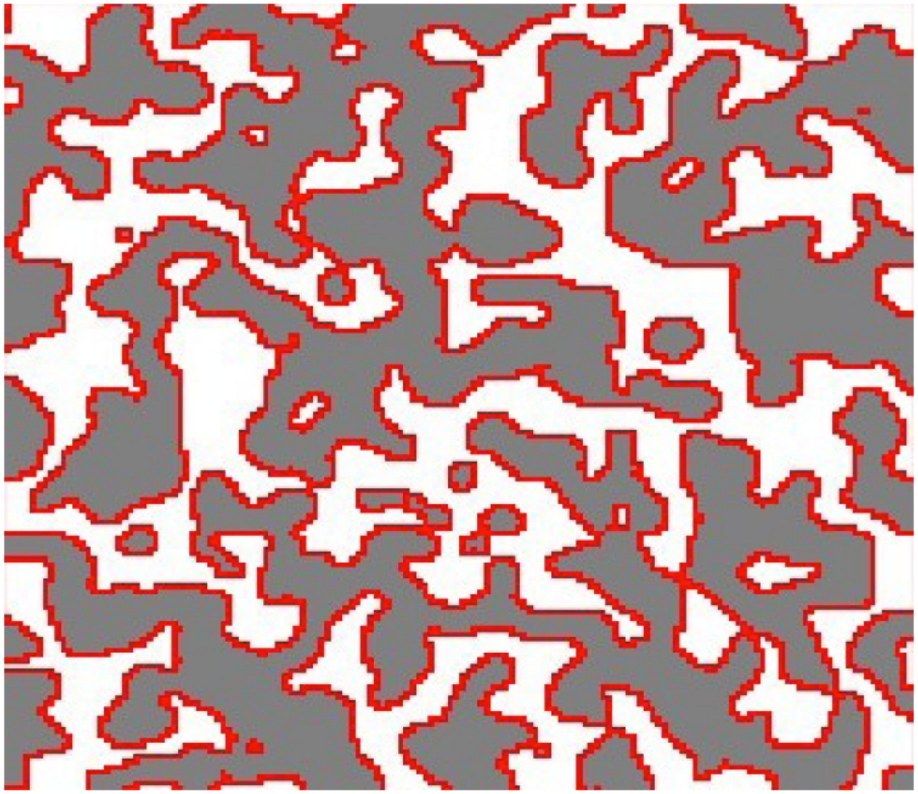




Red=Wall White=Rock, Other=Floor clusters CA params: n =4, M=1, T=5

Example: Infinite Caves*

- Need to connect rooms, and smooth
  - Drill at thinnest points, then run two more iterations



How would you build this?

Controlled Procedural Terrain Generation Using Software Agents

Adapted by Julian Togelius from

Jonathon Doran and Ian Parberry

Published in IEEE TCIAIG, 2010

Five Agent Types

- Apply each of these agents in succession
  - Coastline agents
  - Smoothing agents
  - Beach agents
  - Mountain agents
  - River agents
- Agent Rules
  - Each agent has a number of “tokens” to spend on actions
  - Each agent is allowed to see the current elevation around it, and allowed to modify it
  - Agents don’t interact directly

In the beginning...

- ...there was a vast ocean.
- Then came the first coastline agent.

Coastline Agents

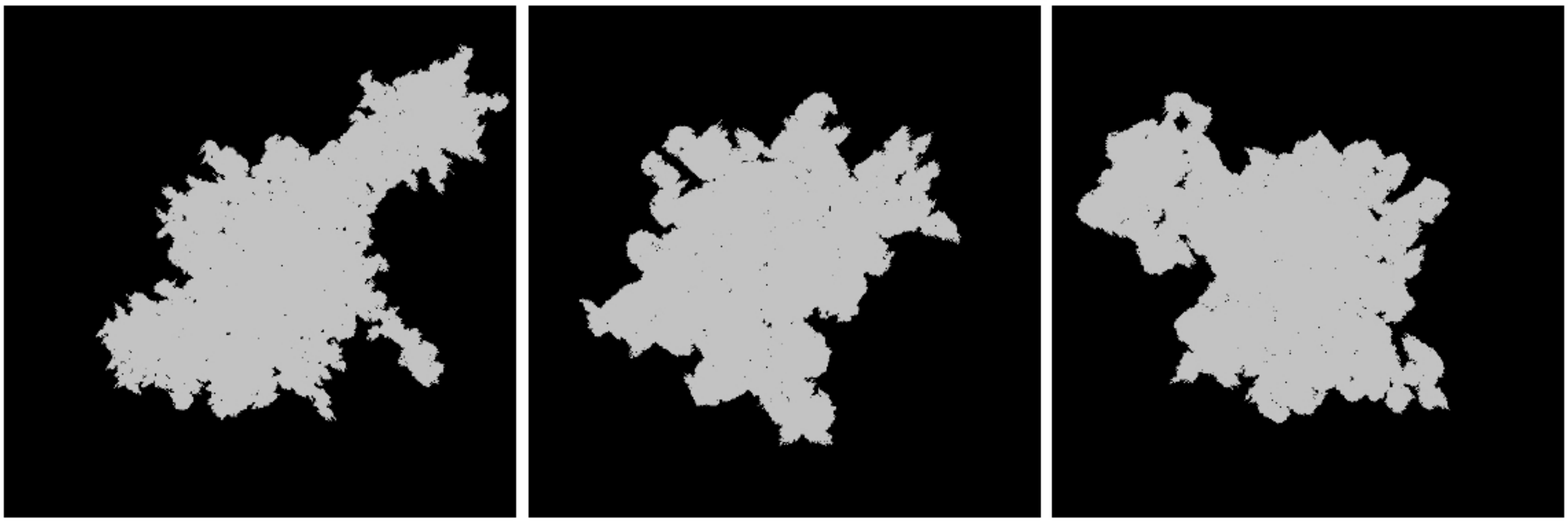
- Multiply until they cover the whole coast
  - About 1000 necessary for this size map
- Move out to position themselves right at the border of land and sea
- Generate a repulsor and an attractor point
- Score all neighboring points according to distance to repulsor and attractor points
- Move to the best-scoring points, adding land as they go along

Coastline Agent Code

```
COASTLINE-GENERATE(agent)
if tokens(agent) ≥ limit
    then
        create 2 child agents
        for each child
            do
                child ← a random seed point on parent's border
                child ← 1/2 of the parent's tokens
                child ← a random direction
                COASTLINE-GENERATE(child)
    else
        for each token
            do
                point ← random border point
                for each point p adjacent to point
                    do
                        score p
                fill in the point with the highest score
```

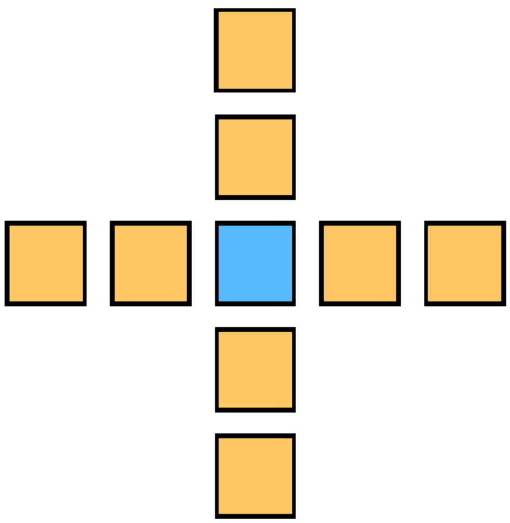
Coastline Agents

- Varying action sizes (number of tokens)



Smoothing Agents

- Take random walks on the map
- Change the elevation of each visited point to (almost) the mean of its extended von Neumann neighborhood



Smoothing Agent Code

```
SMOOTH(starting-point)
location ← starting-point
for each token
        do
            height_location ← weighted average of neighborhood
            location ← random neighboring point
```

Beach Agents

- Select random position along the coast, where coast is not too steep
- Flatten an area around this point (leaving small variations)
- Move randomly a short direction away from the coast, flattening the area

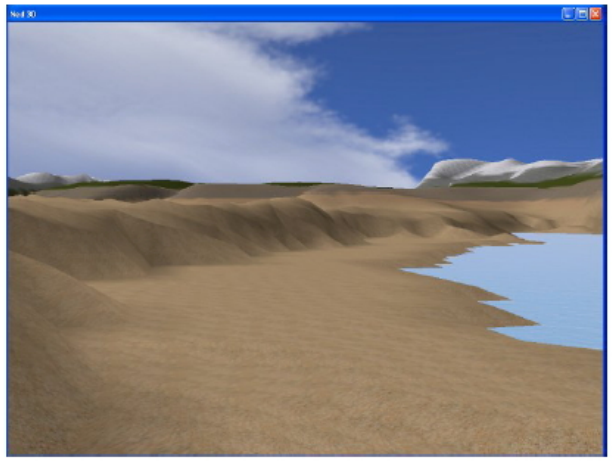
Beach Agent Code

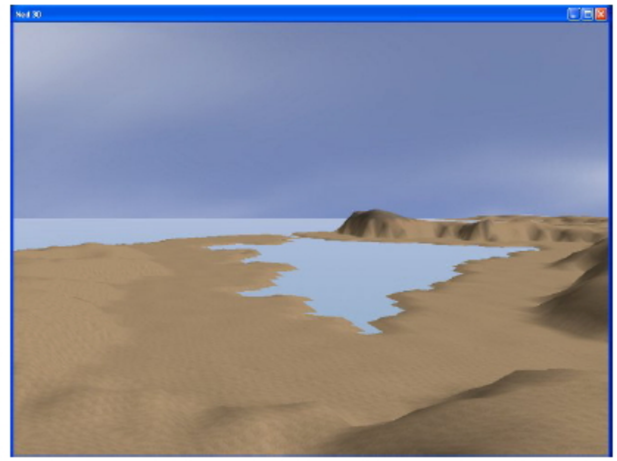
```
BEACH-GENERATE(starting-point)

location ← starting-point
for each token
    do
        if height_location ≥ limit
            then
                location ← random shoreline point
        flatten area around location
        smooth area around location
        inland ← random point a short distance inland from location
        for i ← 0 to size(walk)
            do
                flatten area around inland
                smooth area around inland
                inland ← random neighboring point
        location ← random neighboring point of location
```

Beach Agents

- Varying beach width





Mountain Agents

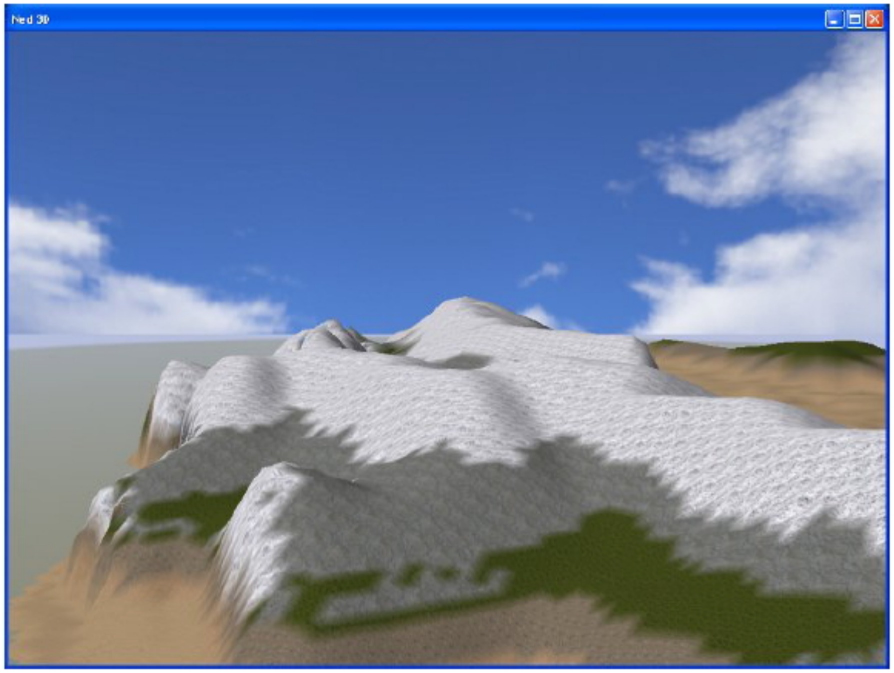
- Start at random positions and directions
- Move forward, continuously elevating a wedge, creating a ridge
- Turn randomly without 45 degrees from the initial course
- Periodically offshoot "foothills" perpendicular to movement direction

Mountain Agent Code

```
MOUNTAIN-GENERATE(starting_point)
location ← starting-point
direction ← random direction
for each token
        do
            elevate wedge perpendicular to direction
            smooth area around location
            location ← next point in direction
            every n-th token
                do
                    direction ← original-direction ± 45-degrees
```

Mountain Agents

- Narrow vs. wide features



River Agents

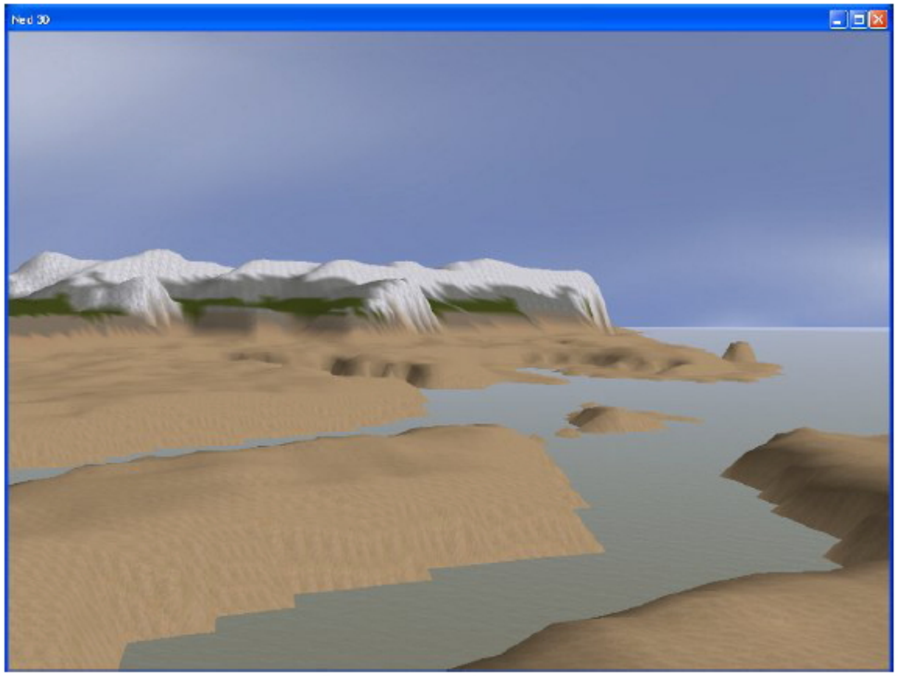
- Move from a random point on the coast towards a random point on a mountain ridge
- "Wiggle" along the path
- Stop when reaching too high altitudes
- Retrace the path down to the ocean, deepening a wedge along the path

River Agent Code

```
RIVER-GENERATE()
coast ← random point on coastline
mountain ← random point at base of a mountain
point ← coast
while point not at mountain
        do
              add point to path
              point ← next point closer to mountain
while point not at coast
        do
              flatten wedge perpendicular to downhill direction
              smooth area around point
              point ← next point in path
```

River Agents

- A dry river, and the outflow of three rivers



In What Order?

- Doran and Parberry suggest
  - Coastline
  - Landform
  - Erosion
- But the "Implementation" suggests random order

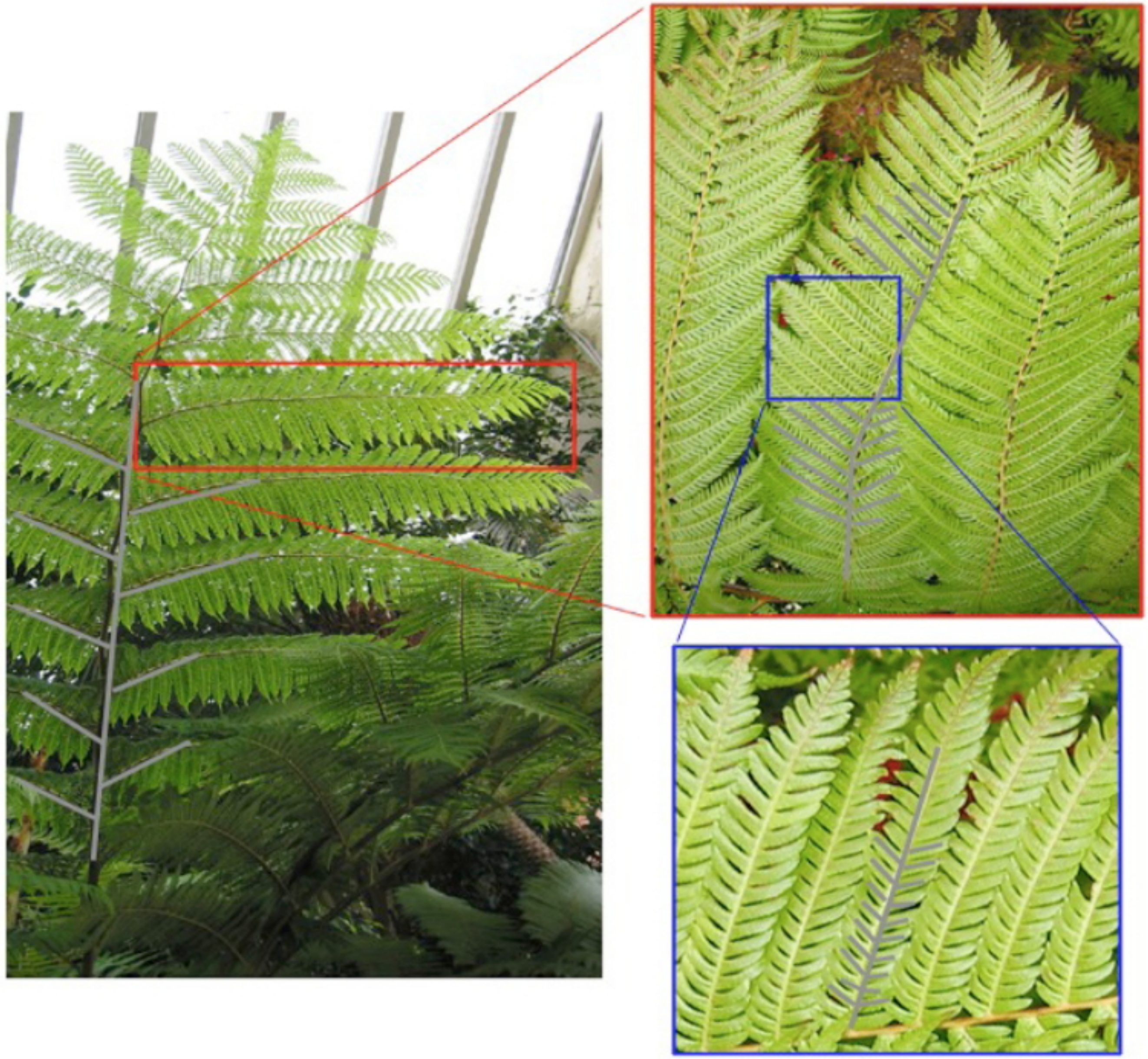
Further Questions

- Parameters... what parameters?
- What features of landscapes do we want to be able to specify?
- How can the human and the algorithm interact productively?

Self Similarity

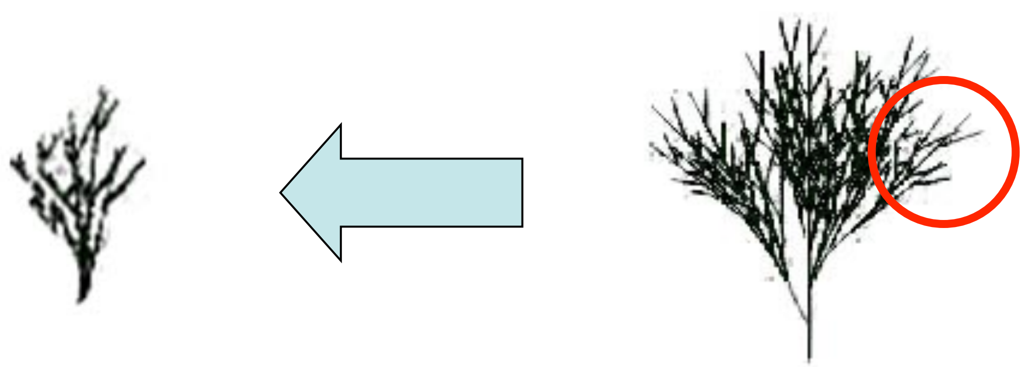
- Level of detail remains the same as we zoom in
- Example
  - Surface roughness, or silhouette, of mountains is the same at many zoom levels
  - Difficult to determine scale
- Types of fractals
  - Exactly self-similar
  - Statistically self-similar

Example: Ferns

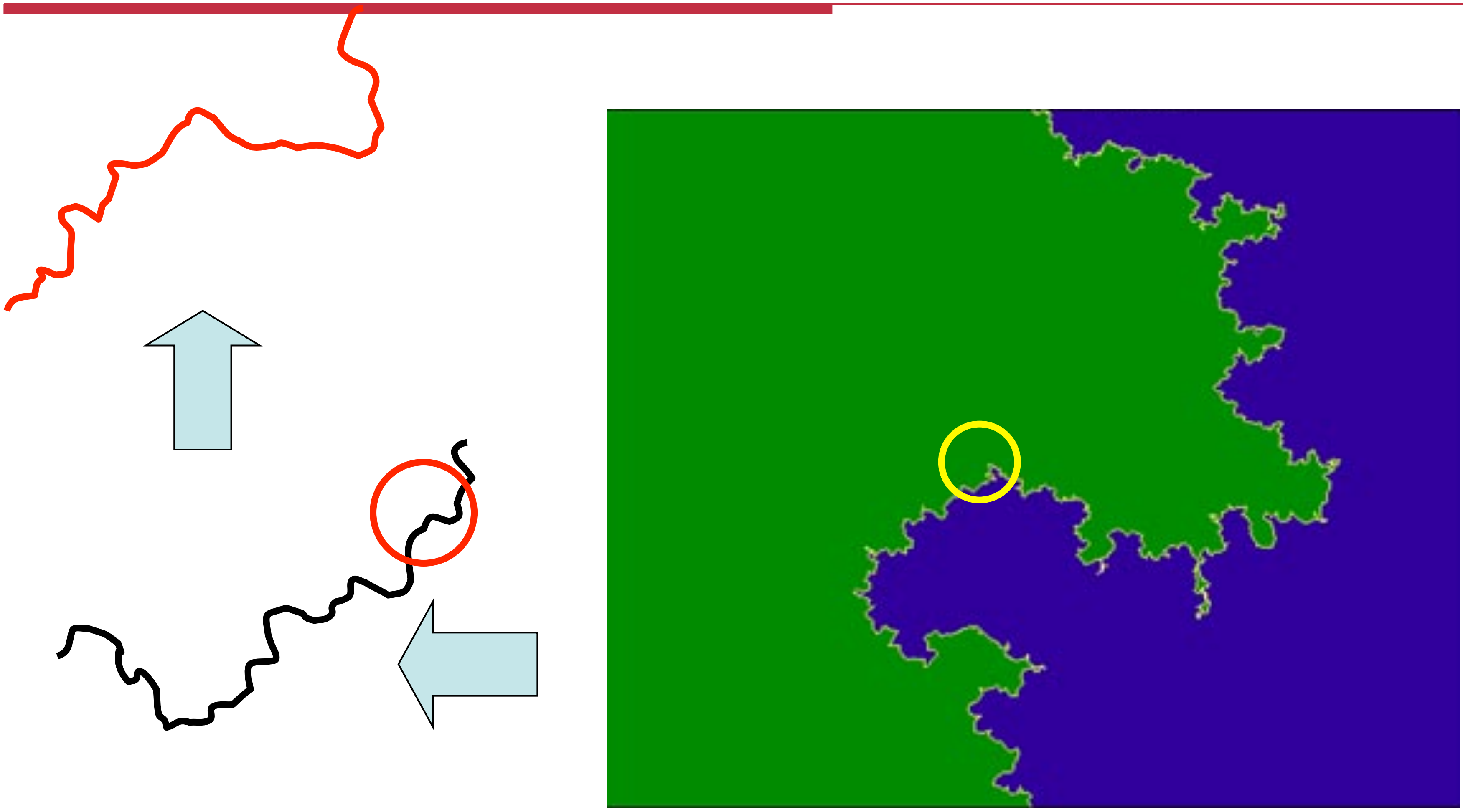


Fractals and Self-Similarity

- **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.



Fractal Coastline

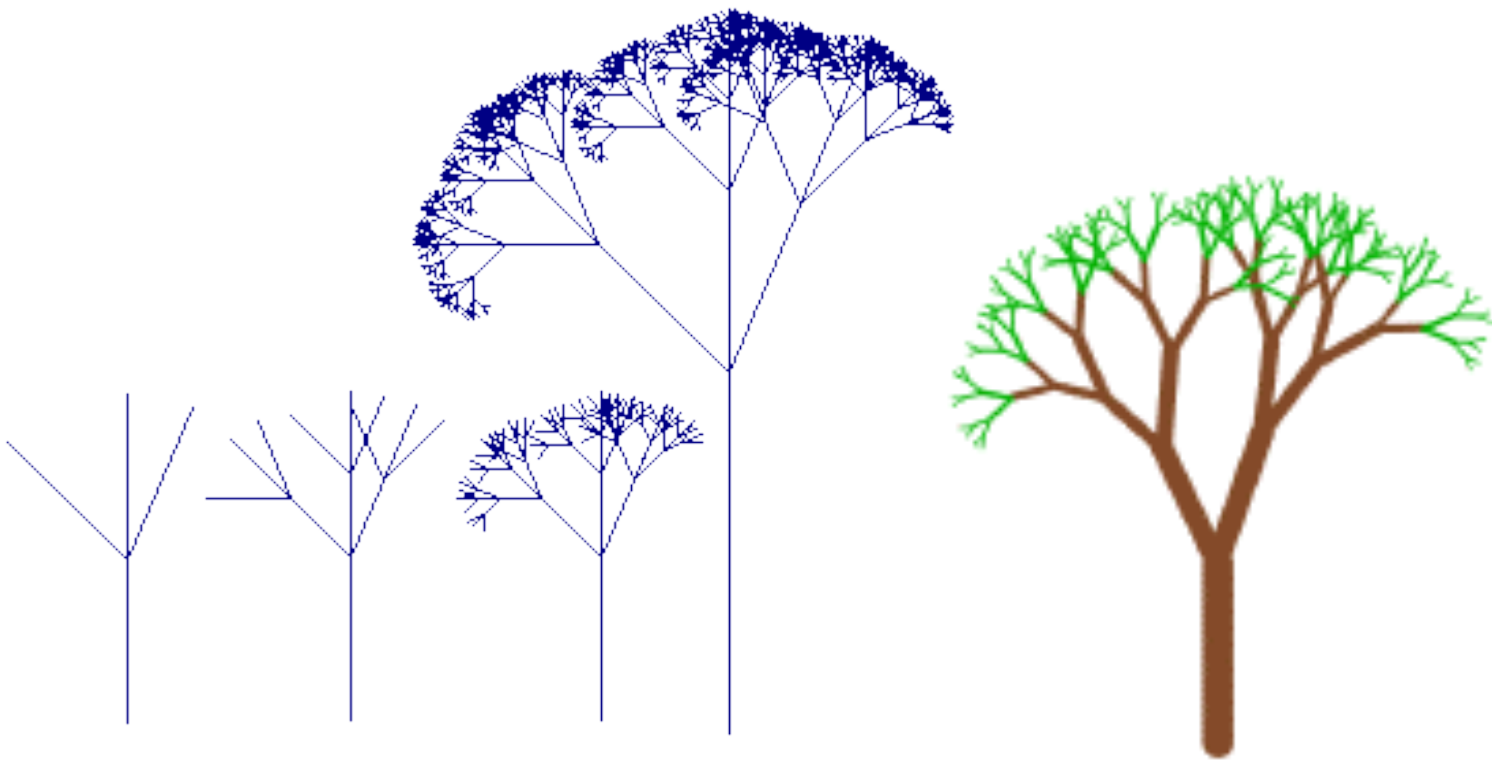


Examples of Fractals

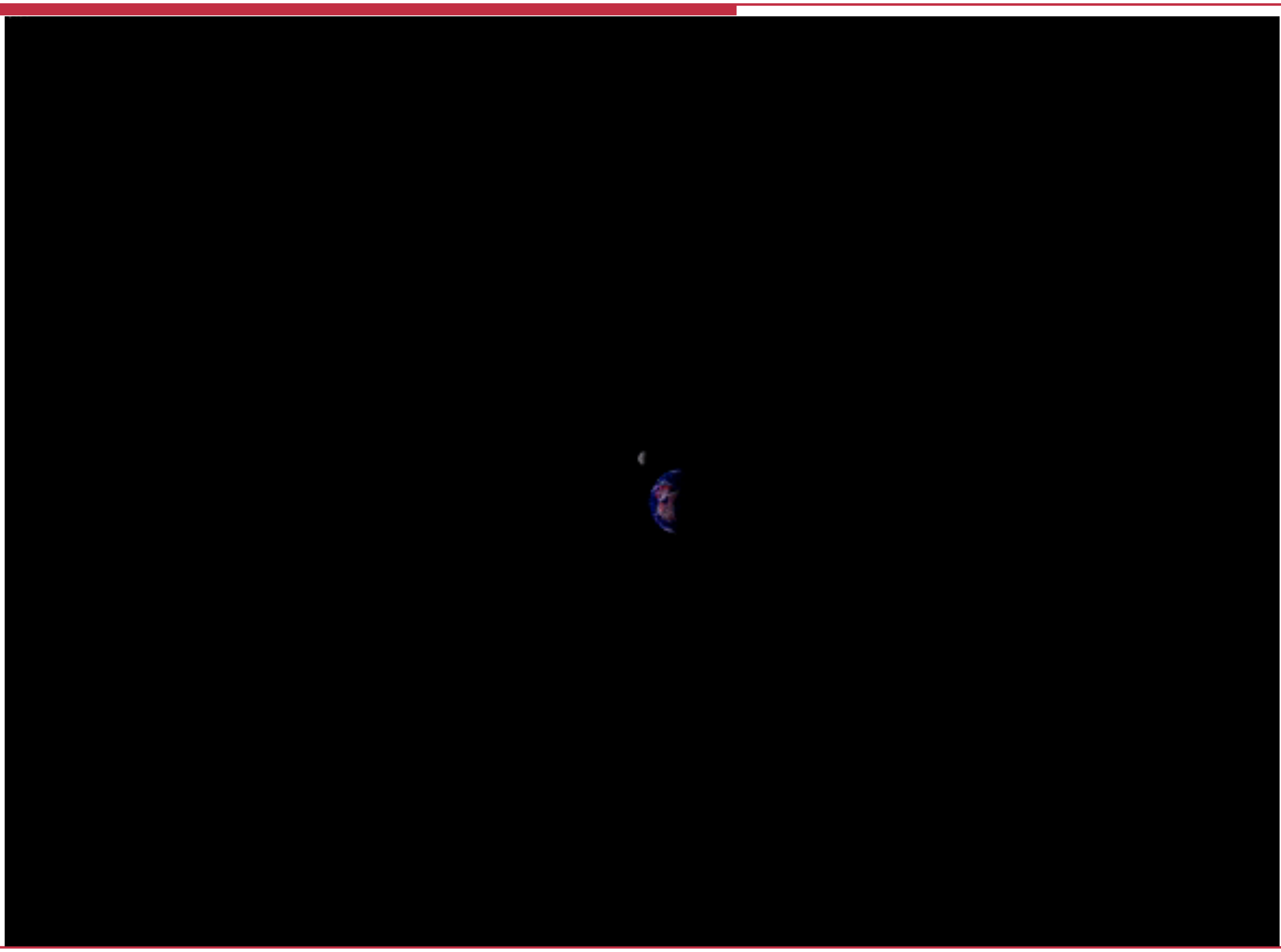
- Modeling mountains (terrain)
- Clouds
- Fire
- Branches of a tree
- Grass
- Coastlines
- Surface of a sponge
- Cracks in the pavement
- Designing antennae (www.fractenna.com)

Examples of Fractals: Trees

Fractals appear "the same" at every scale.

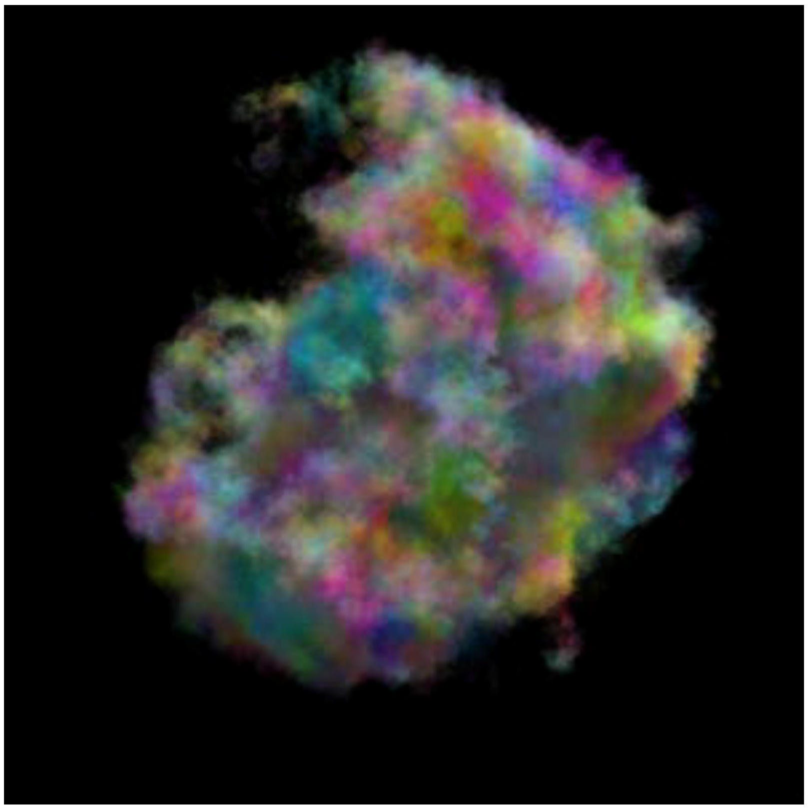


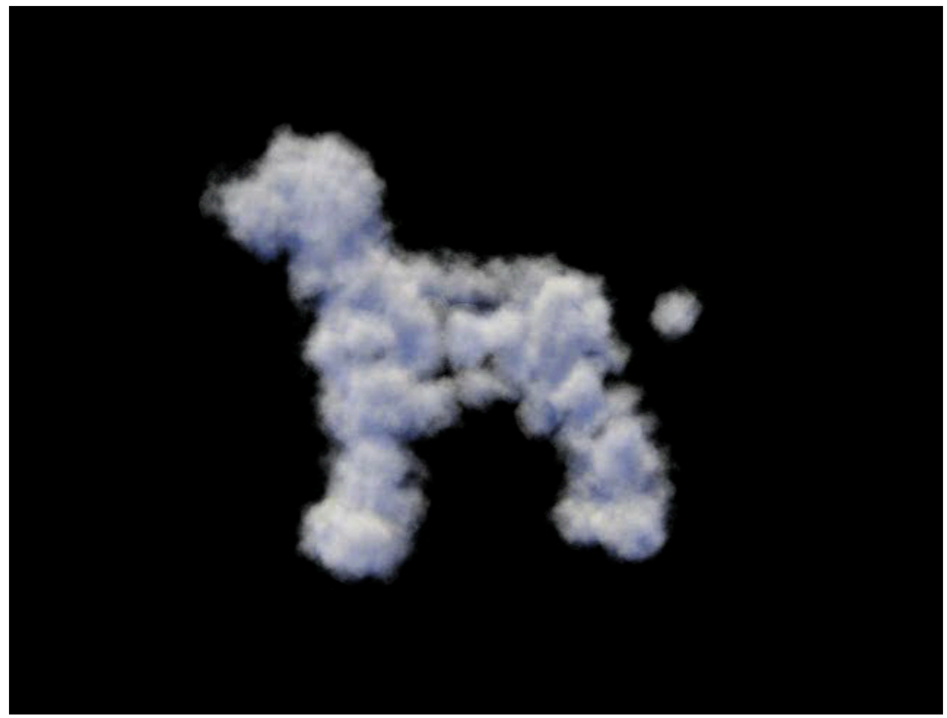
Examples of Fractals: Mountains



Images: www.kenmusgrave.com

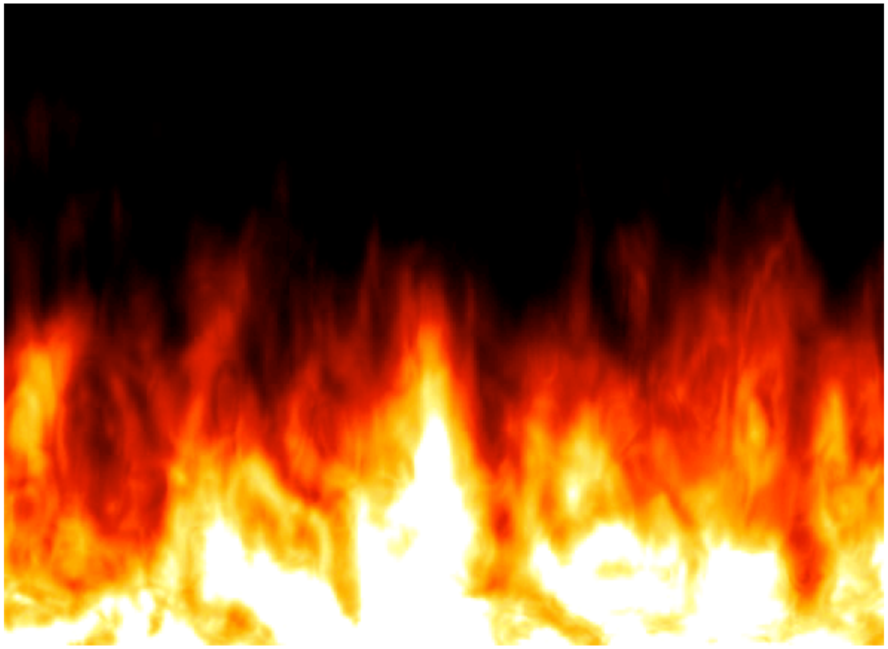
Examples of Fractals: Clouds





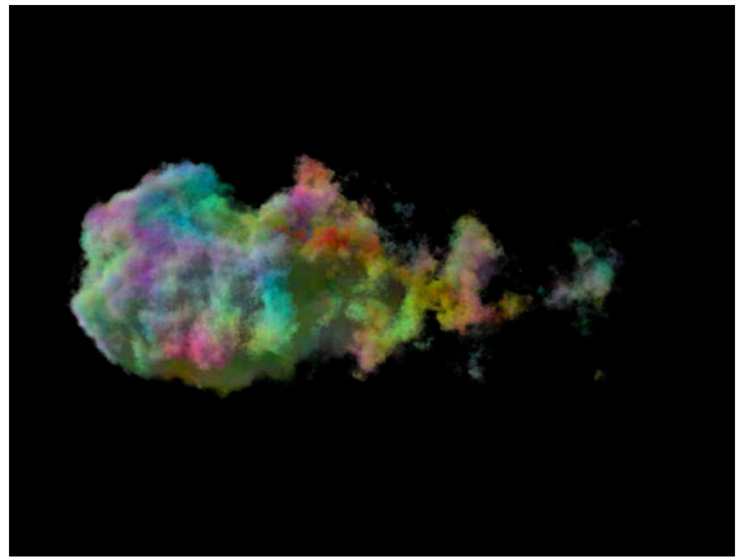
Images: www.kenmusgrave.com

Examples of Fractals: Fire



Images: www.kenmusgrave.com

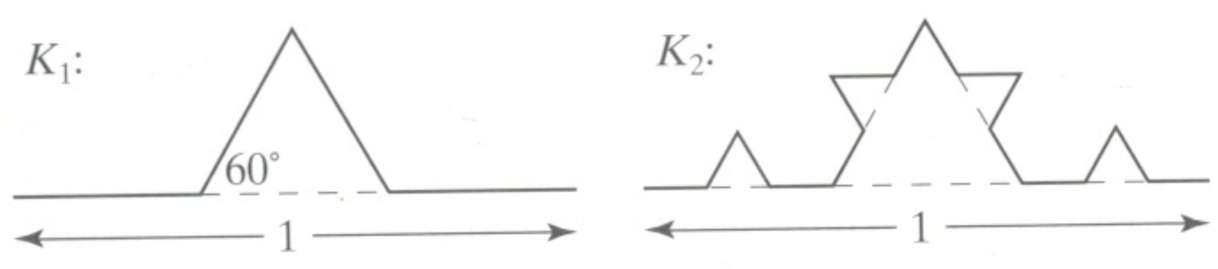
Examples of Fractals: Comets?



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

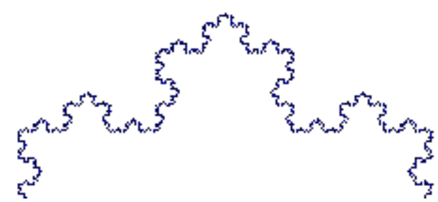


Koch Snowflake

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

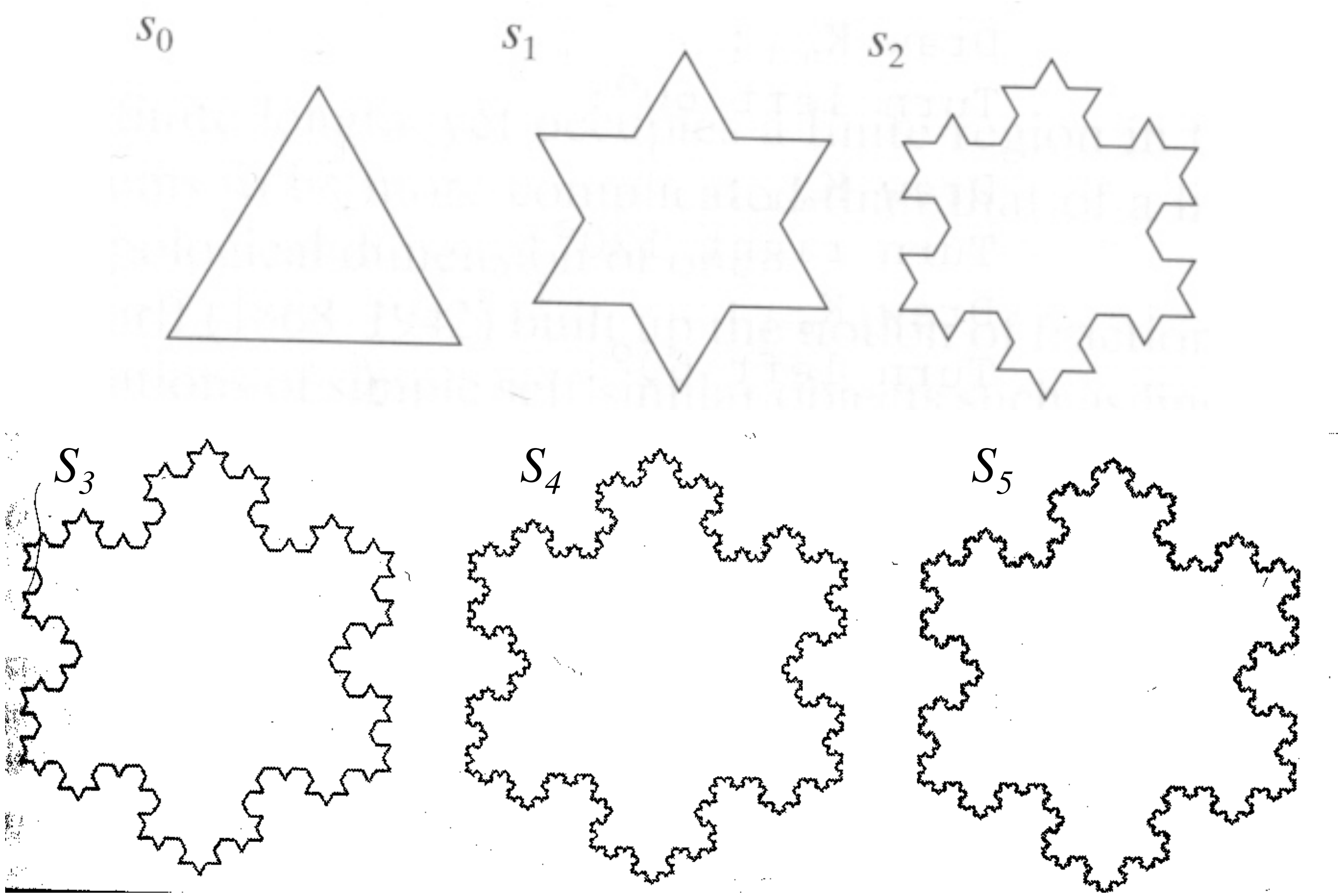
$$P_i = 3\left(\frac{4}{3}\right)^i$$

  where P_i is the perimeter of the *i*th 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

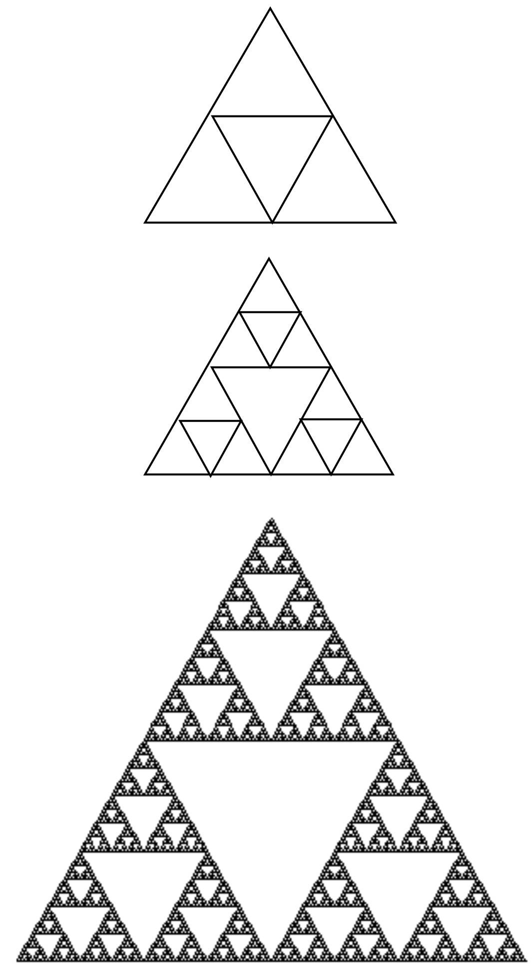


Fractal Dimension – Eg. 2

The Sierpinski Triangle

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

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

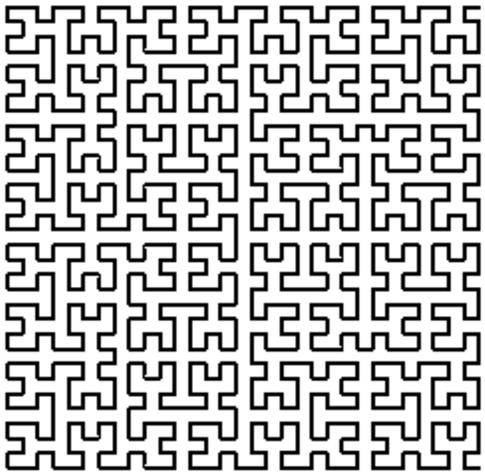


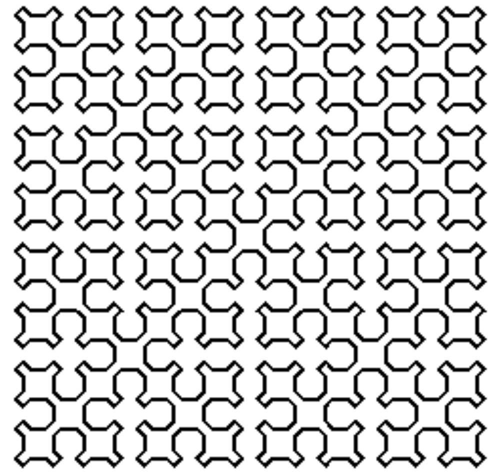
Space-Filling Curves

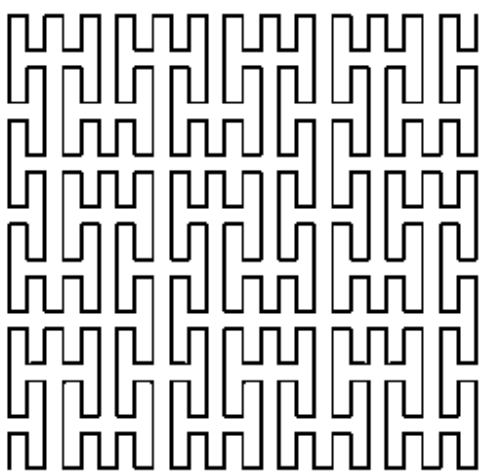
- There are fractal curves which completely fill up higher dimensional spaces such as squares or cubes.
- The space-filling curves are also known as Peano curves (Giuseppe Peano: 1858-1932).
- Space-filling curves in 2D have a fractal dimension 2.

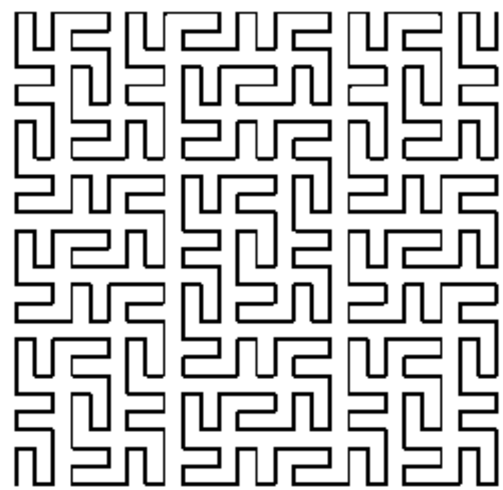
You're not expected to be able to prove this.

Space-Filling Curves

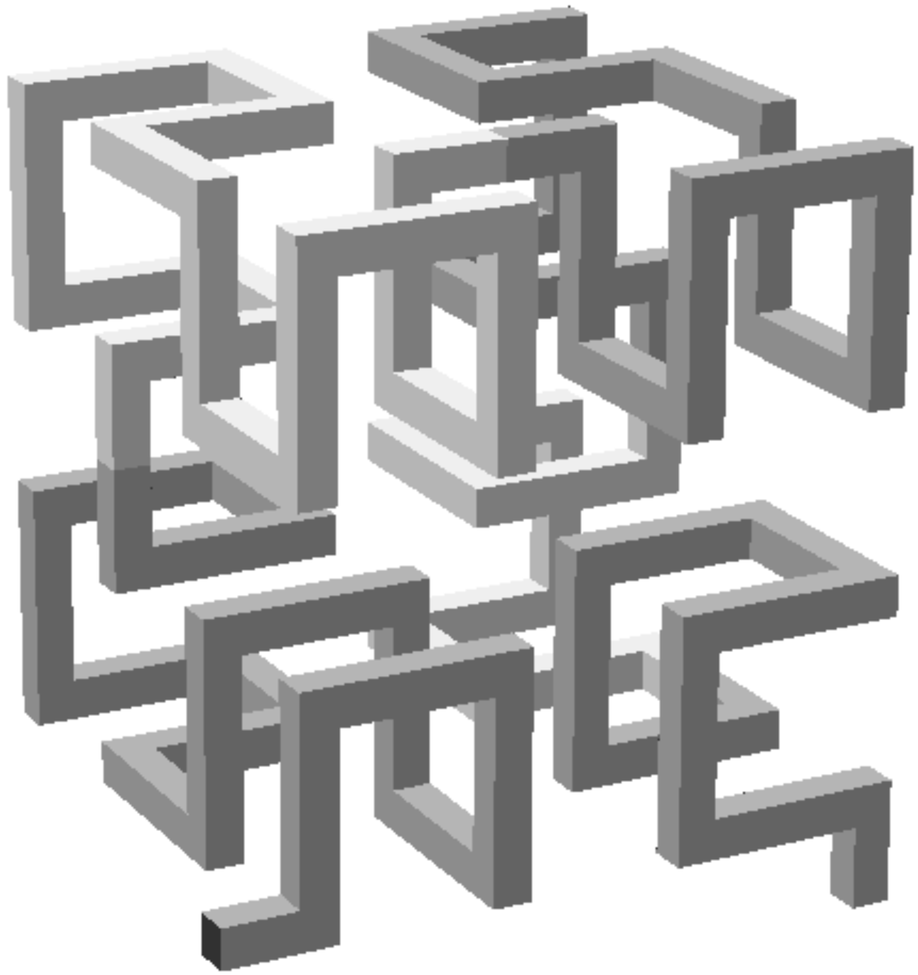








Space-Filling Curves in 3D



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")

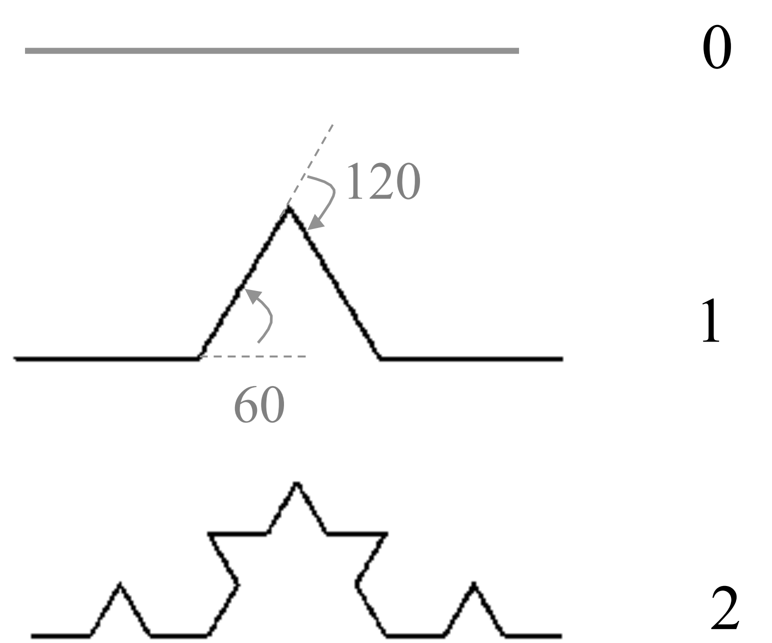
- A grammar-based model for generating simple fractal curves
  - Devised by biologist Aristid Lindenmayer for modeling cell growth
  - Particularly suited for rendering line drawings of fractal curves using turtle graphics
- Consists of a start string (*axiom*) and a set of *replacement rules*
  - At 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 → 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

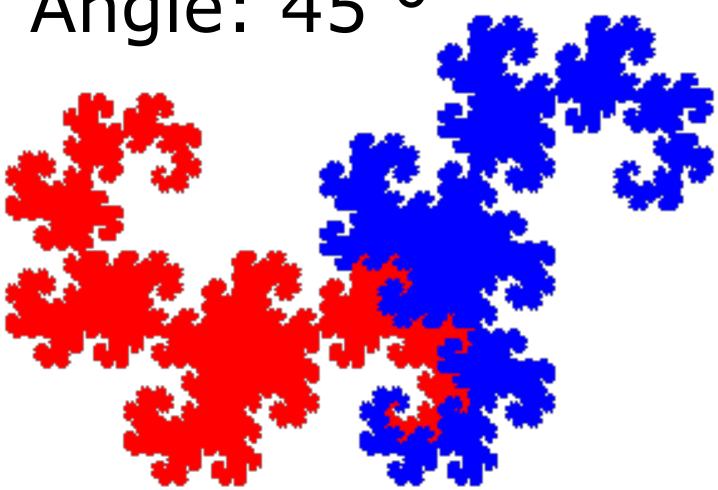
The Dragon Curve

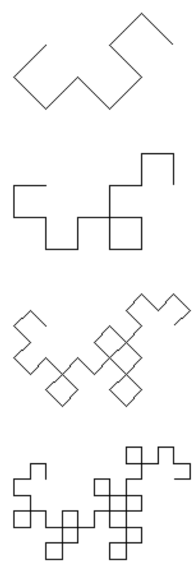
Axiom: FX

Rules:

- F → ∅
- X → +FX−−FY+
- Y → −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.

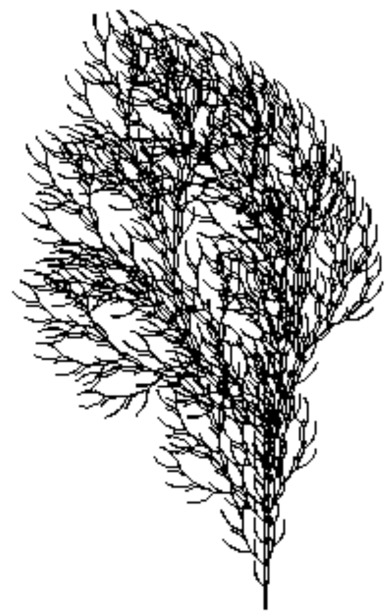
Generalized Grammars

Fractal bush:

S → F

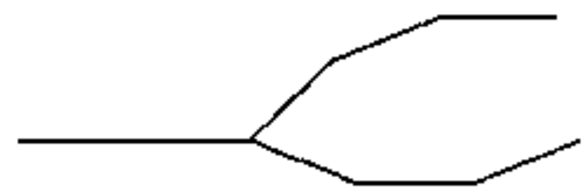
F → FF-[-F+F+F]+[+F-F-F]

(A = 22 degs.)



Fourth order bush
(with 90 deg. rotation)

Zero order bush
F



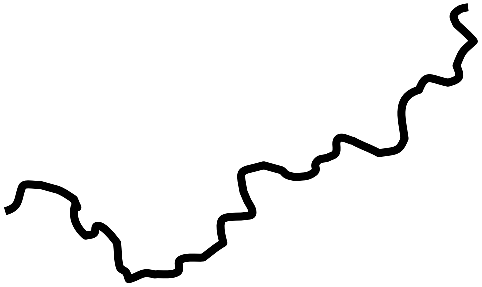
First order bush

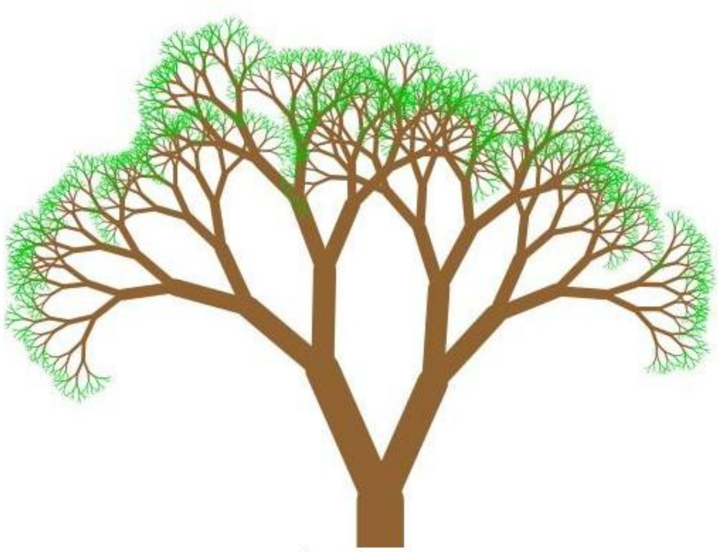
Random Fractals

- 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.



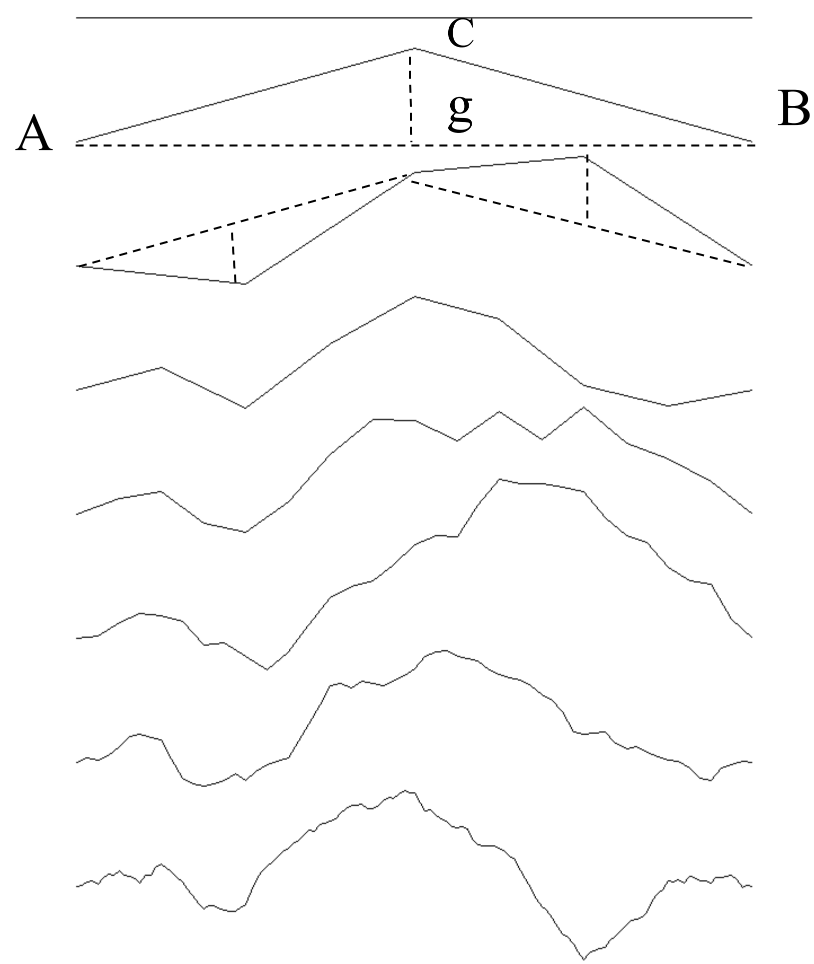


Random Midpoint Displacement Algorithm (2D)

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

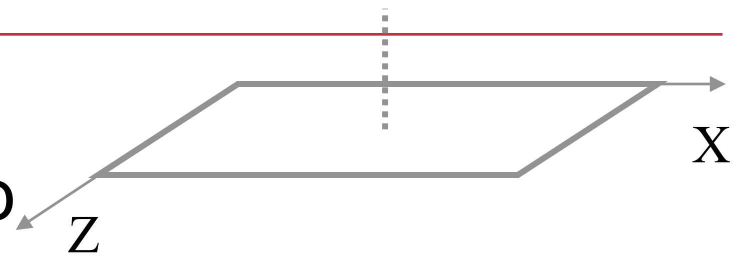
  $$y_C = (y_A + y_B)/2 + g$$

  - 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)

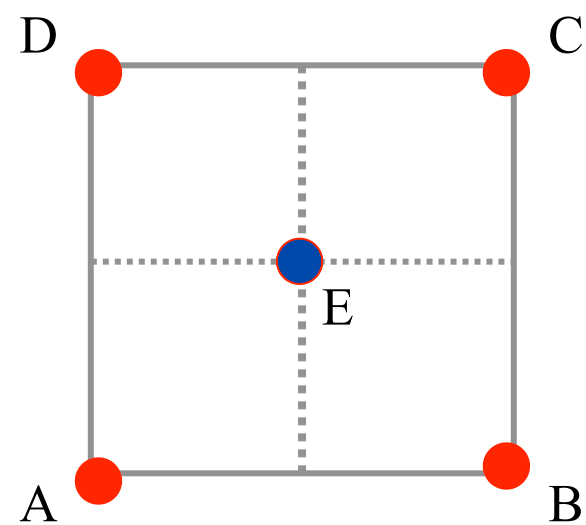
Square-Step:



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

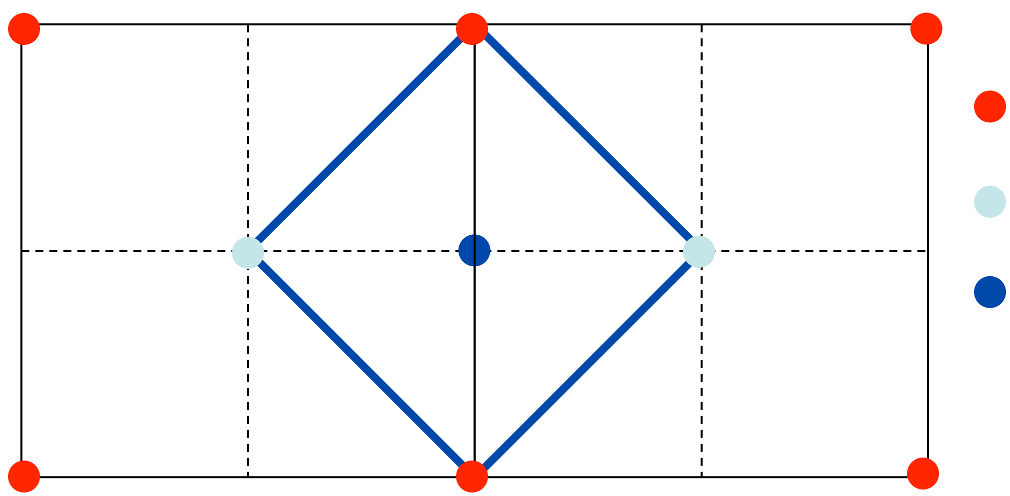
$y_E = (y_A + y_B + y_C + y_D)/4 + g$



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

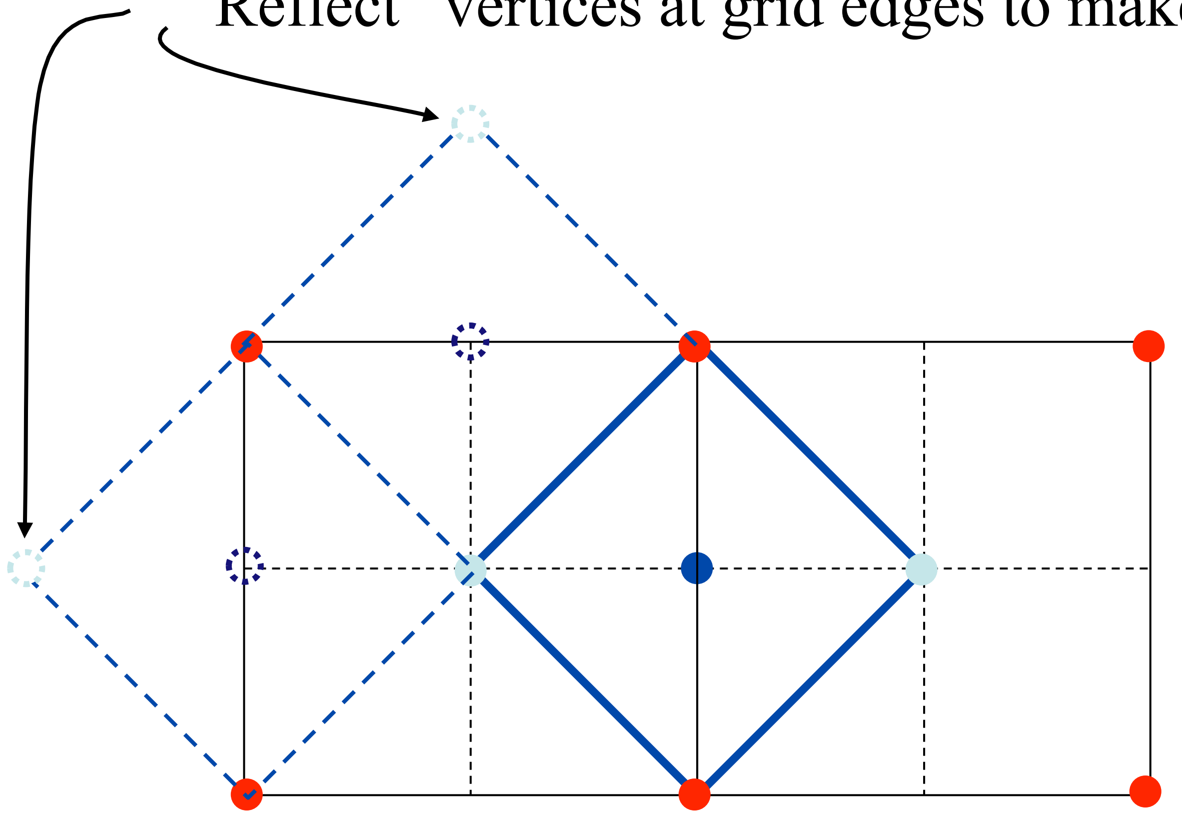
New vertices from square step

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).

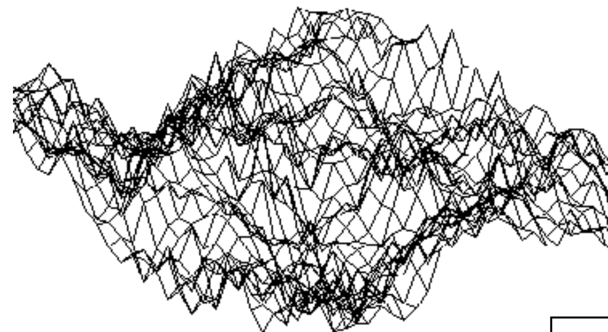
Diamond step (cont’d)

“Reflect” vertices at grid edges to make diamonds there.

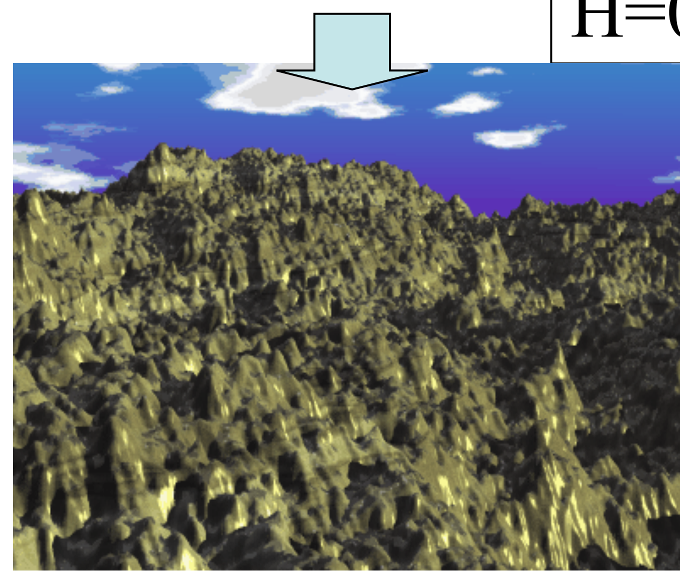


Diamond-Square Algorithm

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

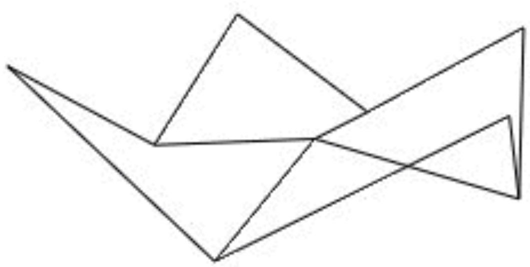


H=0.4

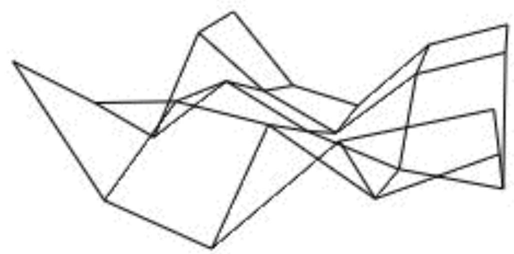


H=0.8

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)

Source for Most of this Material

- Much of the material covered in this lecture came from excellent material from a course on Procedural Content Generation by Julian Togelius, and a good book by Julian, Noor Shaker, and mark Nelson from ITU:
  - http://game.itu.dk/
  - http://pcgbook.com/