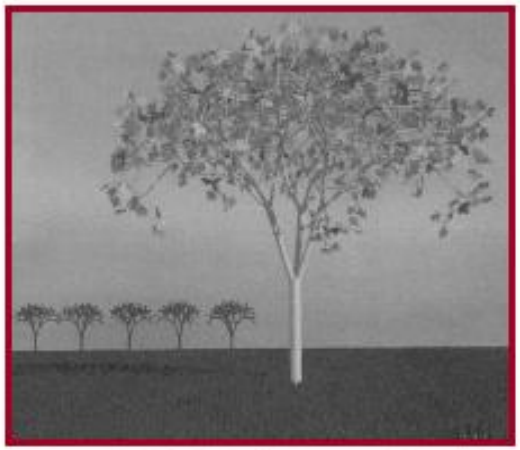
Fundamentals of Procedural Modeling

Modeling

- The subject of rendering covers techniques for generating images of complex models, but says little about the creation of those models
- Within computer graphics, the subject of *modeling* is as complex as the subject of rendering
- The demands of modern computer graphics call for the use of extremely complex models containing millions or billions of primitives
- Examples include scenes in modern special effects movies, or industrial models of buildings and vehicles

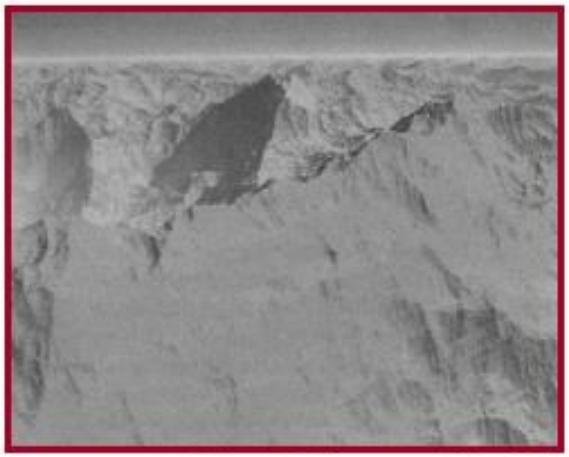
Modeling

- How do we …
  - Represent 3D objects in a computer?
  - Construct such representations quickly and/or automatically with a computer?
  - Manipulate 3D objects with a computer?



H&B Figure 10.79

Fowler



H&B Figure 10.83b

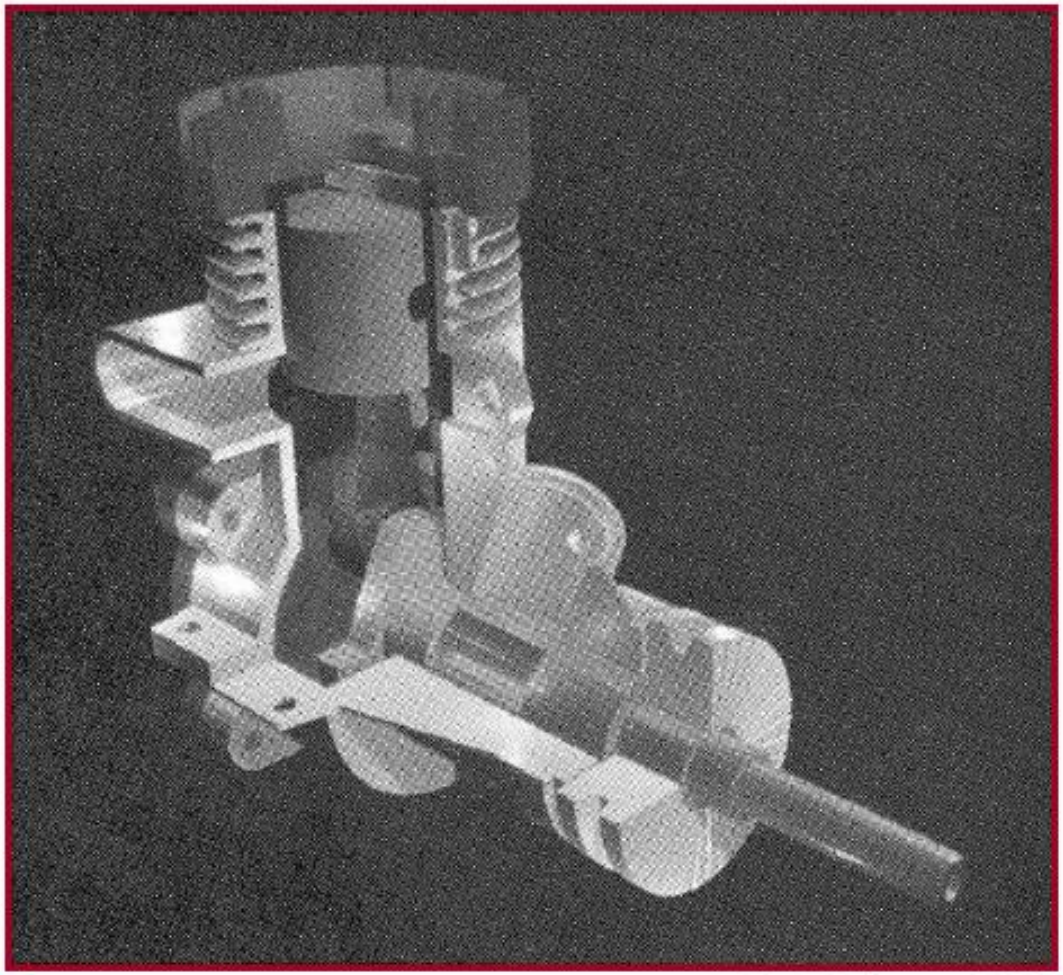
Model Creation

Models are typically built using one or more of the following methods

- *Interactive modeling*
  - Model is constructed by a human using a software modeling tool
- *Procedural modeling*
  - Model constructed by automatic procedure that may make use of randomness for variety
- *Scanning*
  - Model geometry is scanned from a real world example using a laser scanner or similar device
- *Computer vision*
  - Model geometry & material information is scanned from real world example using multiple photographic camera angles (or video sequences)

Interactive Modeling Tools

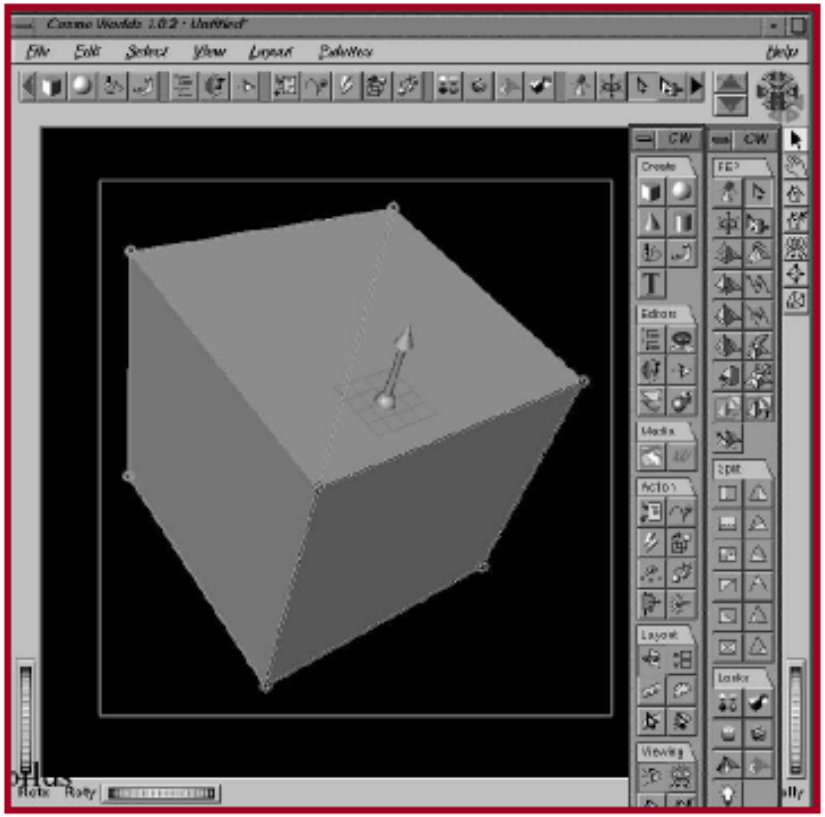
- Example: Mechanical CAD



H&B Figure 9.9

Interactive Modeling Tools

- User constructs objects with drawing program
  - Menu commands, direct manipulation, etc.
  - CSG, parametric surfaces, quadrics, etc.



Cosmoworlds, SGI

Model Create / Delete

- The most basic operations are:

```
Vertex *CreateVertex();
void DeleteVertex(int v);

Triangle *CreateTriangle();
void DeleteTriangle(int t);
```

- Just about all higher-level modeling functions can be broken down into these basic operations
- All higher-level functions go through these interfaces to create and remove data
- These functions need to be fast and reliable
- The ‘delete’ operations can be done in different ways and are not as simple as they might first look….

Shape Primitives

- Many real world objects contain basic shapes like spheres, boxes, cylinders, cones, etc.
- Sometimes, complex models can be built entirely from these simple shapes
- Modeling tools should have functions for creating a variety of primitive shapes like these

Copy

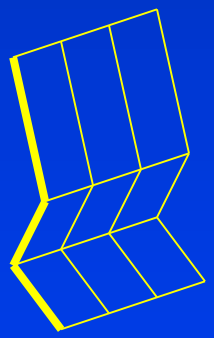
- One of the most basic modeling tools is the simple copy operation
- Models can be built up from multiple copies of simpler shapes
- A copy operation would probably take a source and destination object as well as a matrix as input
- It would add new vertices and triangles to the destination object by transforming the vertices of the source object by the matrix

Duplicate

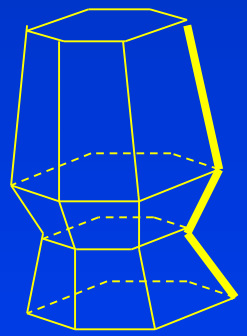
- The *dupe* or *duplicate* operation is a more complex variation on copy
- There are several variations on dupe operations and there really isn't any standard on this stuff
- A common dupe operation might take a source object as well as a group of points as input and generate a copy of the source object at every point
- More complex dupe operations might take several source objects as input and choose a random one to place at the point and might apply additional randomness such as a random rotation or slight variation in the scale
- This can be used to do things such as placing a bunch of trees along the side of a road, for example

Extrude / Lathe

- Many useful shapes can be constructed by extruding or lathing a line (or curve)
- The extrude operation generates a surface by connecting copies of the line that have been placed in a straight line
- The lathe operation works in a similar way, except the copies are rotated around a circle



Extrude



Lathe

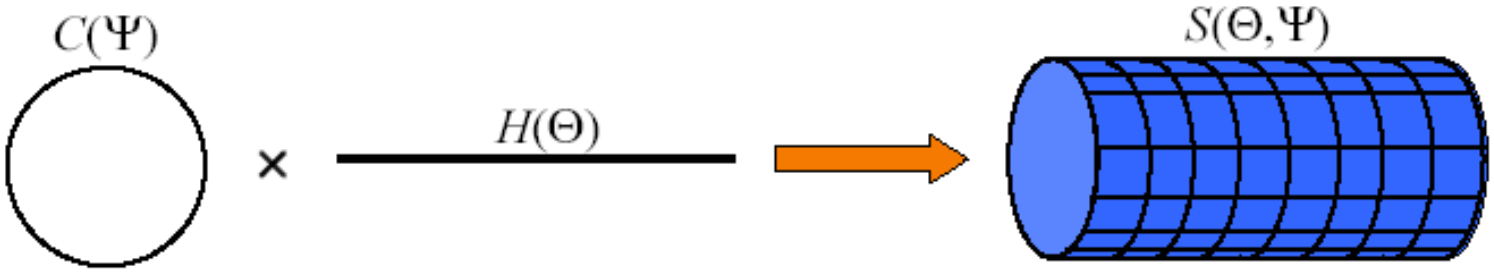
Procedural Modeling

Procedural Modeling

- Goal:
  - Describe 3D models algorithmically
- Best for models resulting from ...
  - Repeating processes
  - Self-similar processes
  - Random processes
- Advantages:
  - Automatic generation
  - Concise representation
  - Parameterized classes of models

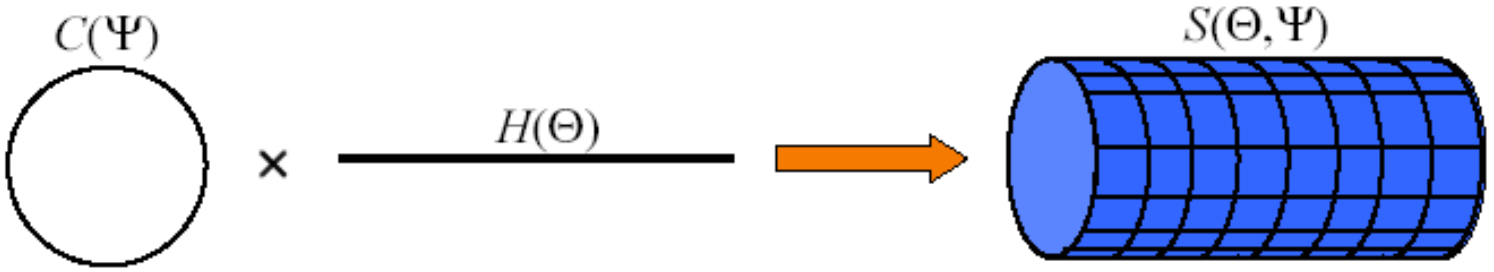
Sweeps

Given a 3D sweep curve $H(\Theta)$ and a 2D generating curve $C(\Psi)$, we define the sweep surface $S(\Theta,\Psi)$ as the sweep of C along H:



Sweeps

Given a 3D sweep curve $H(\Theta)$ and a 2D generating curve $C(\Psi)$, we define the sweep surface $S(\Theta,\Psi)$ as the sweep of C along H:



In this example, the sweep curve is simply used to translate the generating curve:

$$S(\Theta,\Psi) = H(\Theta) + C(\Psi)$$

We can define more complex sweep surfaces.

Example: Seashells

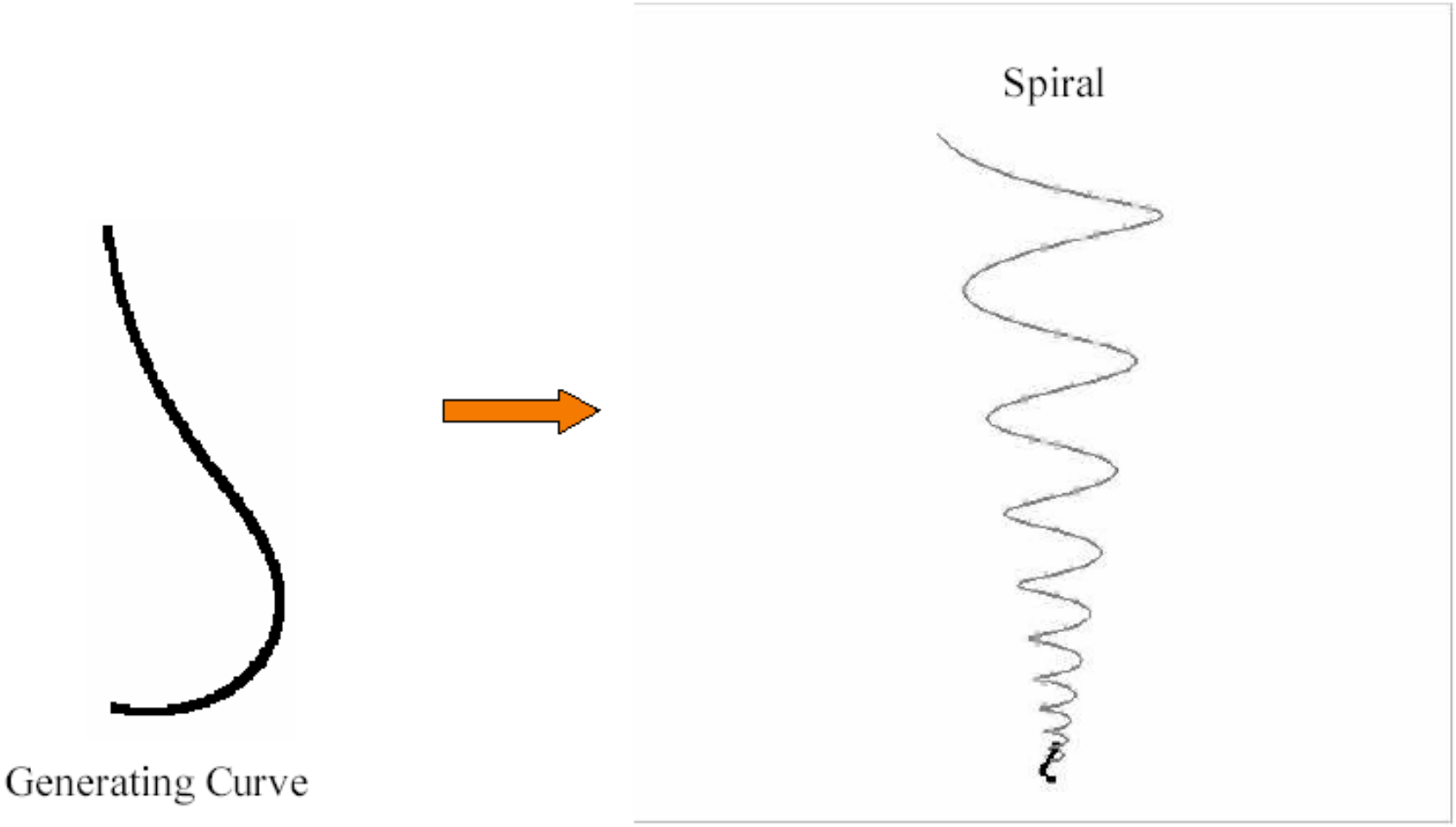
- Create 3D polygonal surface models of seashells

"Modeling Seashells,"
Deborah Fowler, Hans Meinhardt,
and Przemyslaw Prusinkiewicz,
Computer Graphics (SIGGRAPH 92),
Chicago, Illinois, July, 1992, p 379-387.

Fowler et al. Figure 7

Example: Seashells

- Sweep generating curve around helico-spiral axis



Example: Seashells

- Sweep generating curve around helico-spiral axis

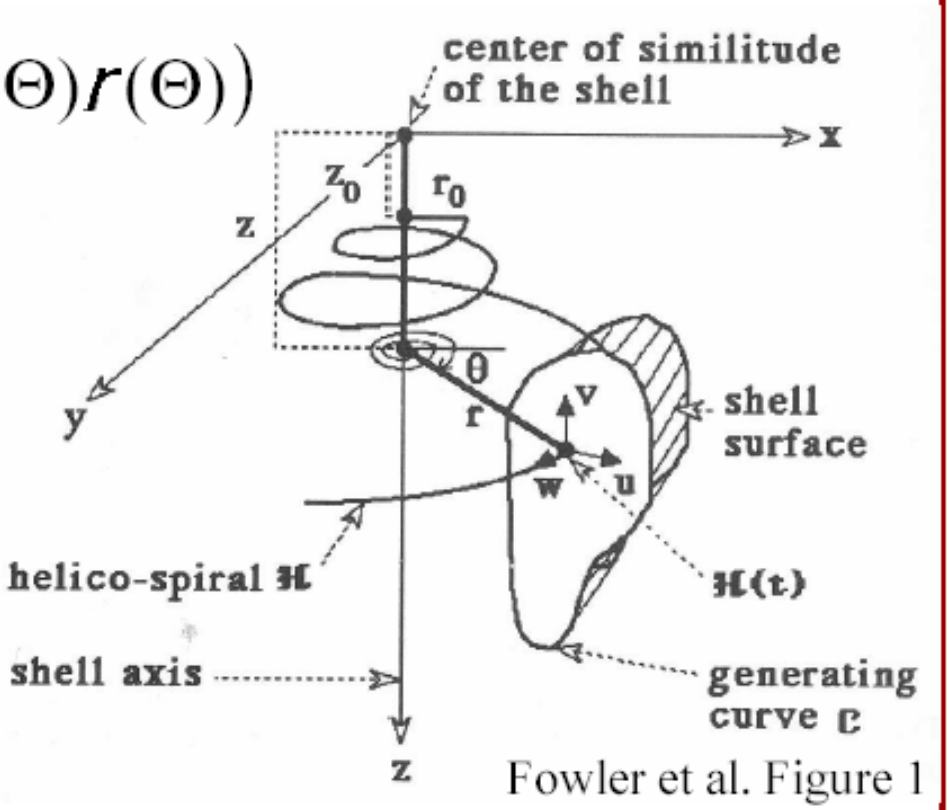
Helico-Spiral definition:

$$H(\Theta) = \left(\cos(\Theta) r(\Theta), z(\Theta), \sin(\Theta) r(\Theta)\right)$$

Θ (angle)

$r(\Theta) = e^{\lambda\Theta}$ (radius)

$z(\Theta) = e^{\mu\Theta}$ (height)



Fowler et al. Figure 1

Example: Seashells

- Sweep generating curve around helico-spiral axis

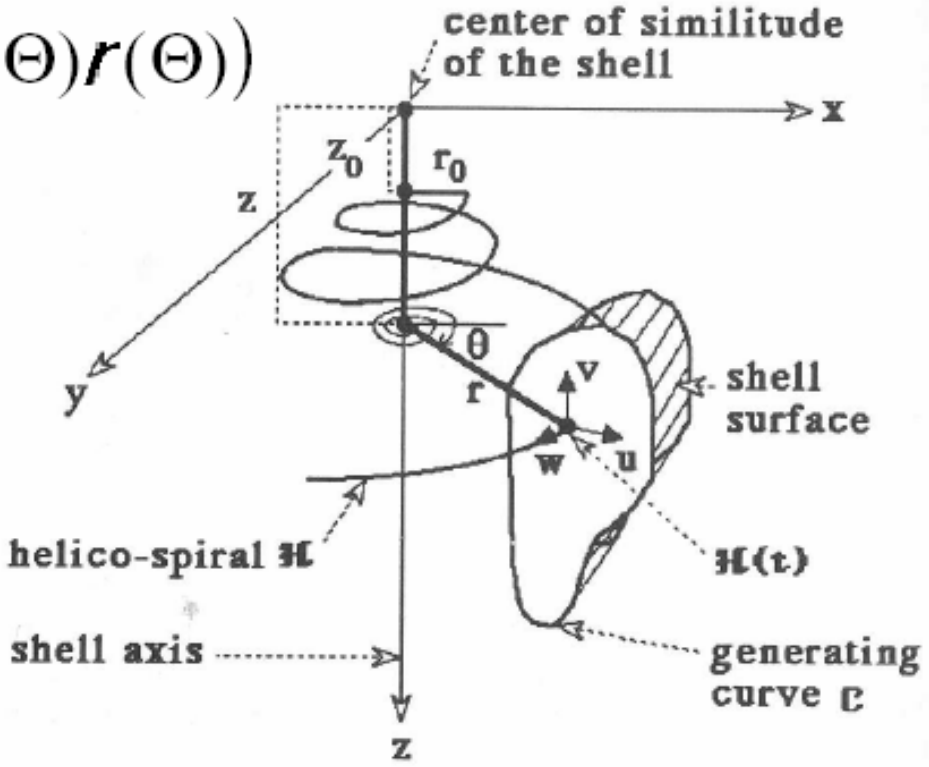
Helico-Spiral definition:

$$H(\Theta) = \left(\cos(\Theta) r(\Theta), z(\Theta), \sin(\Theta) r(\Theta)\right)$$

Θ (angle)

$r(\Theta) = e^{\lambda\Theta}$ (radius)

$z(\Theta) = e^{\mu\Theta}$ (height)



Shell-Surface Definition:

$$S(\Theta,\Psi) = H(\Theta) + \left(u(\Theta) C_x(\Psi) + v(\Theta) C_y(\Psi)\right) r(\Theta)$$

Example: Seashells

- Sweep generating curve around helico-spiral axis

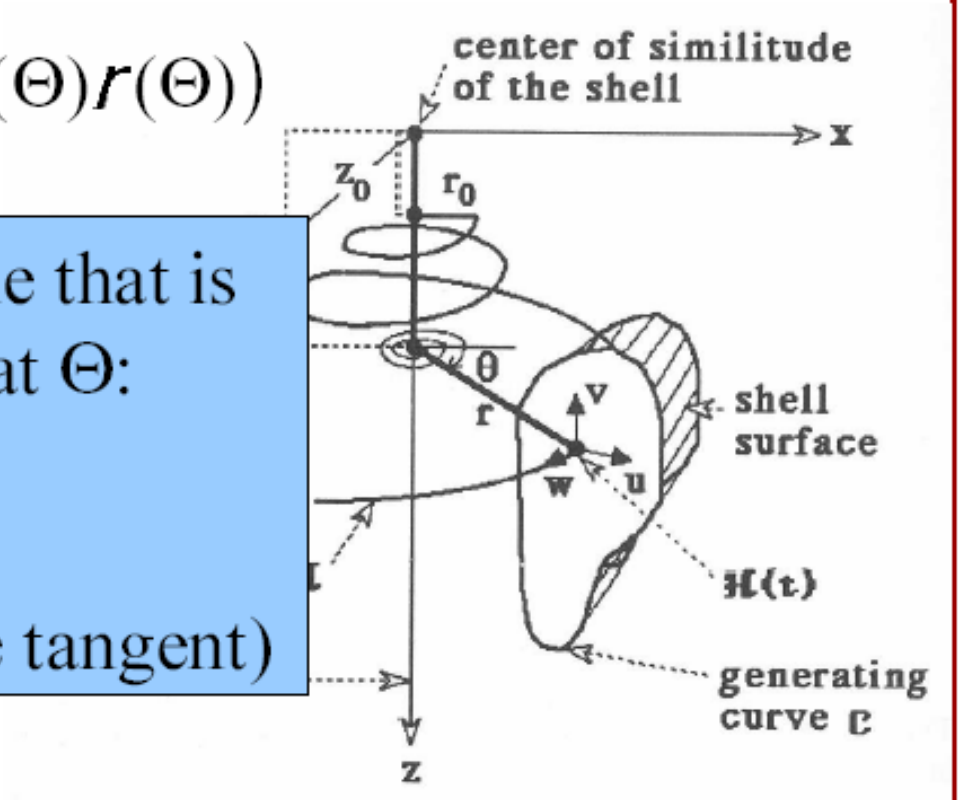
Helico-Spiral definition:

$$H(\Theta) = \left(\cos(\Theta) r(\Theta), z(\Theta), \sin(\Theta) r(\Theta)\right)$$

Θ (angle)

$u(\Theta)$ and $v(\Theta)$ define the plane that is perpendicular to the curve H at Θ:

- $u(\Theta)$ is the curve normal
- $v(\Theta)$ is the curve bi-tangent (perp. to $u(\Theta)$ and the curve tangent)



Shell-Surface Definition:

$$S(\Theta,\Psi) = H(\Theta) + \left(u(\Theta) C_x(\Psi) + v(\Theta) C_y(\Psi)\right) r(\Theta)$$

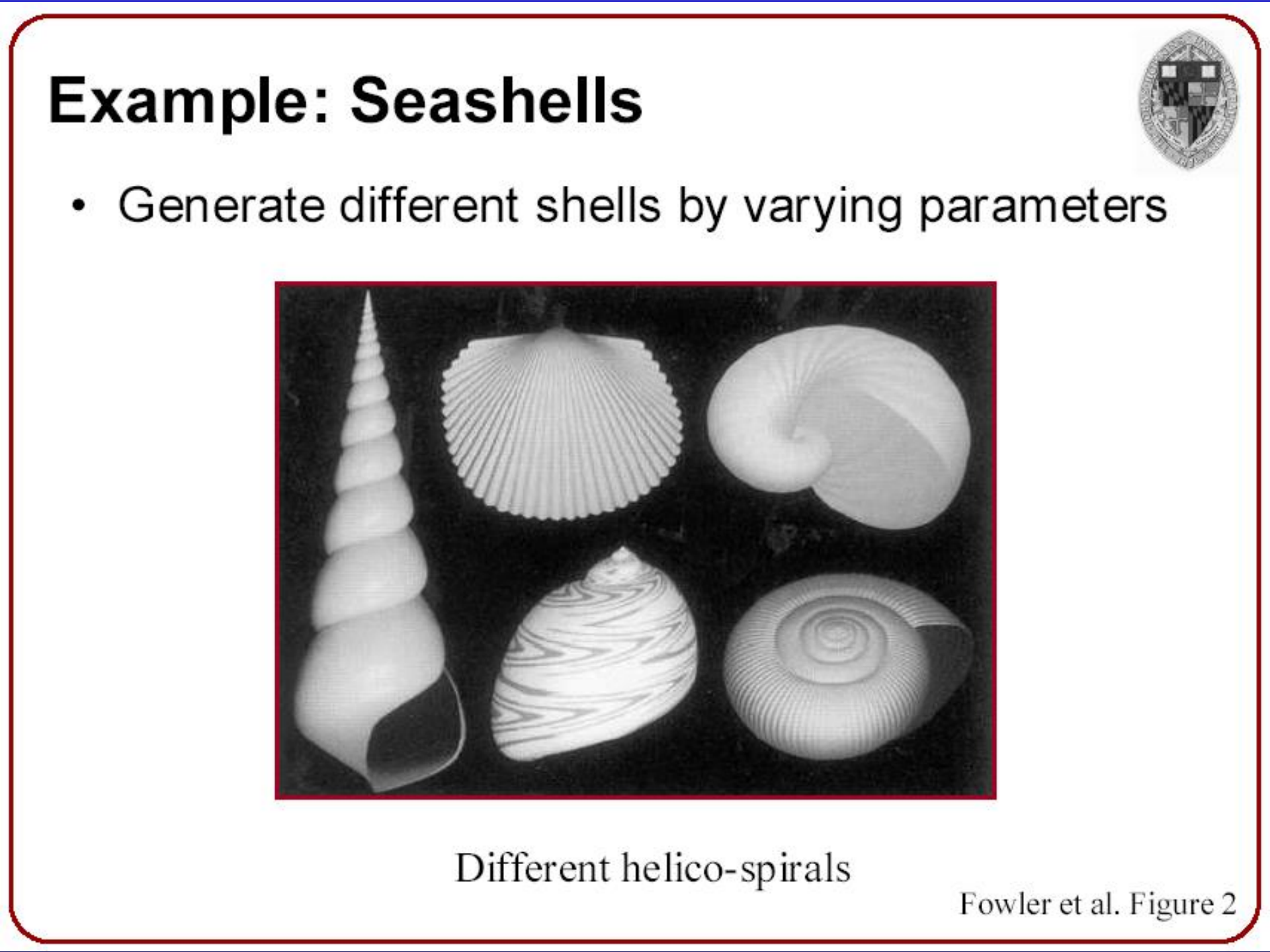
Path Extrude

- A powerful variation on extrusion is the *path extrude* operation
- With this one, we have one or more lines (or curves) that make up a *cross section* and a second line (or curve) that makes up the *path*
- The path extrusion connects several copies of the cross section along the path that orient to the path as it turns
- This can be used to make a tree trunk, or a freeway overpass (or tunnel), for example
- The cross section could also vary along the path to allow for additional control

Lofting

- There are also a variety of *lofting* tools that can be used to create surfaces out of a set of input lines (or curves)
- For example, various lofting tools can be used to model shapes like boat hulls, airplane wings, and car bodies

Example: Seashells



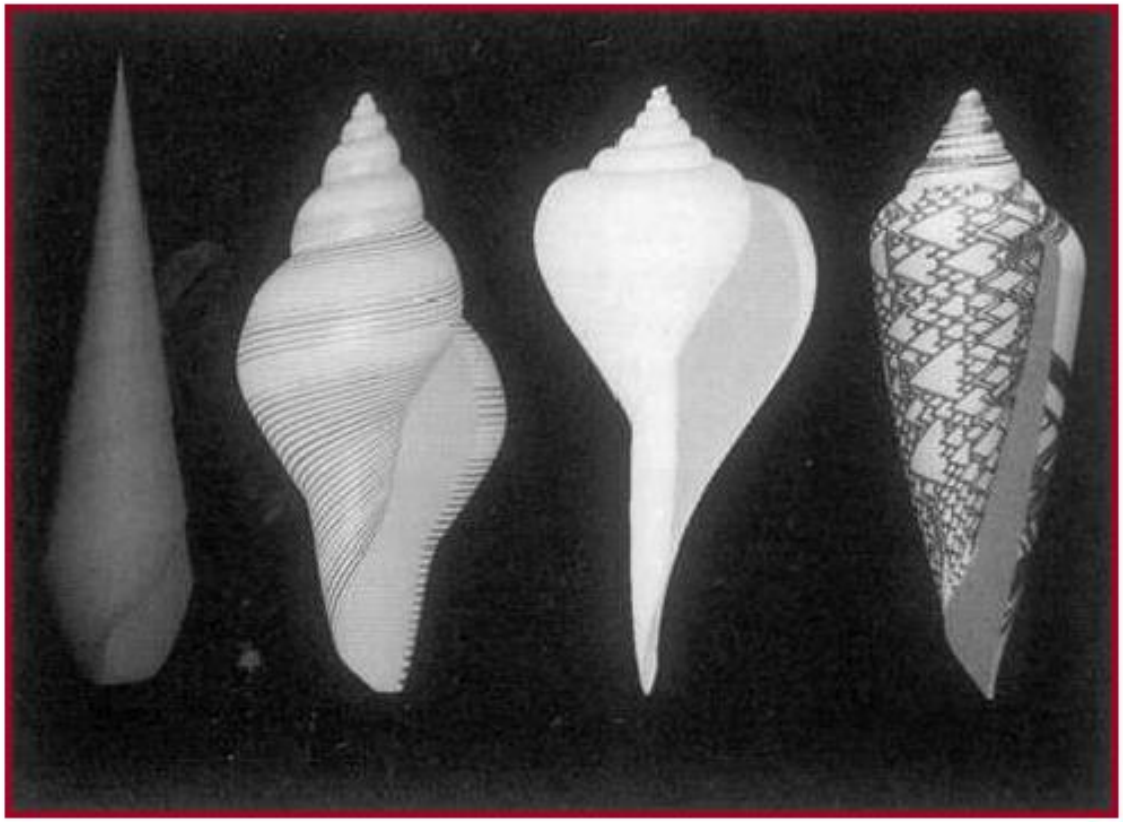
- Generate different shells by varying parameters

Different helico-spirals

Fowler et al. Figure 2

Example: Seashells

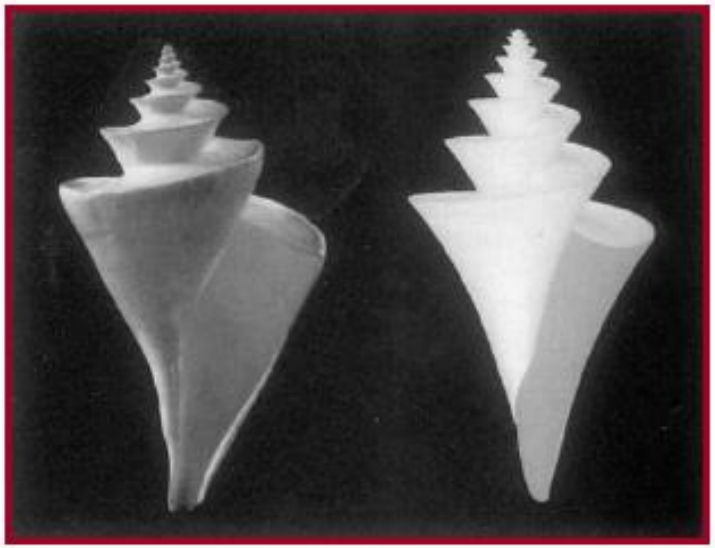
- Generate different shells by varying parameters



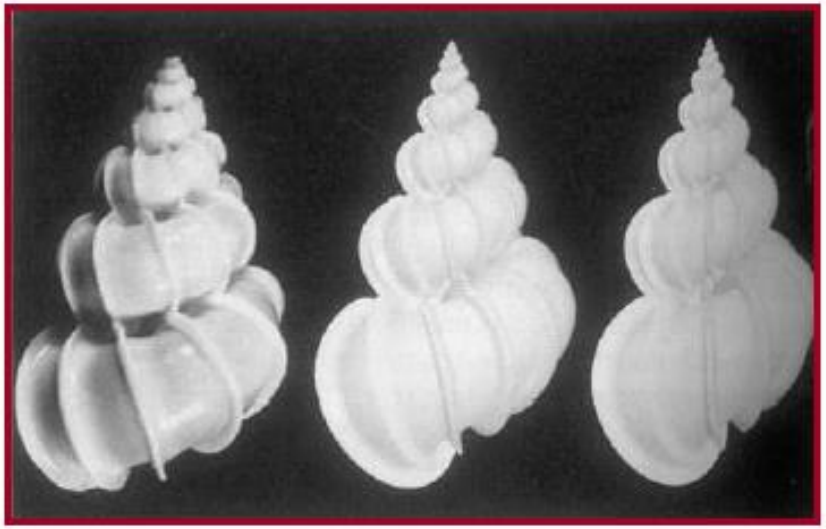
Different generating curves

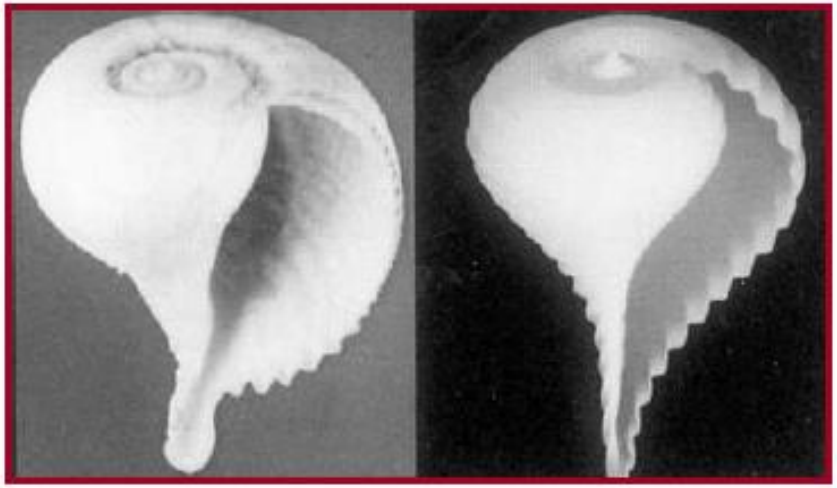
Fowler et al. Figure 3

Example: Seashells



Generate many interesting shells with a simple procedural model!





Fowler et al. Figures 4,5,7

Boolean

- Boolean operations can be used to compute unions, intersections, and subtractions with complex 3D shapes
- Many industrial models are build from Boolean operations

Basic Modeling Operations

- The modeling operations we have discussed so far make up some of the most common functions found in interactive modeling tools
- They are also the foundation of more automated procedural modeling tools
- These operations have been used for many years and continue to be useful
- One reason for their popularity is that they directly relate to the way that many objects are designed and built in the real world

"Implicit" vs. "Explicit" Procedural Models

- Explicit approach:
  - Directly generate the points that make up an object
  - Good for Z-buffer/OpenGL style rendering
- Implicit approach:
  - Answer questions about particular points
  - Isocurve (2D) or Isosurface (3D)
  - Good for ray-tracing/ray-casting

Simple Explicit Procedural Model

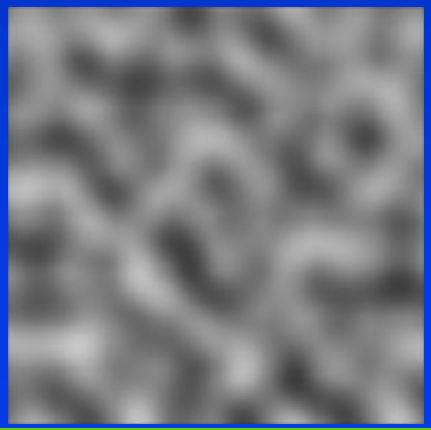
- Begin with a regular mesh
- Perturb vertex geometry procedurally (typically pseudo-randomly)
- Iterate this process until desired shape is achieved
- Very general technique that can also be used to add irregularity (“noise”) to arbitrary mesh objects

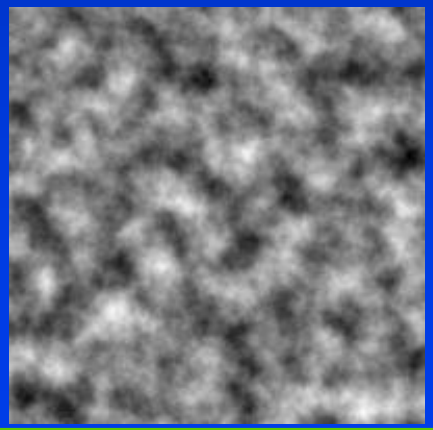
Randomness

- Procedural models often make use of some form of randomness
- A simple random number generator is usually sufficient for many operations
- As computers can not usually generate *true* random numbers, they typically make use of *pseudorandom* number generation algorithms
- A pseudorandom number generator outputs a sequence of (apparently) random numbers based on some initial seed value
- In this sense, the sequence is repeatable, as one can always reset the sequence
- For example, if a procedural model like a tree is built from by making use of several random numbers (maybe hundreds), then the entire tree can be rebuilt by just resetting the seed to its initial value
- If the seed is set to a different value, a different sequence of numbers will be generated, resulting in a slightly different tree

Noise

- Another form of randomness which is sometimes useful for procedural modeling is *noise*
- Noise represents a distribution of randomness over some space (usually 2D or 3D)
- Noise is not entirely random, as two points nearby will have a similar value
- In this way, noise has a frequency associated with it
- By combining noise patterns of different frequencies, one can make more complex *turbulence* patterns





Fractals

- A fractal is a geometric object that is self-similar when viewed at different scales
- For example, the shape of a coastline may appear as a jagged line when we view a map of Long Island. As we zoom in closer and closer, we see that there is more and more detail at finer scales. We always see a jagged line no matter how close we look at the coastline

Fractals

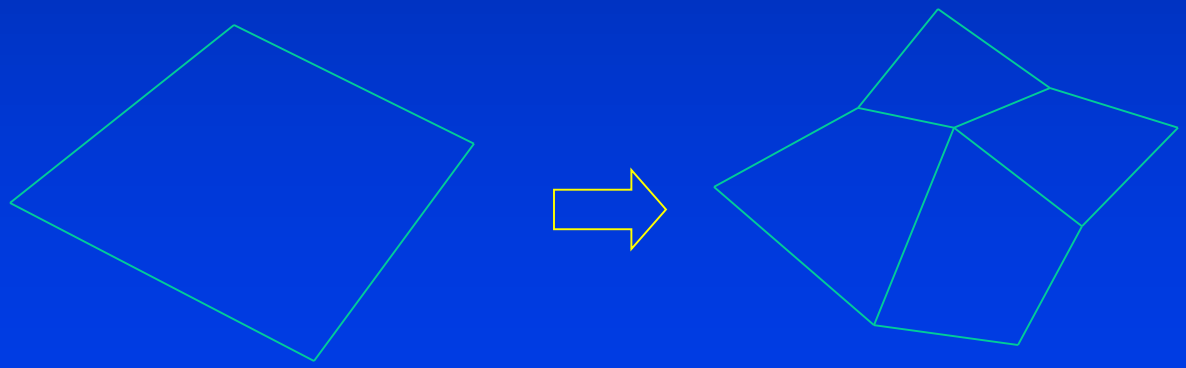
- Fractals can be regular repeated patterns, or can be irregular and incorporate randomness as well
- Random fractals are useful for creating a wide variety of natural shapes such as mountain landscapes
- Even trees can be thought of as a fractal, as the branching patterns are similar when one looks at the main trunk down to the finest branches
- For procedural modeling, we may borrow some fractal concepts, but we rarely deal with true mathematical fractals with infinite detail
- We usually think of fractals as techniques for generating randomness in some limited range of scales

Fractals

- Consider a simple line fractal
- We start with a single line segment and then split it in the middle, randomizing the height of the midpoint by some number in the [-*r*,*r*] range
- We then split each of the new line segments at the middle and randomize them by [-*r*/2,*r*/2]
- This process is repeated some desired number of steps, randomizing by half as much each step

Fractals

- A similar process can be applied to squares in the xy plane
- At each step, an xy square is subdivided into 4 squares, and the z component of each new point is randomized
- By repeating this process recursively, we can generate a mountain landscape

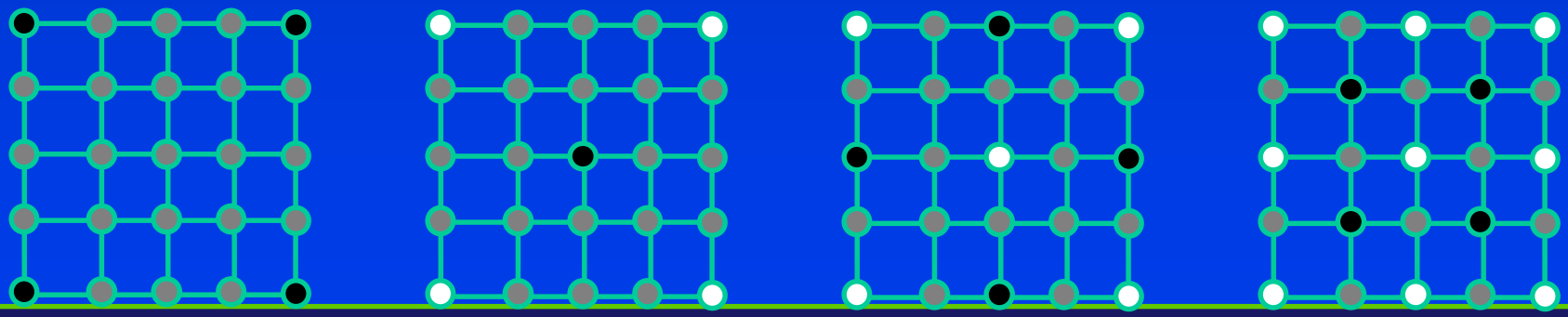


Height Fields

- Landscapes are often constructed as *height fields*
- In a height field, we assume a regular grid in the ground plane (for us, that is the xy plane)
- At each grid point, we store a height (z) value
- In this way, a large terrain can be stored in memory without explicitly storing the x & y coordinates of the vertices or the triangle connection information
- The terrain can be shaped by operations that modify the z coordinates
- In a lot of ways, shaping the terrain is like rendering an image, where the heights of the cells in the height fields can be compared to the pixel colors in an image
- Similar tools can be used to shape the height field to the tools used in rendering, such as the use of triangles or noise patterns

Midpoint Displacement For Terrain

- Seed corners with values
- Perturb midpoint randomly from mean
- Recursion using a smaller window
- In 2D, best to use “diamond-square” recursion (to prevent axis-aligned artifacts)

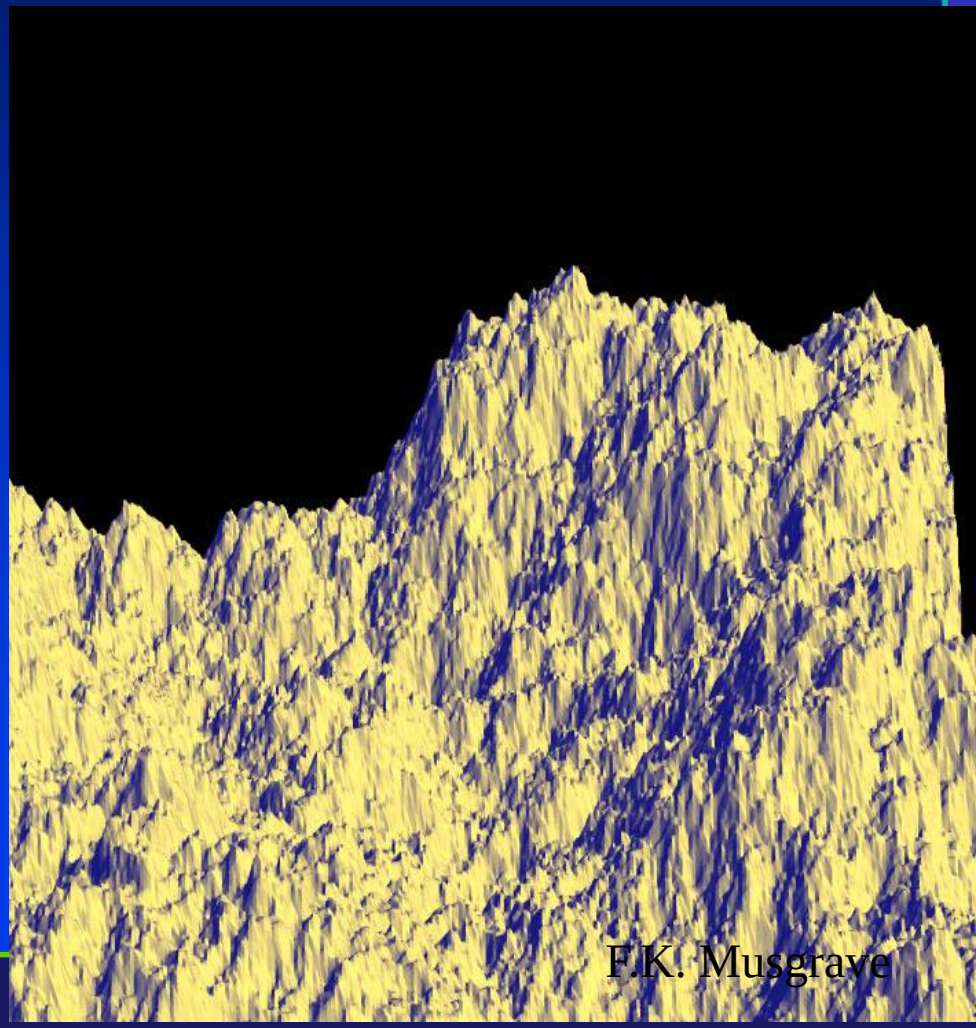


One Example: Natural Terrain Modeling

Fractal Noise Terrain

- Use fractal noise to generate terrain
- Can be made tile-able over unit square:

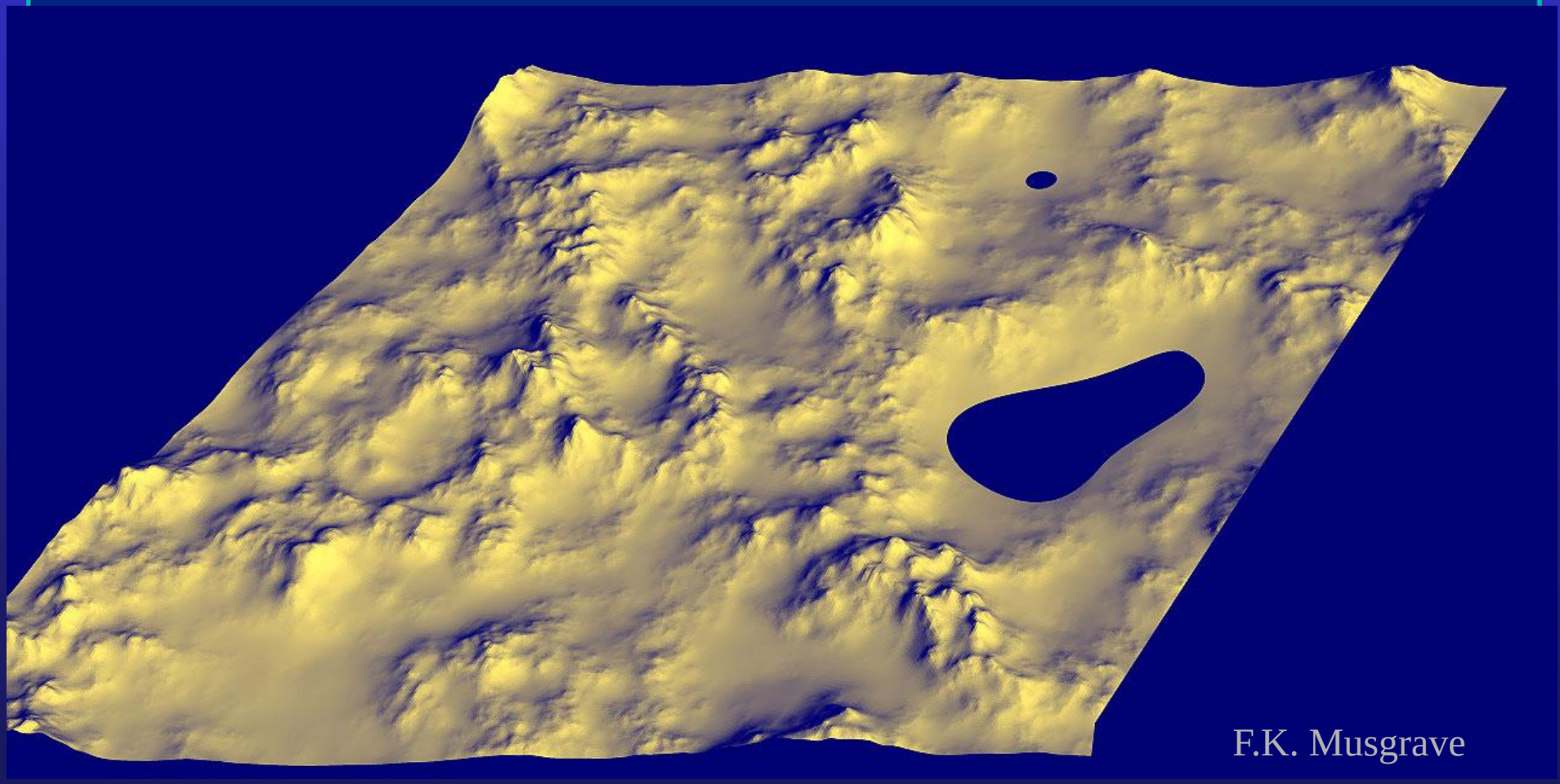
$$F_{tileable}(x,y) = [$$
$$F(x,y) * (1-x) * (1-y) +$$
$$F(x-1,y) * x * (1-y) +$$
$$F(x-1,y-1) * x * y +$$
$$F(x,y-1) * (1-x) * y]$$



F.K. Musgrave

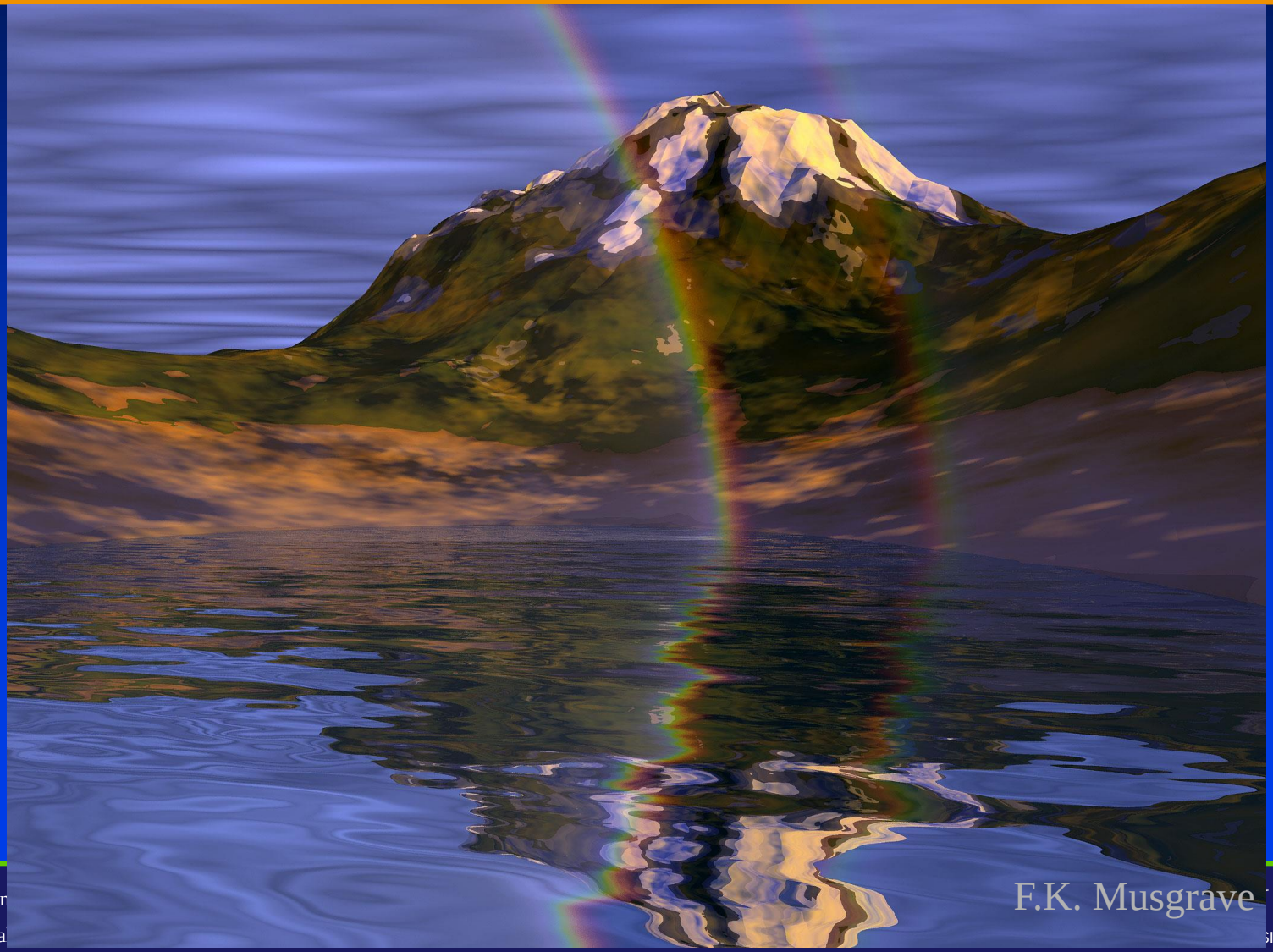
Adding Water

- **Use an elevation threshold ($z < z_{water}$)**



Terrain Example

Terrain Example



Terrain Example

Terrain Example

Height Fields

- The height field itself is an efficient data structure for storing the shape of the terrain, but it still must be converted to triangles to render
- We could simply generate a grid of triangles
- However, if we use a grid, we will end up with too many triangles in flat regions and too high of a triangle density off in the distance
- It would be better to perform some sort of adaptive tessellation of the height field, much like the tessellations used in patch rendering and displacement mapping

Quadtree Tessellation

- One way to triangulate height fields adaptively is through the use of a *quadtree*
- The quadtree is a 2D data structure that is usually based on rectangles or squares
- It works in a very similar way as the fractal subdivision we just covered, except it can be used for triangulating height fields (or Bezier patches…)
- We start with single square around our whole terrain
- We perform some sort of analysis on the square and determine if it contains more detail than is adequately represented by a square
- If the detail is insufficient, the square is split into four smaller squares, which are recursively tested
- Ultimately, squares are then split into two triangles

Landscape

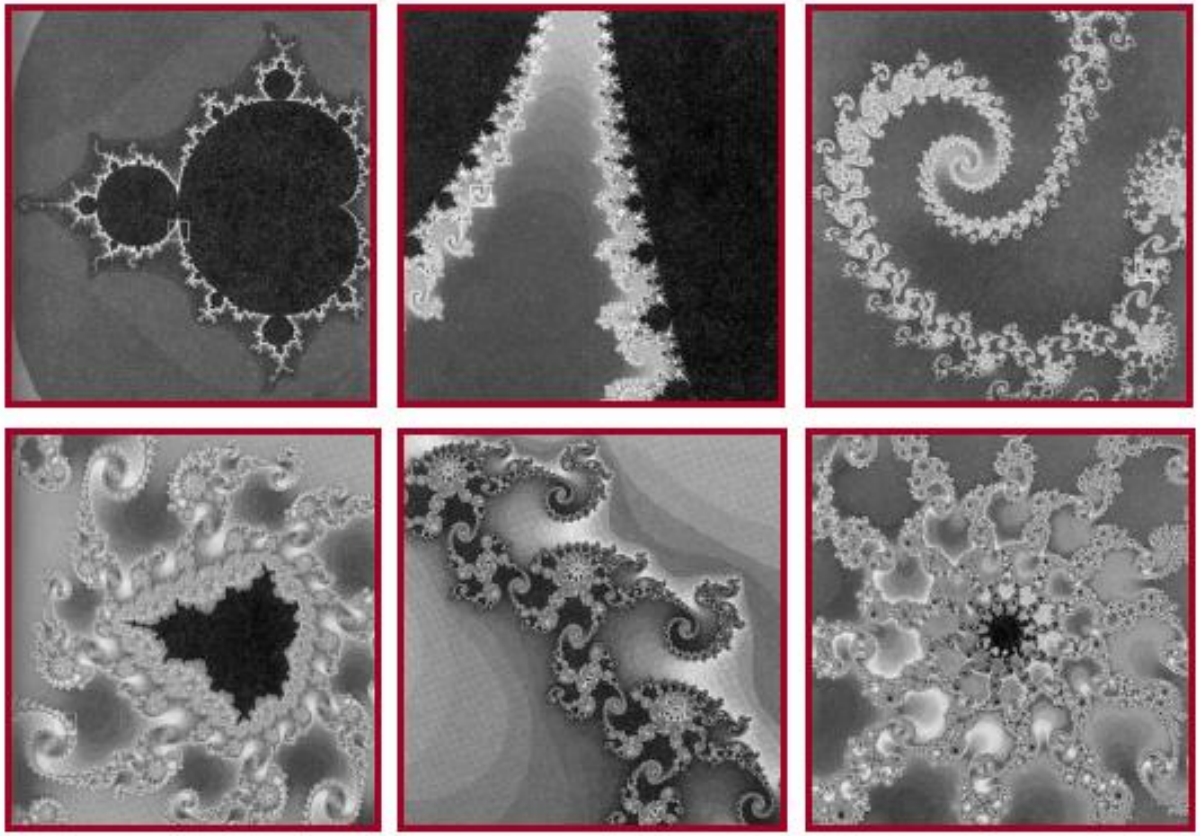
- By combining a variety of tools such as fractals, noise patterns, triangle rasterization, and others, one can build up a set of tools for modeling natural terrains (and man made modifications to terrain)
- One can also run simulations of erosion to achieve additional realism
- One can also use real world data of the Earth to model specific regions
- Geographic data exists in many formats, but one of the more useful ones is the *DEM* or *digital elevation map*, which is essentially a height field for a rectangular region of the Earth's surface
- The USGS has DEM files for the entire continental US at 10 meter resolution, and for the entire world at 30 meter resolution, available for free downloading!

Fractals

- Mandelbrot set
  - $$z_0 = z$$
    $$z_k = z_{k-1}^2 + z_0 \qquad k = 1,2,3\ldots$$
  - The boundary of the convergence region in the complex plane is fractal
  - To speed up, we use different color values according to the number of iterations executed by the loop.
  - Could zoom in/out of any particular regions

Fractals

- Defining property:
  - Self-similar with infinite resolution



Mandelbrot Set

H&B Figure 10.100

Fractals

- Useful for describing natural 3D phenomenon
  - Terrain
  - Plants
  - Clouds
  - Water
  - Feathers
  - Fur
  - etc.

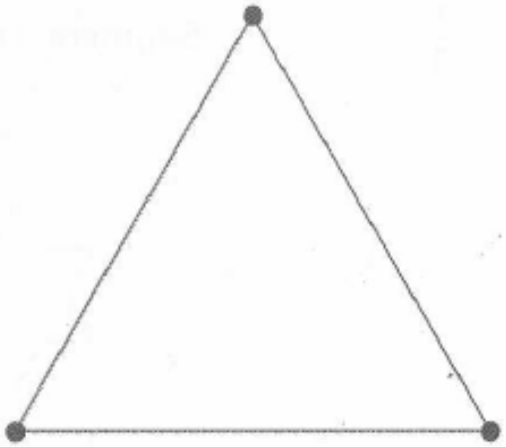
H&B Figure 10.80

Fractal Generation

- Deterministically self-similar fractals
  - Parts are scaled copies of original
- Statistically self-similar fractals
  - Parts have same statistical properties as original

Deterministic Fractal Generation

- General procedure:
  - Initiator: start with a shape
  - Generator: replace subparts with scaled copy of original



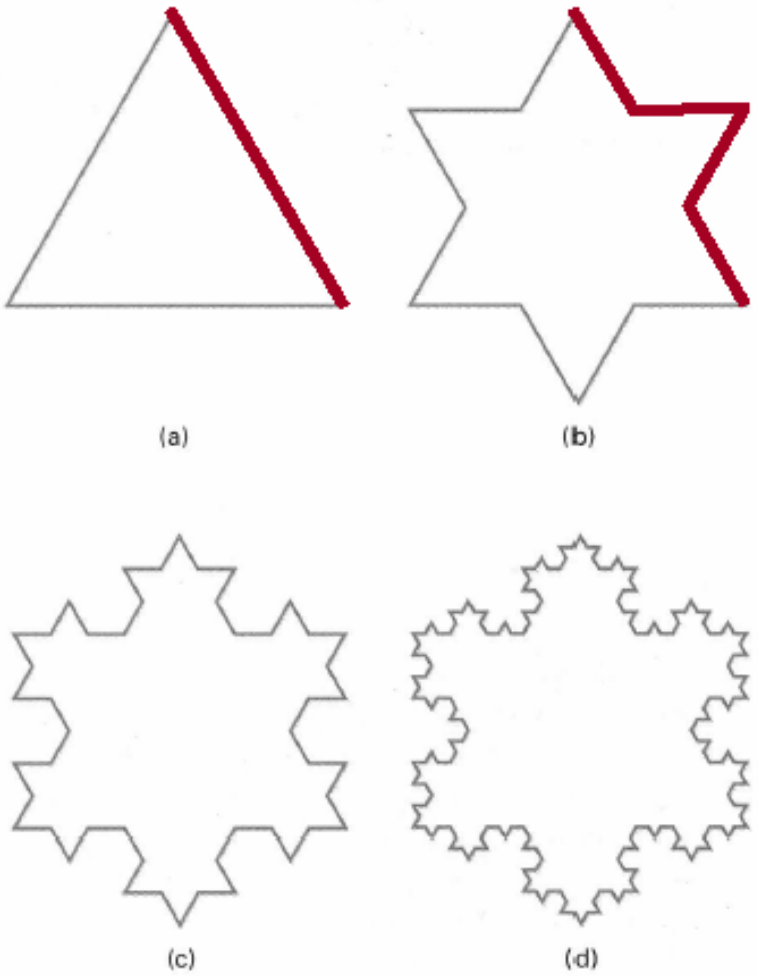
Initiator

Generator

H&B Figure 10.68

Deterministic Fractal Generation

- Apply generator repeatedly

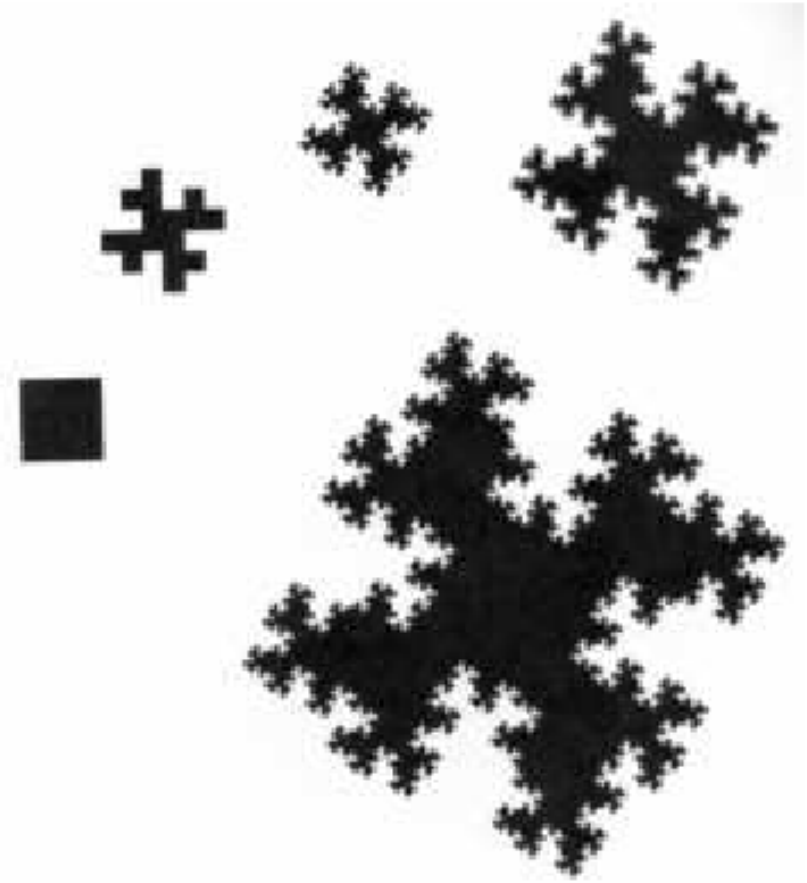


Koch Curve

H&B Figure 10.69

Deterministic Fractal Generation

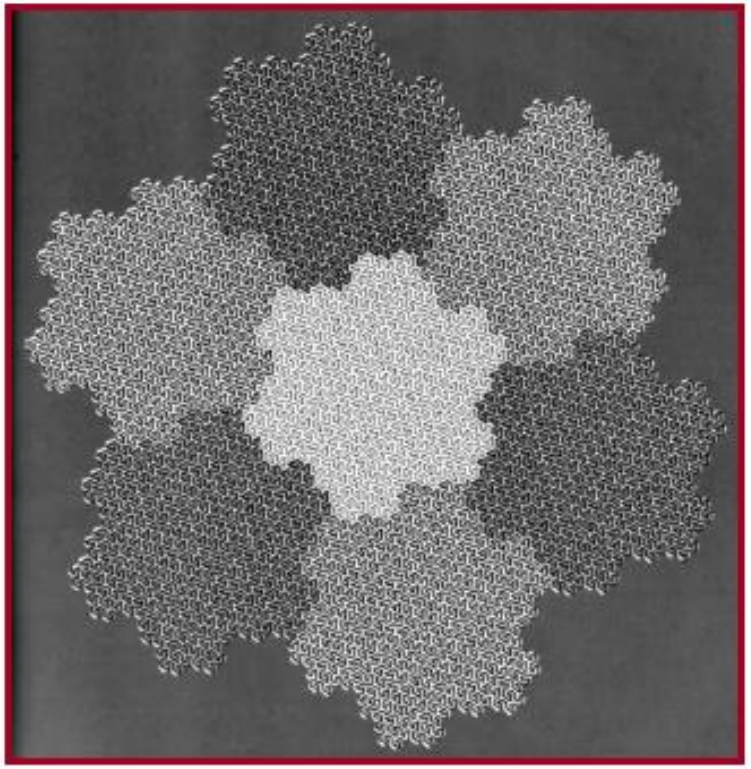
- Useful for creating interesting shapes!



Mandelbrot Figure X

Deterministic Fractal Generation

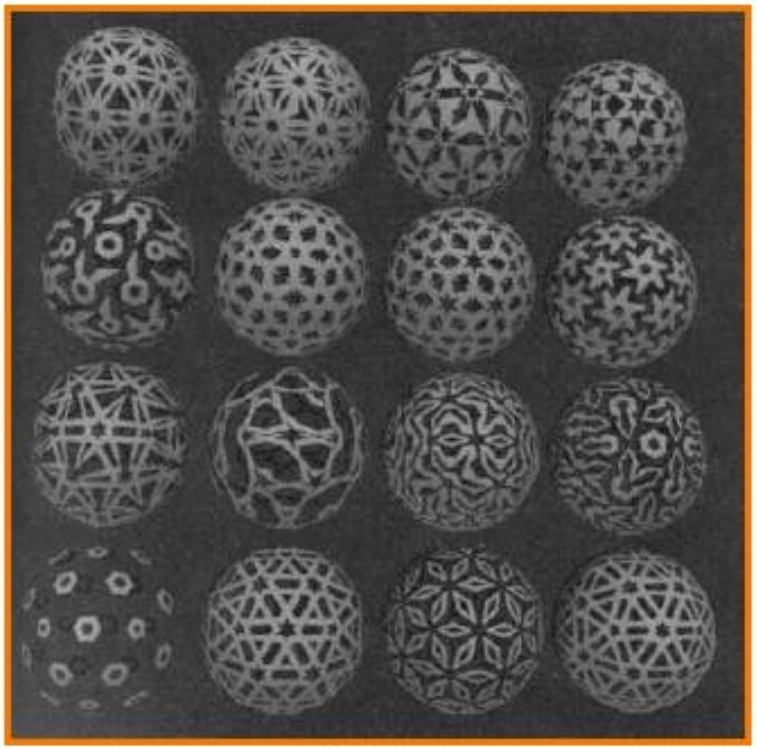
- Useful for creating interesting shapes!

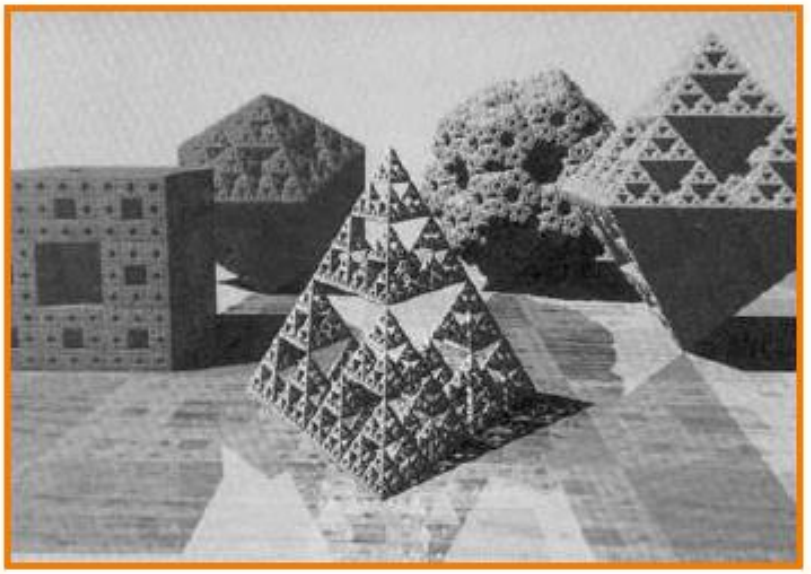


Mandelbrot Figure 46

Deterministic Fractal Generation

- Useful for creating interesting shapes!





H&B Figures 75 & 109

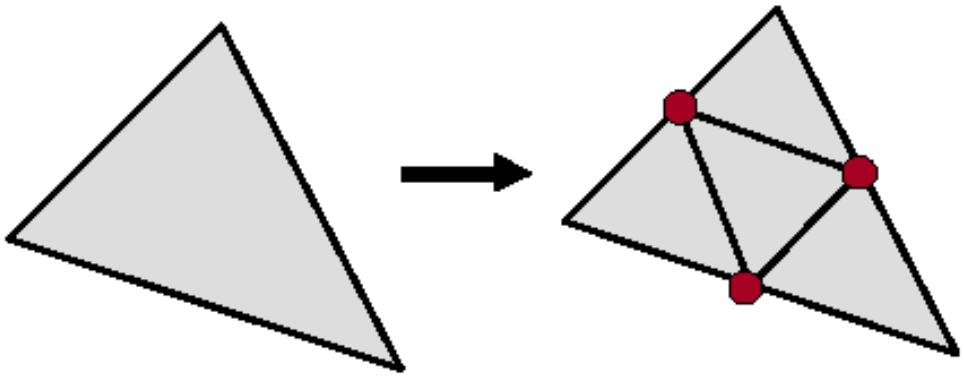
Fractal Generation

- Deterministically self-similar fractals
  - Parts are scaled copies of original

- Statistically self-similar fractals
  - Parts have same statistical properties as original

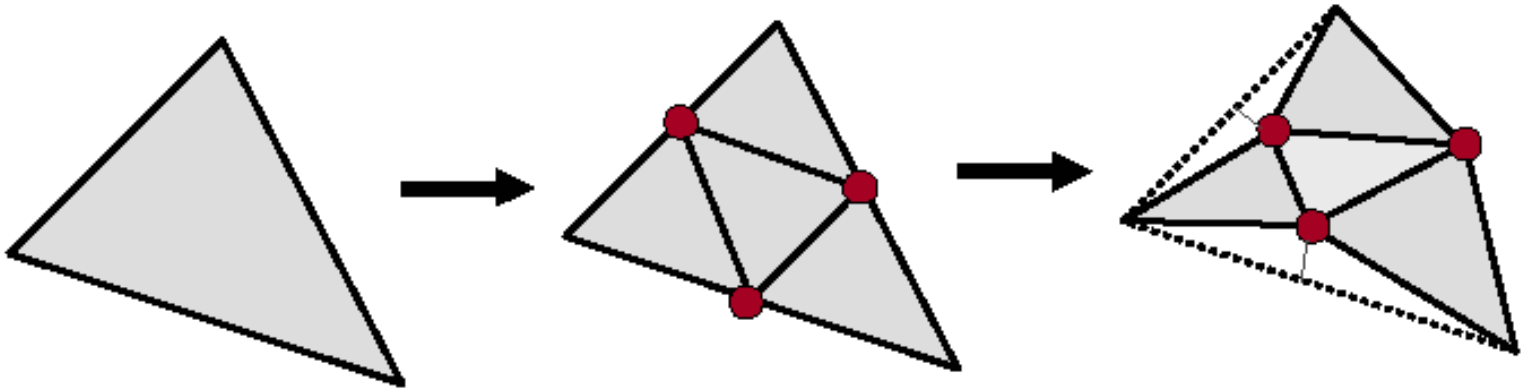
Statistical Fractal Generation

- General procedure:
  - Initiator: start with a shape
  - Generator: replace subparts with a self-similar



Statistical Fractal Generation

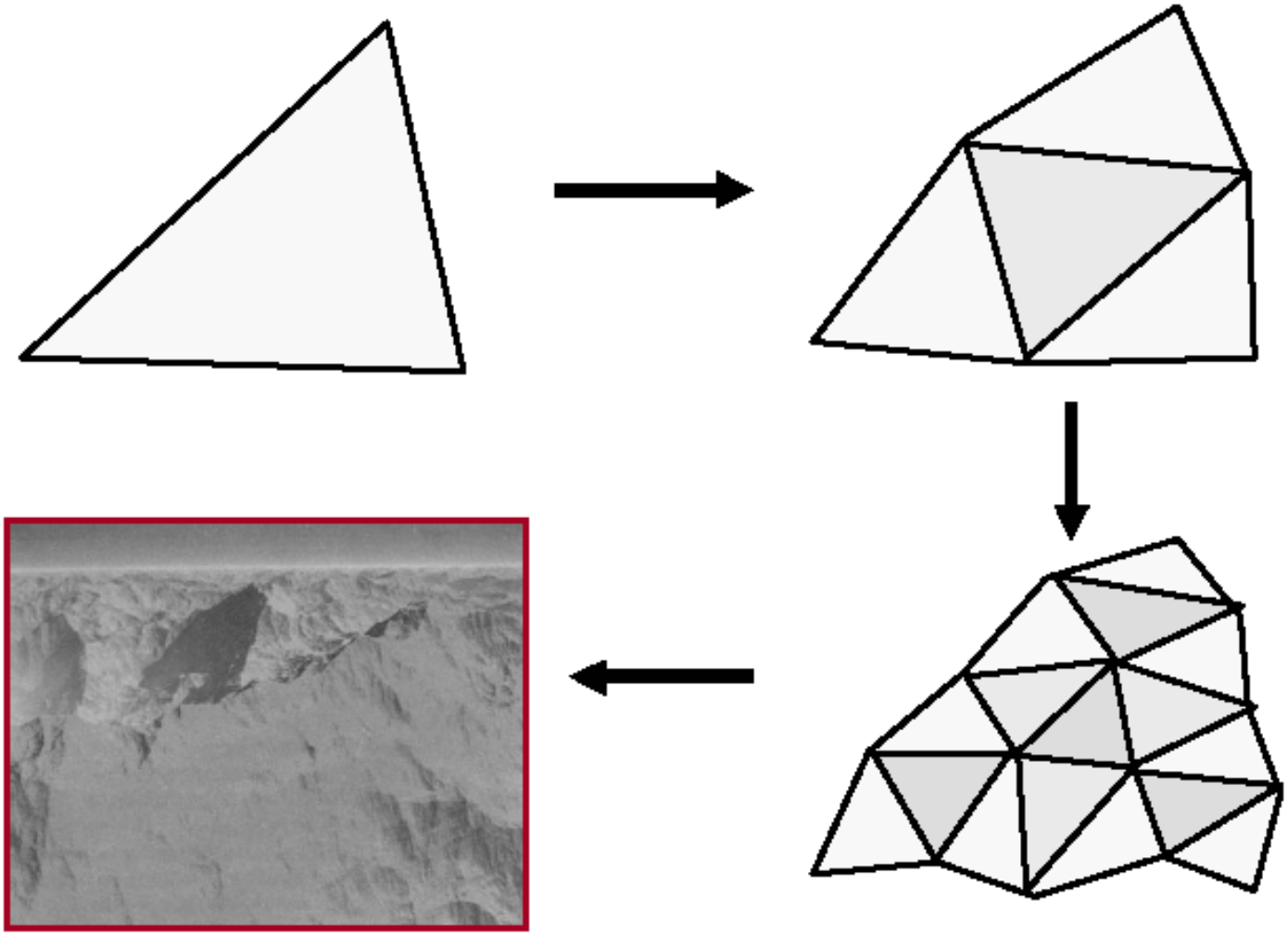
- General procedure:
  - Initiator: start with a shape
  - Generator: replace subparts with a self-similar random pattern



Random Midpoint Displacement

Statistical Fractal Generation

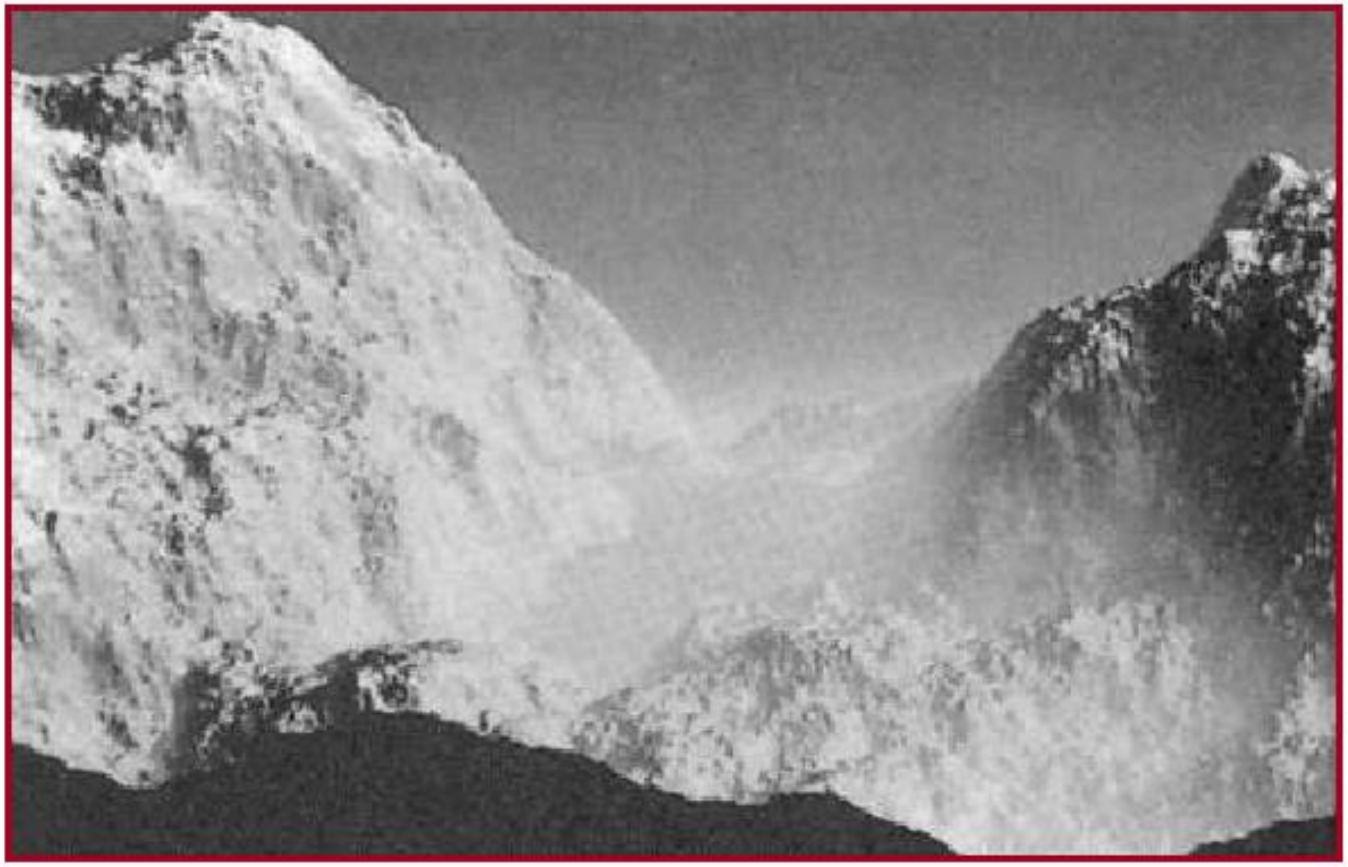
- Example: terrain



H&B Figure 10.83b

Statistical Fractal Generation

- Useful for creating mountains



H&B Figure 10.83a

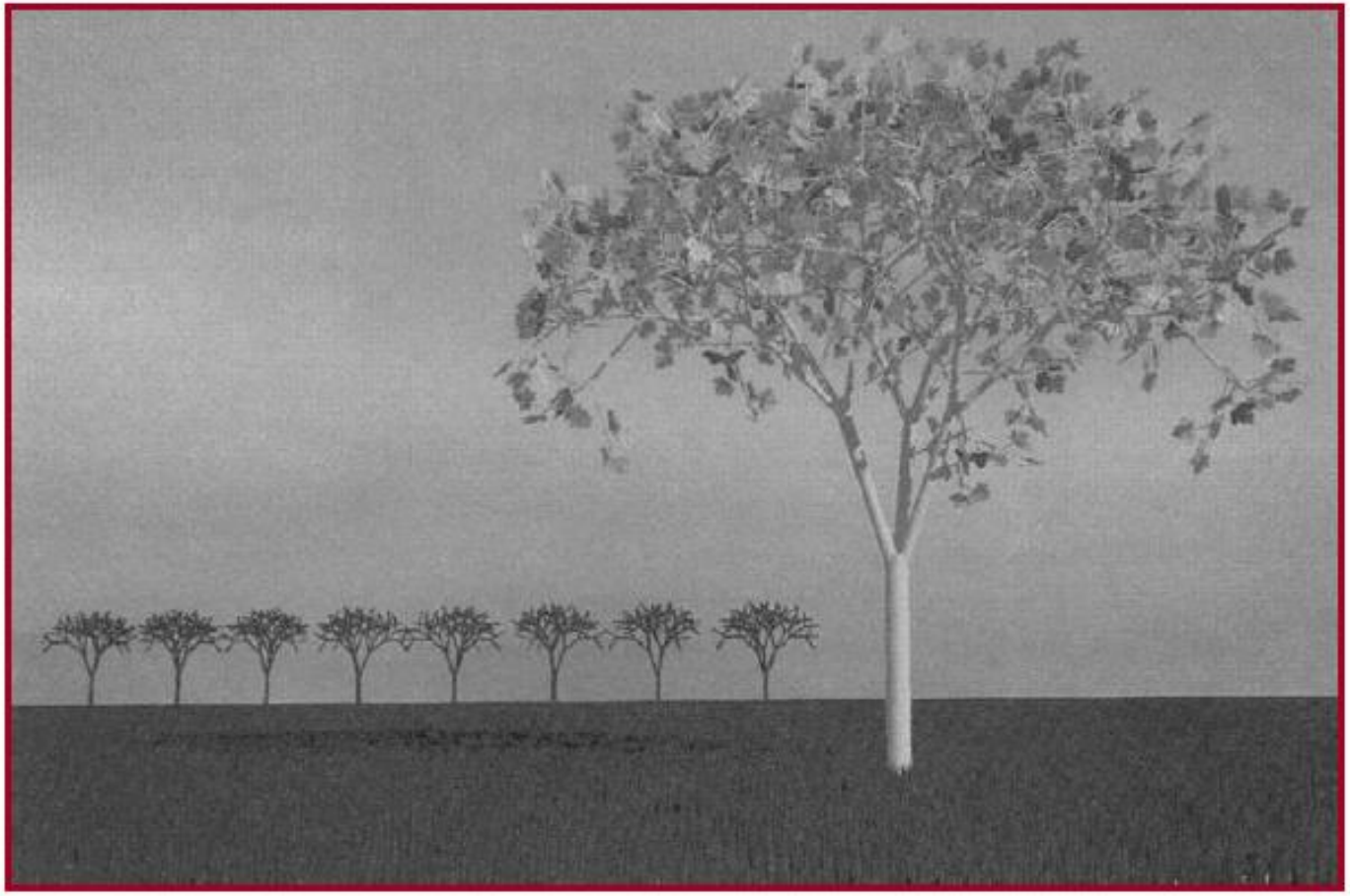
Statistical Fractal Generation

- Useful for creating 3D plants

H&B Figure 10.82

Statistical Fractal Generation

- Useful for creating 3D plants



H&B Figure 10.79

Procedural Modeling

L-Systems

Procedural Terrain

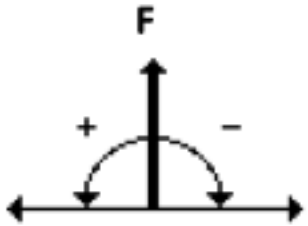
Procedural Behavior

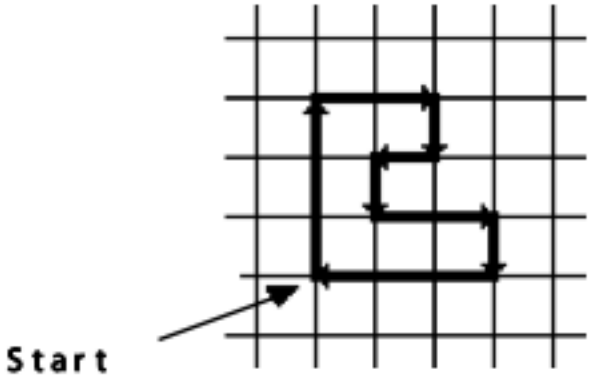
L-Systems (Background)

- Developed by Aristid Lindenmayer to model the development of plants
- Based on grammars
  - based on parallel string-rewriting rules
- Excellent for modeling organic objects (plants) and fractals
- Recent applications
  - Cities, feathers, etc.

L-Systems (Basic Example)

- **Turtle Commands:**
  - **F_x: move forward one step, drawing a line**
  - **f_x: move forward one step, without drawing a line**
  - **$+_x$: turn left by angle ∂**
  - **$-_x$: turn right by angle ∂**

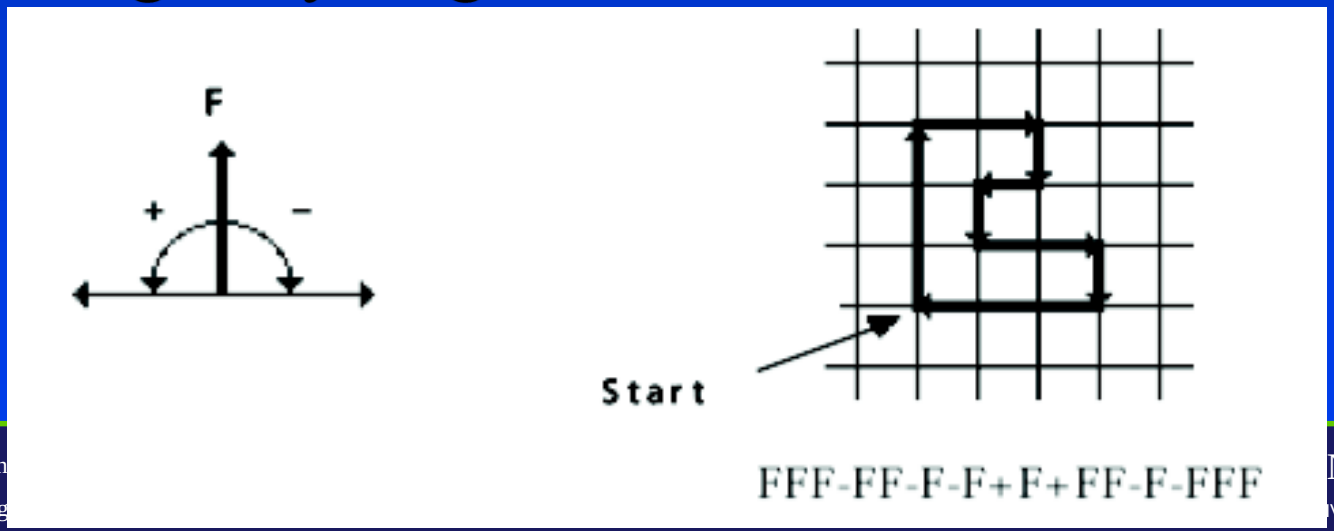




FFF-FF-F-F+F+FF-F-FFF

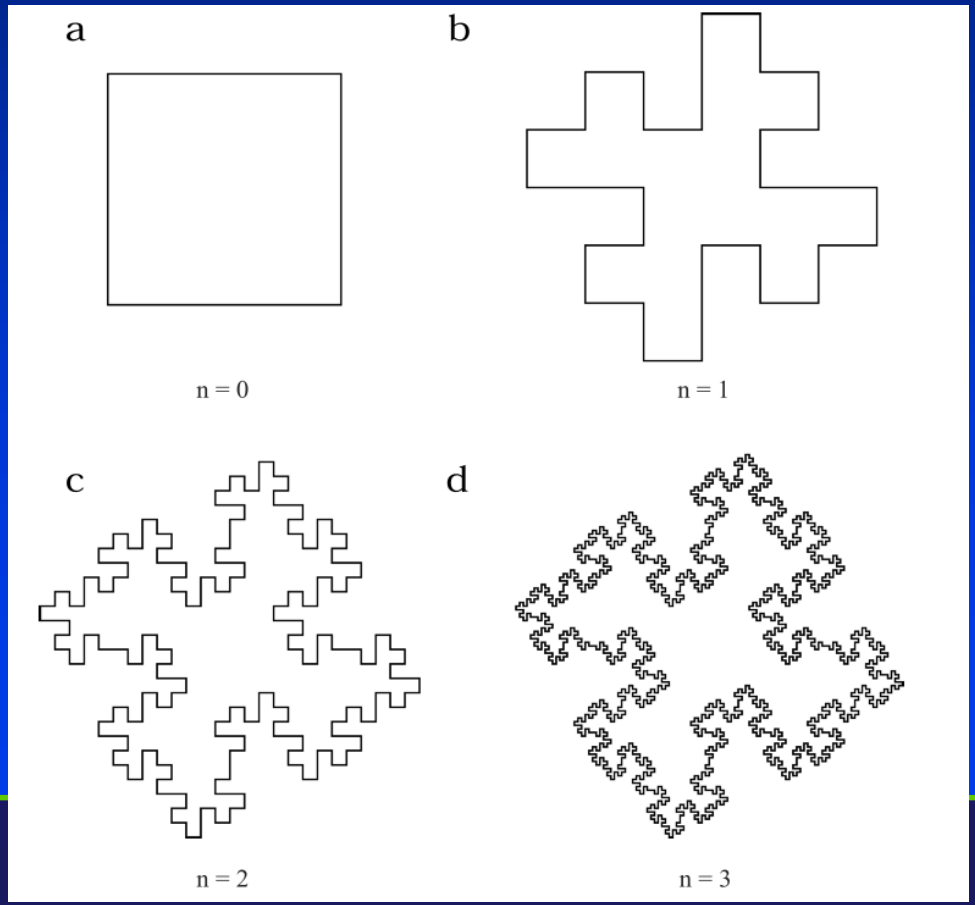
Turtle Commands

- F_x: move forward one step, drawing a line
- f_x: move forward one step, without drawing a line
- $+_x$: turn left by angle ∂
- $-_x$: turn right by angle ∂



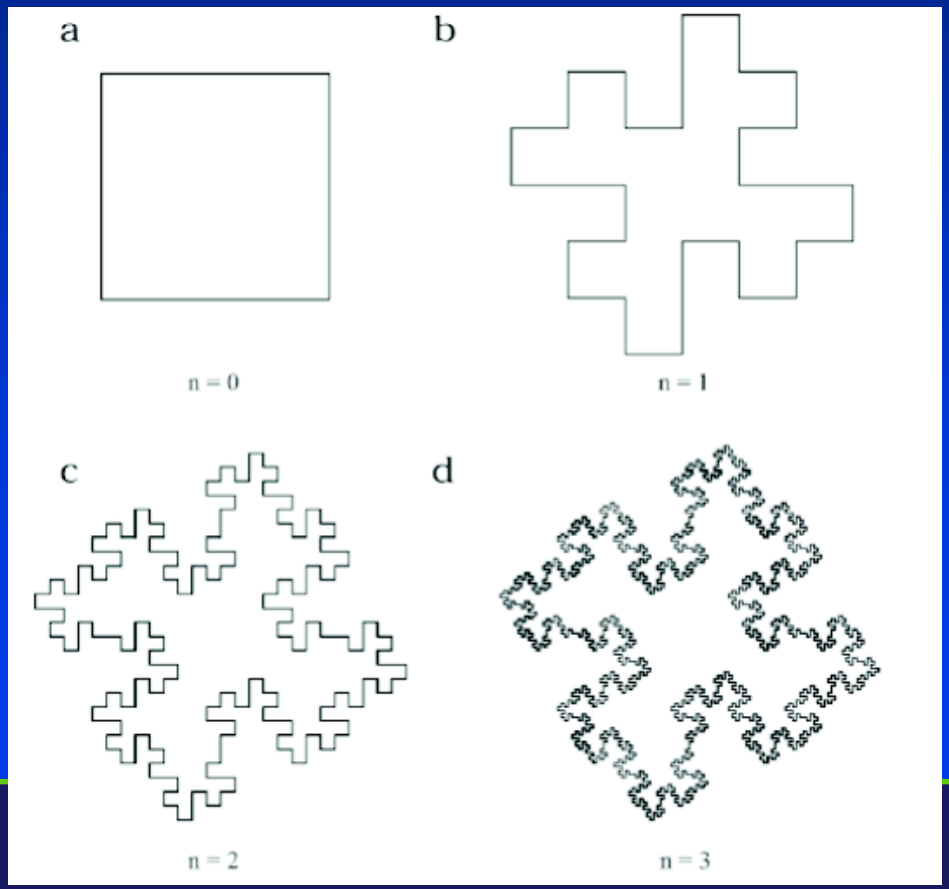
L-Systems (Koch Snowflake)

- **Axiom: F-F-F-F ∂:90 degrees**
- **F➔F-F+F+FF-F-F+F**



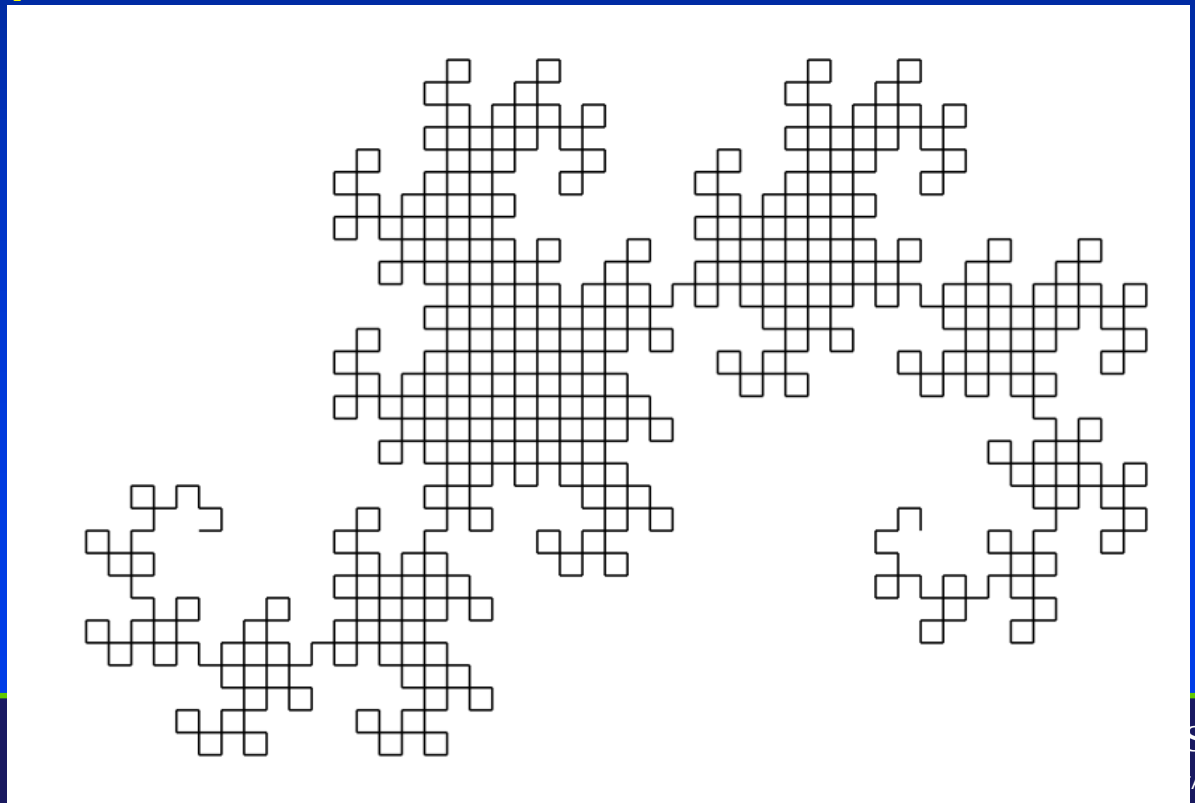
L-Systems Example: Koch Snowflake

- Axiom: F-F-F-F ∂ :90 degrees
- F -> F-F+F+FF-F-F+F



L-Systems (Dragon Curve)

- **Axiom:F_l ∂:90 degrees n:10 iterations**
- **$F_l \rightarrow F_l+F_r+$**
- **$F_r \rightarrow F_l-F_r-$**



L-Systems Grammar (Concepts)

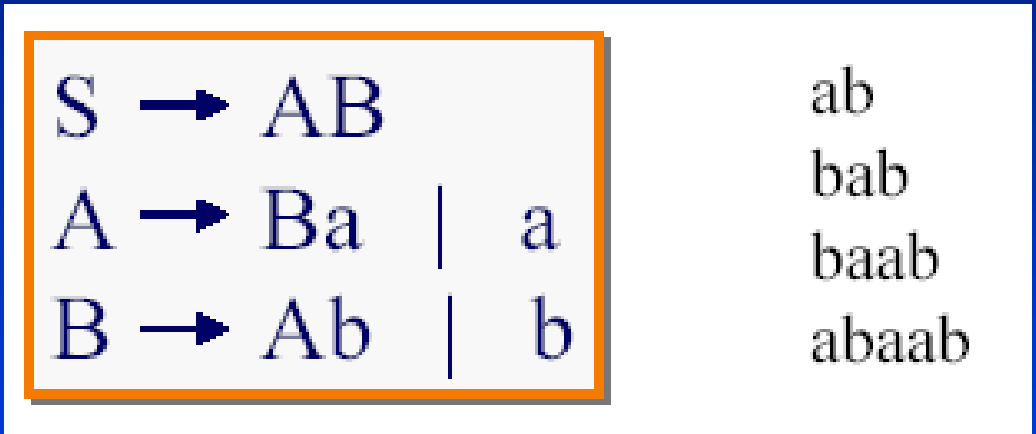
- Begin with a set of "productions" (replacement rules) and a "seed" axiom
- In parallel, all matching productions are replaced with their right-hand sides
- Example:
  - Rules:
    - B -> ACA
    - A -> B
  - Axiom: AA
  - Sequence: AA, BB, ACAACA, BCBBCB, etc.
- Strings are converted to graphics representations via interpretation as turtle graphics commands

Plants

- Complex systems
- Often, have a well defined structure
  - Trunk
  - Big branches
  - Little branches
  - Leaves
- High degree of "recursiveness"
  - Grammars/compilers are good with this.

Grammar-based Models

- Generate description of geometric model by applying production rules



Grammar-based Models

- Useful for modeling plants

  $F \rightarrow F[RF]F[LF]F,$

  where F: forward

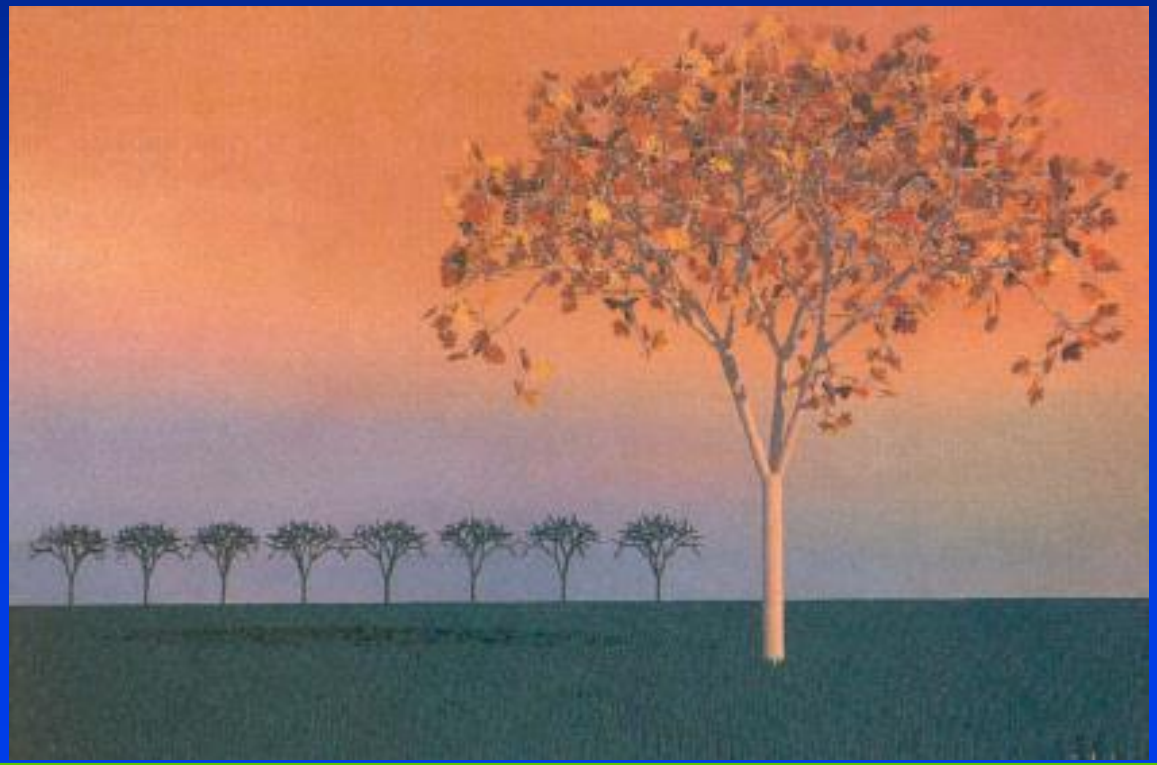
  R: turn right

  L: turn left.

Grammar-based models

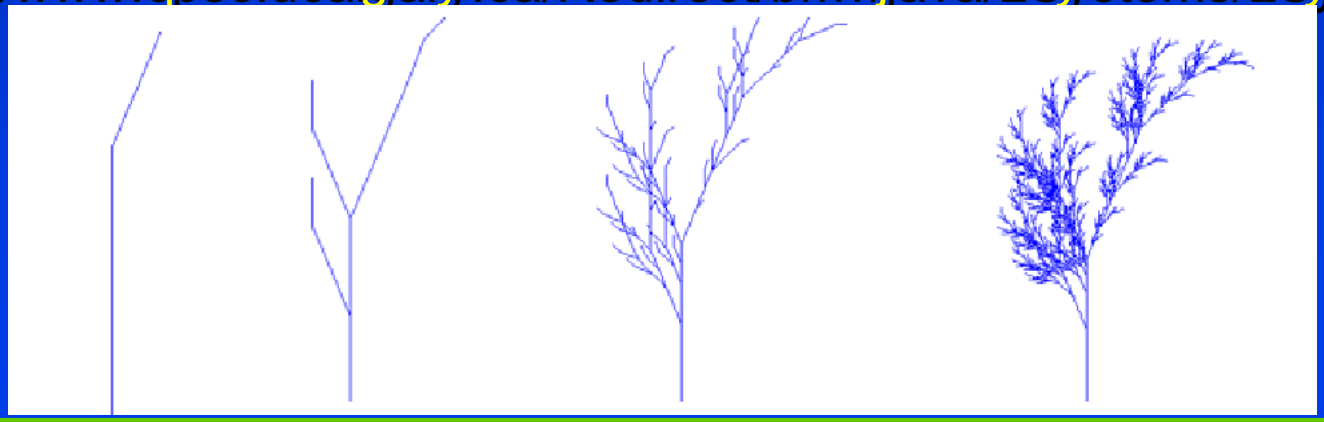
- Apply the rule randomly to occurrences of F.

  $F \rightarrow F^*[RF]F^*[LF]F^*$.

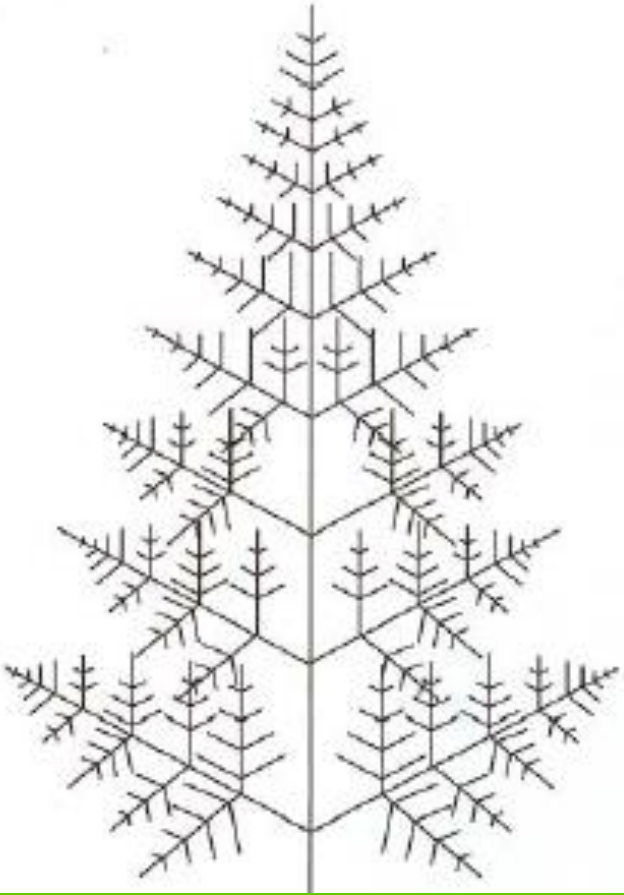


Grammar-based Model: L-systems

- Grammar-based fractal-like models
- Describe an object by a string of symbols and provide a set of production rules
- Incorporate notions such as branching, pruning, …
- Can also vary objects by randomly applying rules
- Demo: http://www.cpsc.ucalgary.ca/Redirect/bmv/java/LSystems/LSys.html



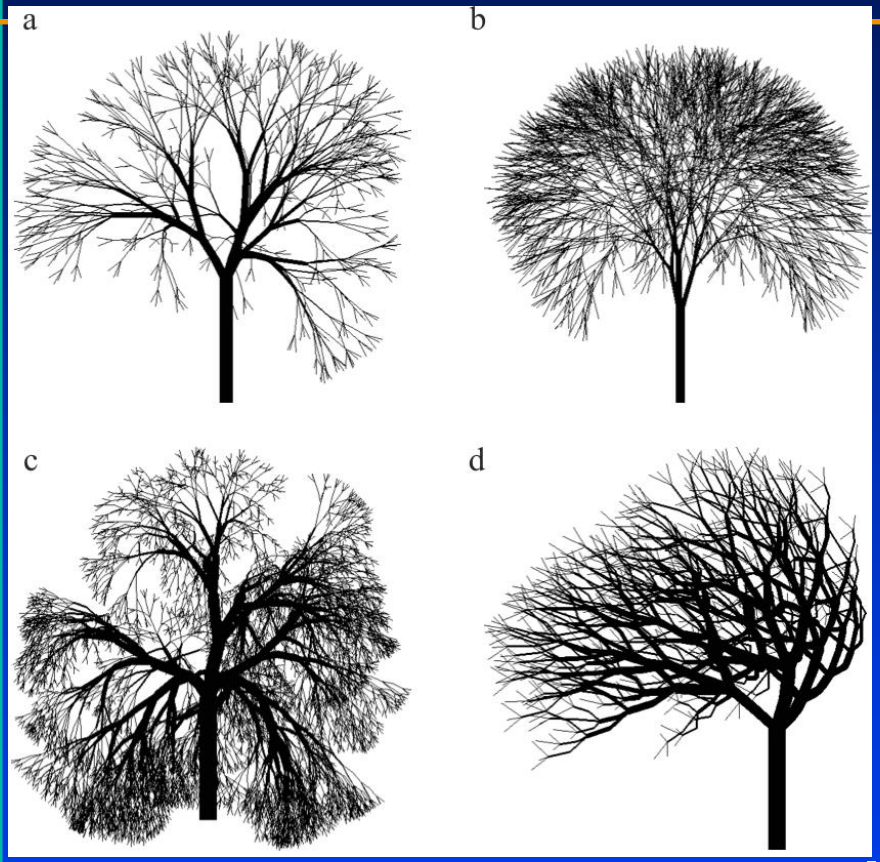
Grammar-based Models



L-Systems Grammar: Extensions

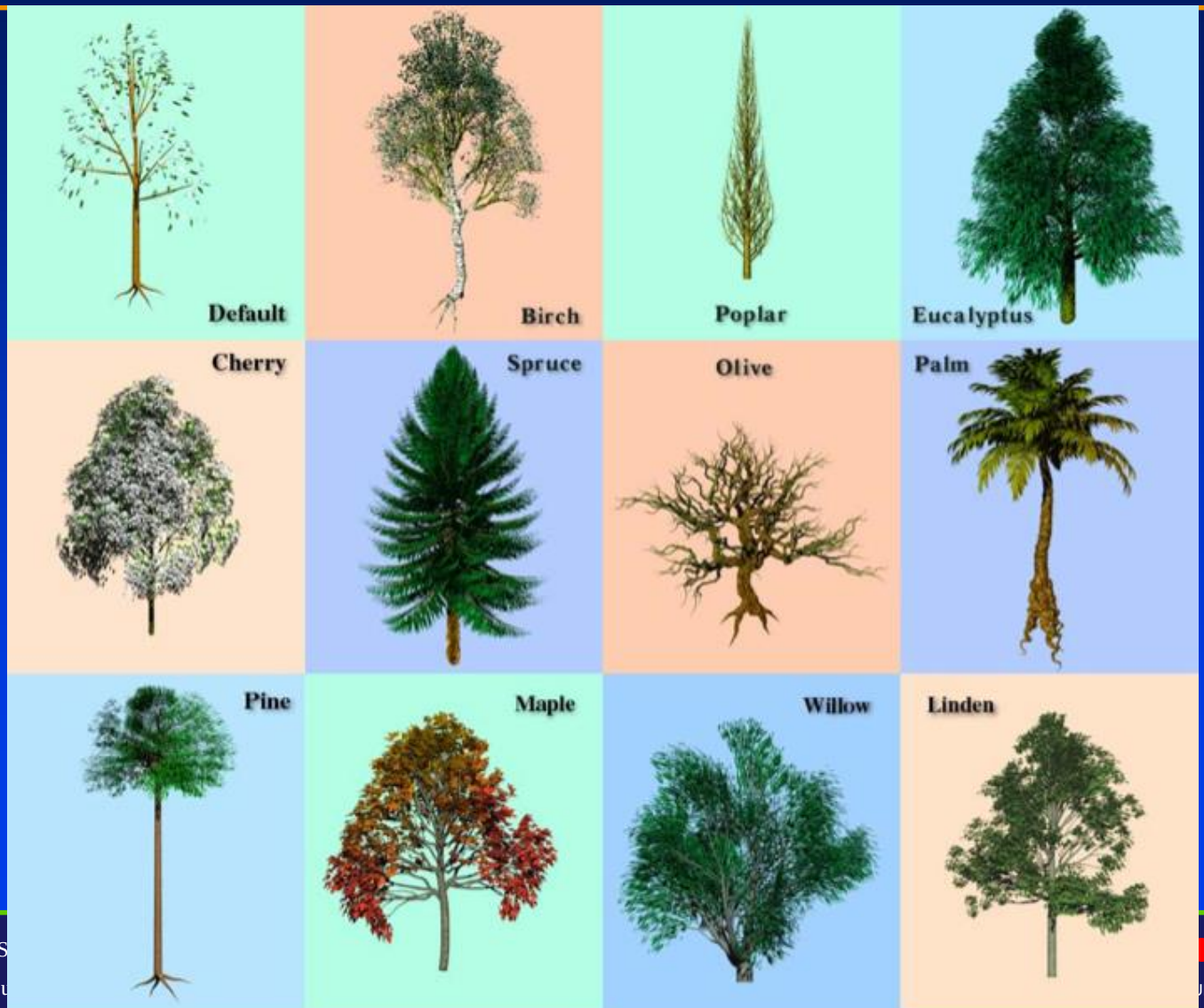
- Basic L-Systems have inspired a large number of variations
- Context sensitive: productions look at neighboring symbols
- Bracketed: save/restore state (for branches)
- Stochastic: choose one of n matching productions randomly
- Parametric: variables can be passed between productions

L-Systems For Plants



- **L-Systems can capture a large array of plant species**
- **Designing rules for a specific species can be challenging**

PovTree


Default
Birch
Poplar
Eucalyptus
Cherry
Spruce
Olive
Palm
Pine
Maple
Willow
Linden

- **Fast procedural foliage is important for real-time applications**

Procedural Modeling

- Sweeps
- Fractals
- Grammars

Grammars

- Generate description of geometric model by applying production rules

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow Ba \mid a \\
B &\rightarrow Ab \mid b
\end{aligned}
$$

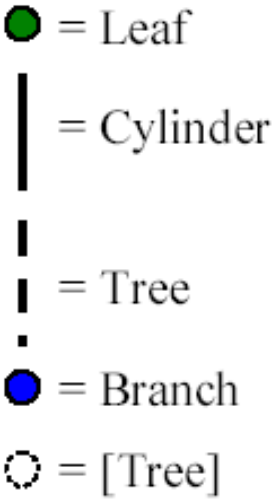
AB
BaB
BaAb
AbaAb
.
.
.

Grammars

- Useful for creating plants

T

Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]

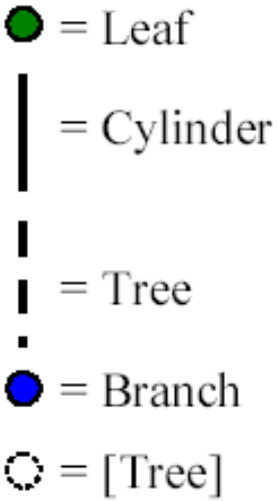


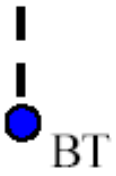
T

Grammars

- Useful for creating plants

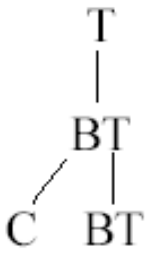
Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]



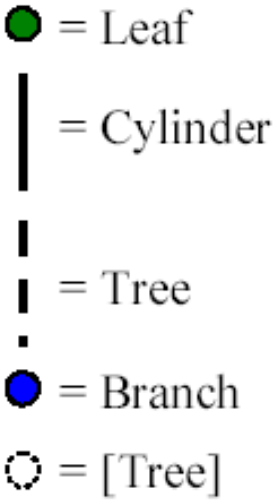


Grammars

- Useful for creating plants

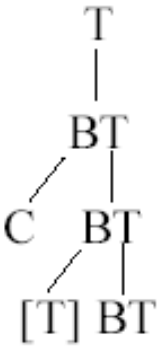


Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]

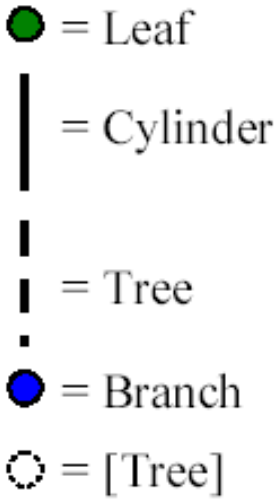


Grammars

- Useful for creating plants



Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]

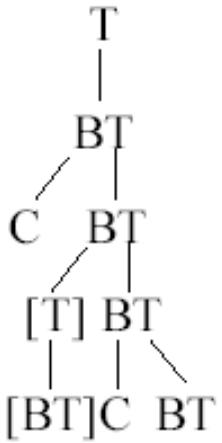


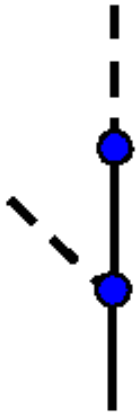
C[T]BT

Grammars

- Useful for creating plants

Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]





C[BT]CBT

= Leaf

= Cylinder

= Tree

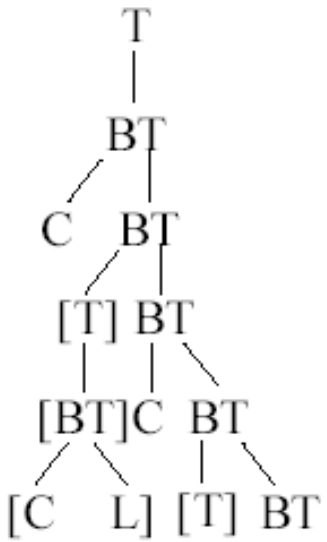
= Branch

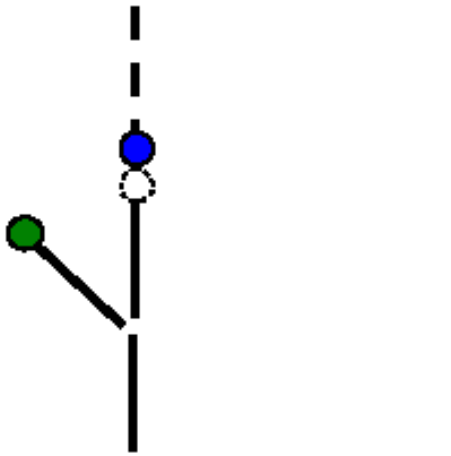
= [Tree]

Grammars

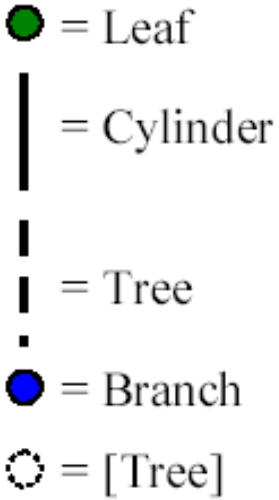
- Useful for creating plants

Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]





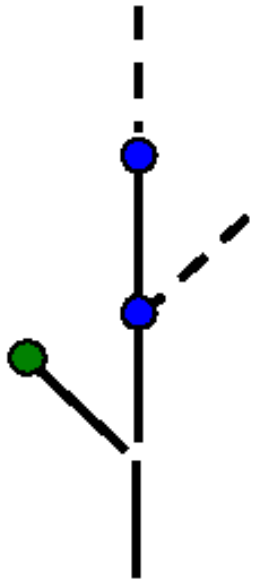
C[CL]C[T]BT



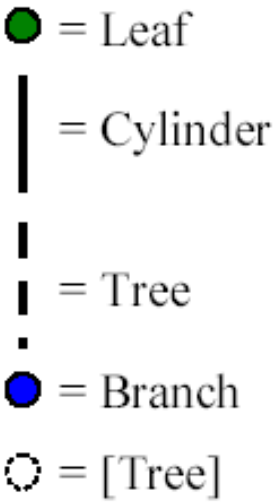
Grammars

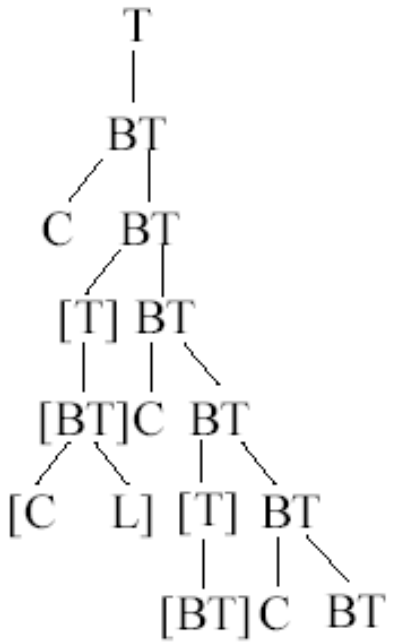
- Useful for creating plants

Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]



C[CL]C[BT]CBT

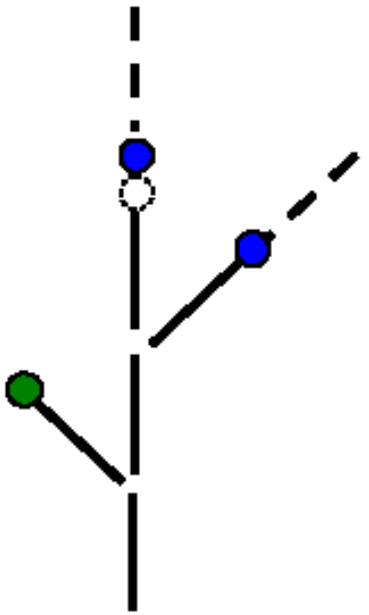




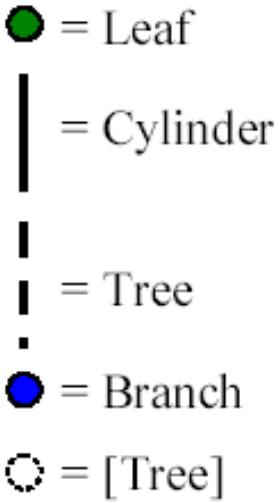
Grammars

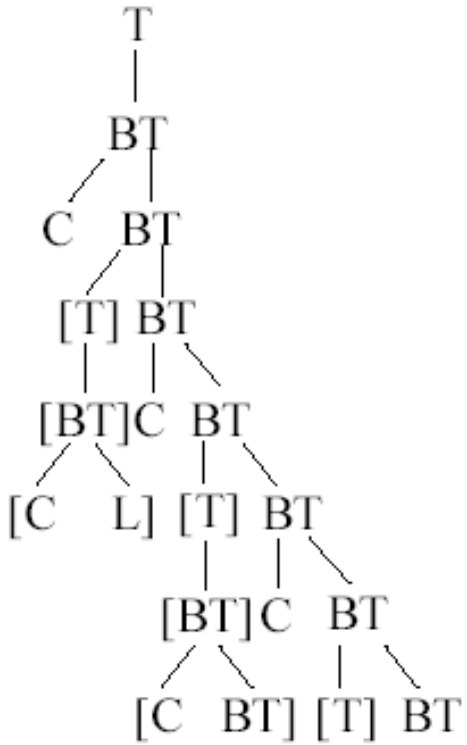
- Useful for creating plants

Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]



C[CL]C[CBT]C[T]BT

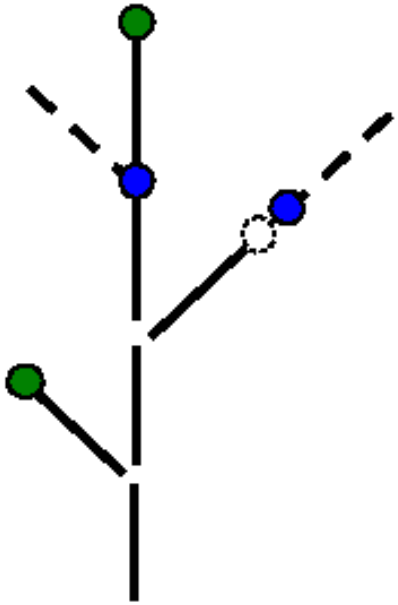




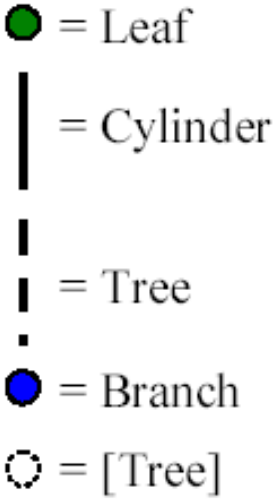
Grammars

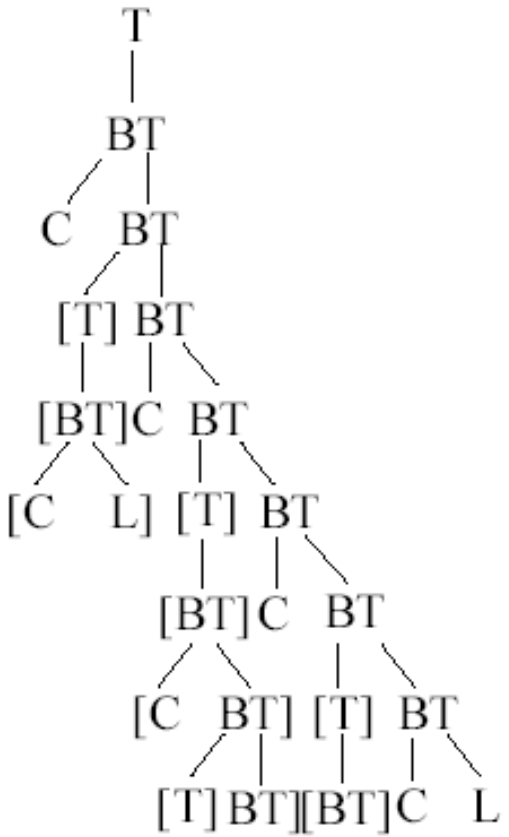
- Useful for creating plants

Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]



C[CL]C[C[T]BT]C[BT]CL

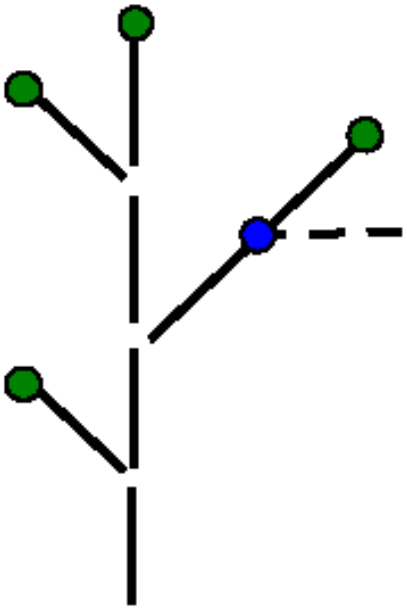




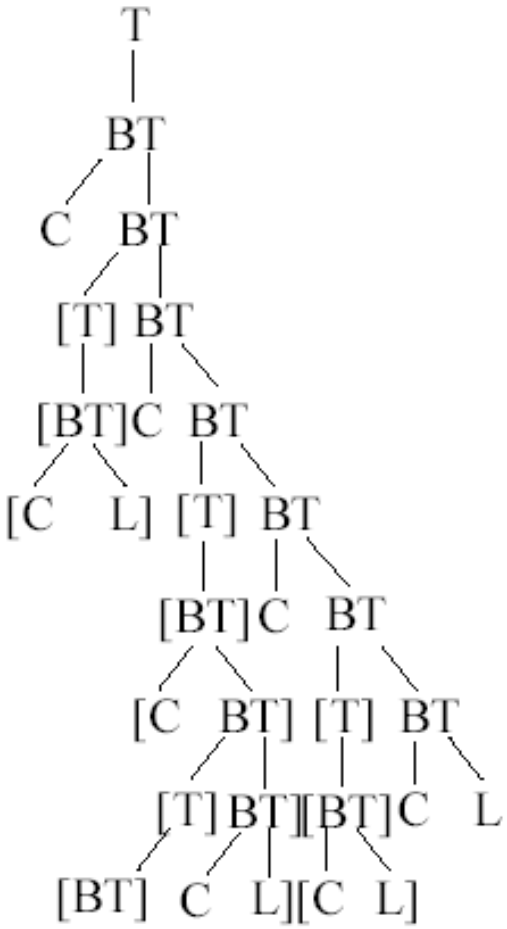
Grammars

- Useful for creating plants

Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]



= Leaf
= Cylinder
= Tree
= Branch
= [Tree]

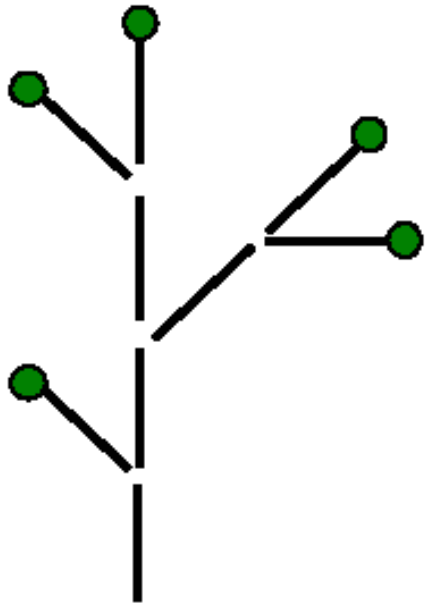


C[CL]C[C[BT]CL]C[CL]CL

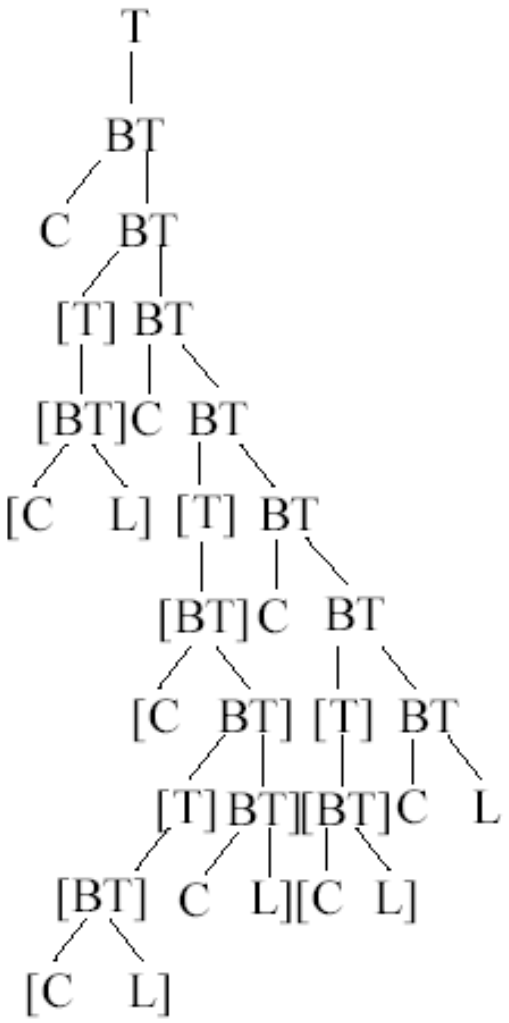
Grammars

- Useful for creating plants

Start → Tree
Tree → Branch Tree | leaf
Branch → cylinder | [Tree]



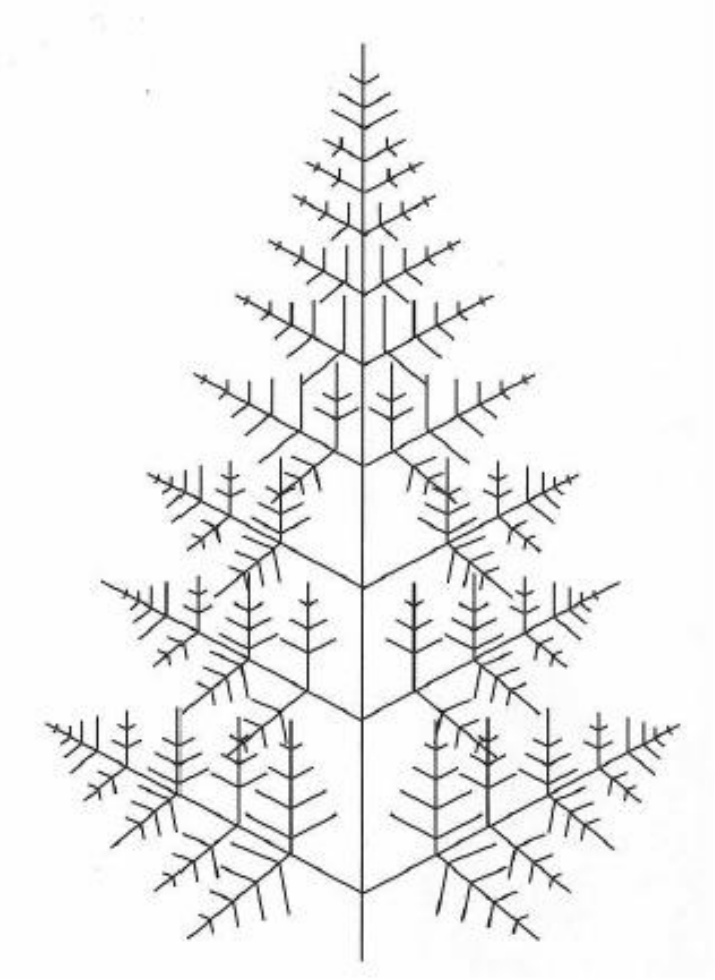
= Leaf
= Cylinder
= Tree
= Branch
= [Tree]

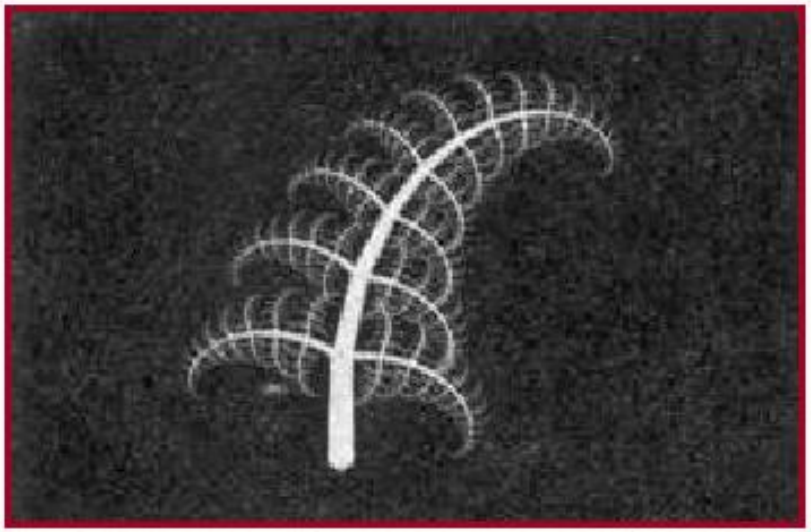


C[CL]C[C[CL]CL]C[CL]CL

Grammars

- Useful for creating plants





H&B Figure 10.77

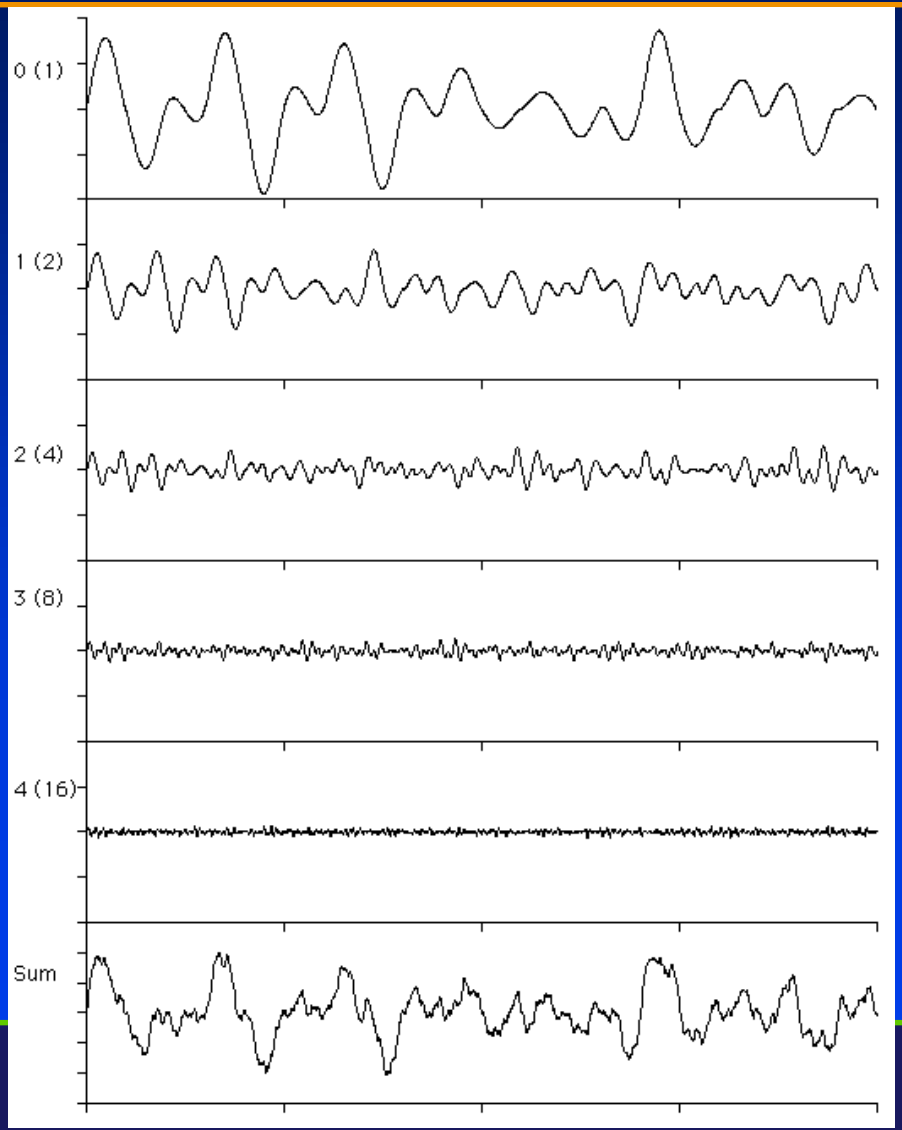
Trees

- Trees are a classic example of complex natural objects that can be procedurally modeled
- There have been numerous research papers published on various aspects of botanical modeling
- One recent paper focused on creating the detailed sub-millimeter scale vein patterns seen on leaf surfaces
- By varying a number of key parameters, one can model a wide variety of plants and trees to any level of detail desired
- Even with about 10 parameters, one can model a wide variety of overall plant shapes, but real plant modeling systems may allow hundreds of parameters as well as the inclusion of custom geometric data to define leaf shapes or branch cross sections….

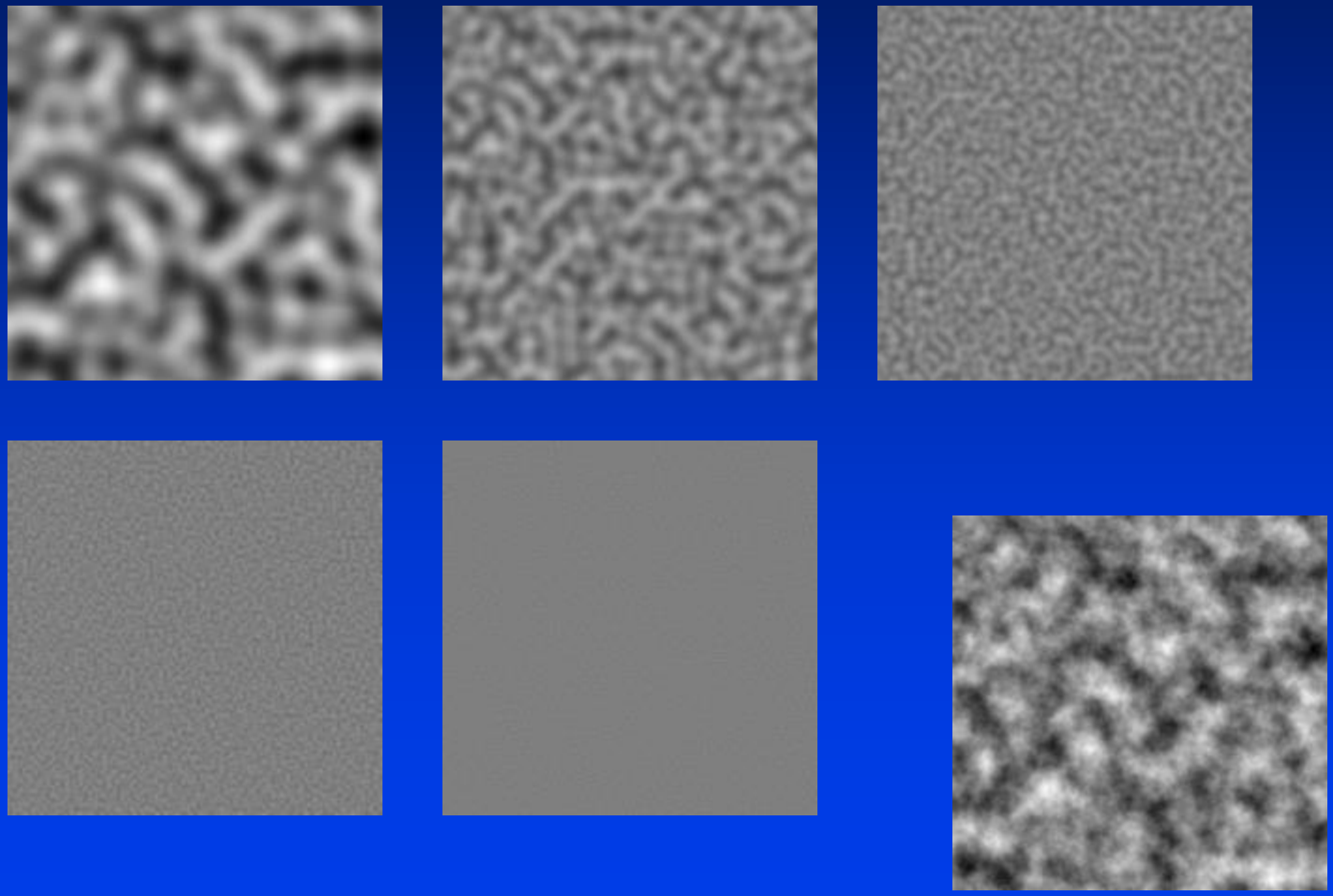
Procedural Terrain: Perlin Noise

- Noise Functions
  - Seeded pseudo-random number generator
  - Over R^n
  - Approximation to Gaussian filtered noise
  - Implemented as a pseudo-random spline
  - The trick is to make it fast

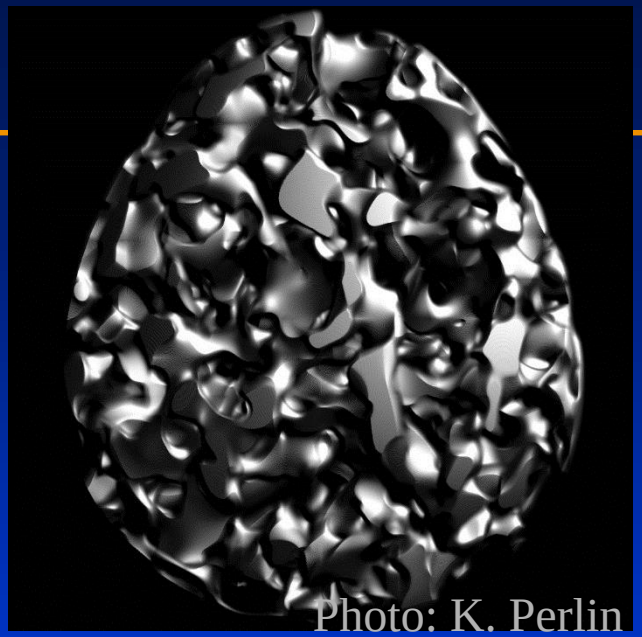
Perlin Noises in 1-D



Perlin Noises in 2-D

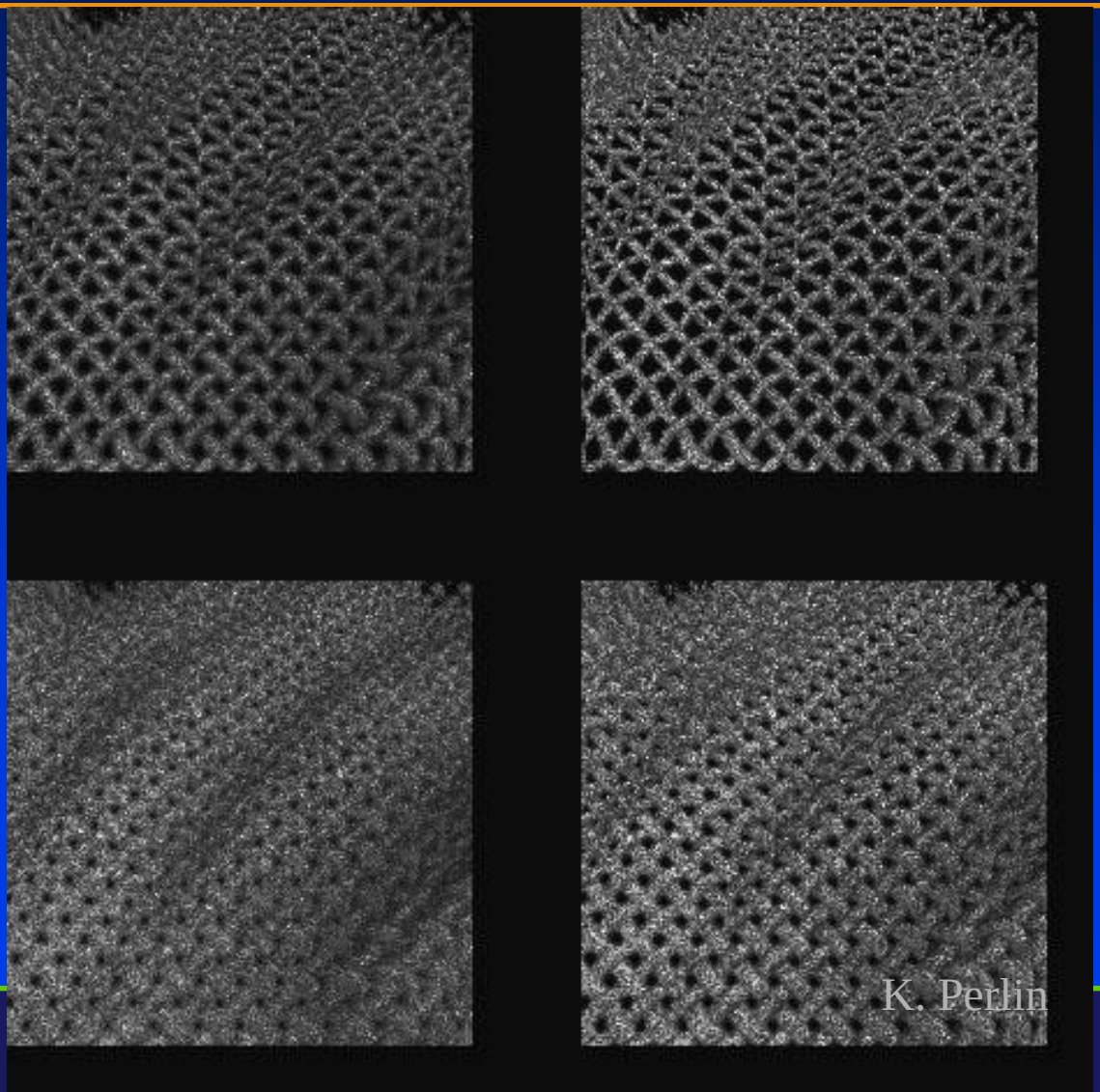


Hypertexture

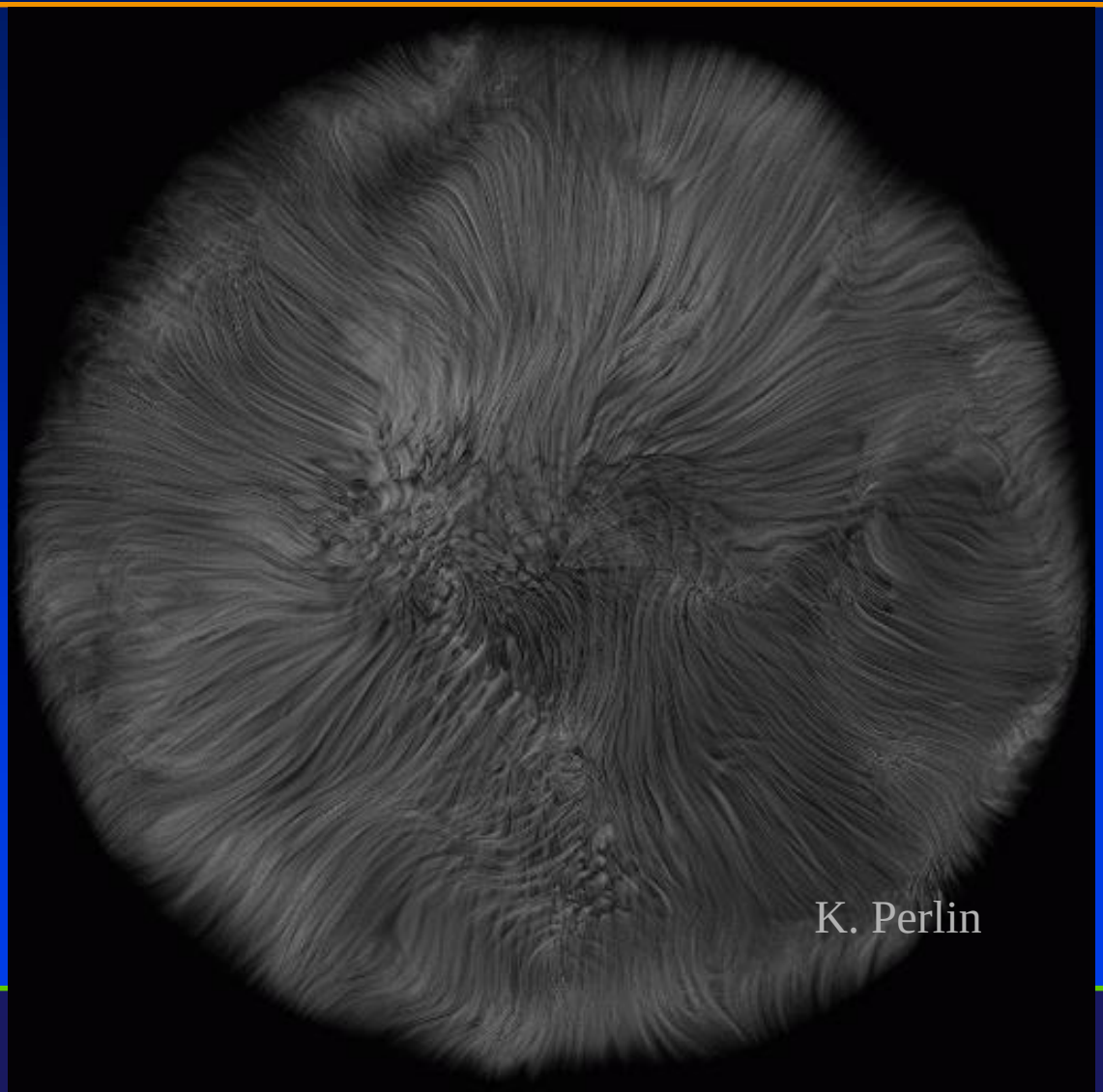


- **Implicit procedural model**
- **Treat the isosurface of a function as the boundary of an object**
- **Above: fractal egg**

Hypertexture Example

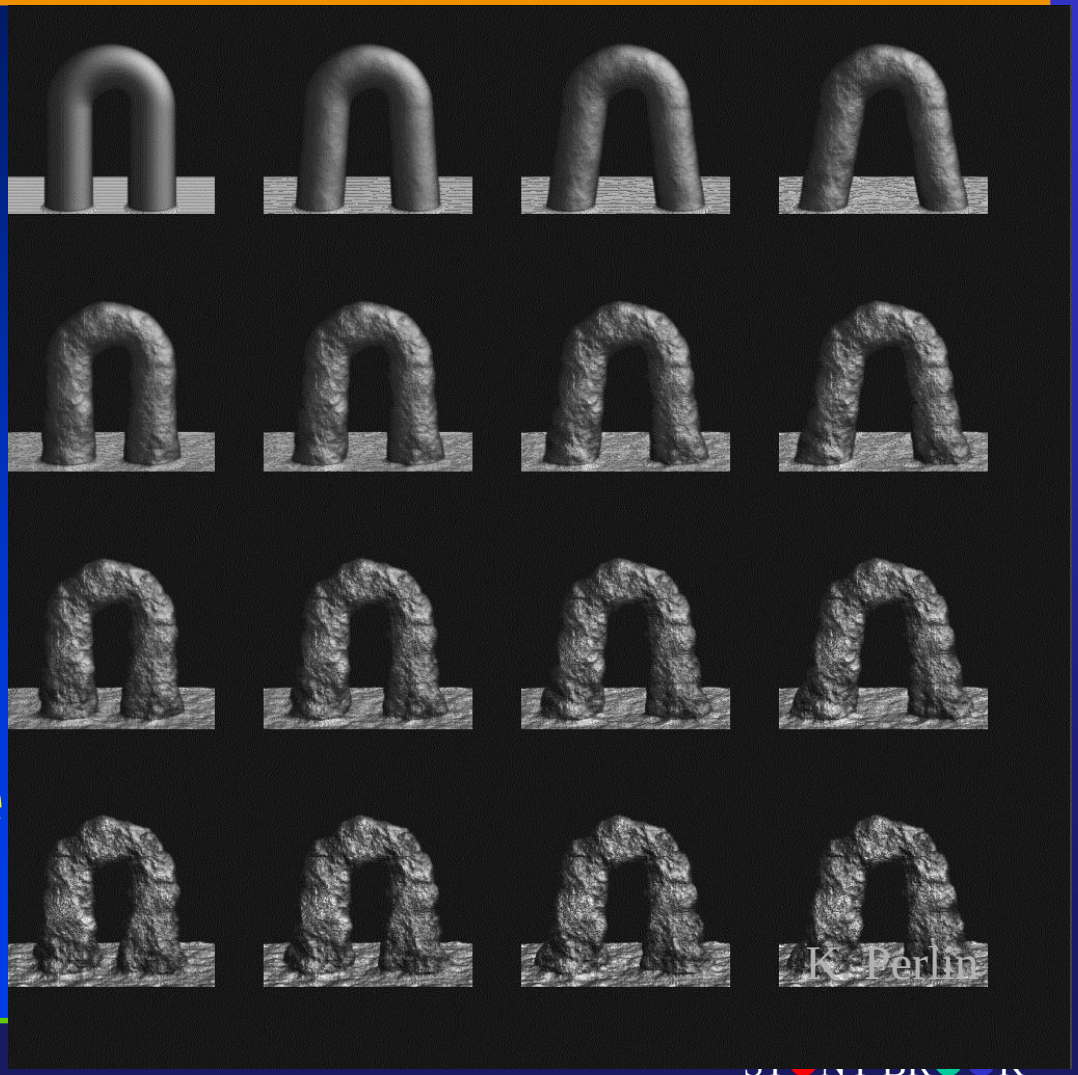


Hypertexture Example



Architexture

- **Sweep the path of a line drawing with a sphere**
- **Apply hypertexture to resulting shape**



Roads

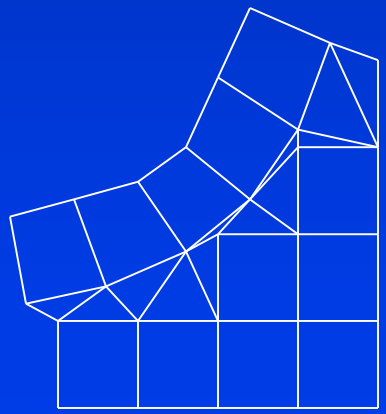
- Roads can be modeled as cross sections that get path extruded along some curve
- The cross section can include lanes, curbs, sidewalks, and center islands
- Intersections require special handling, but can still be generated using a set of procedural techniques
- As with using DEMs for creating height fields, it is possible to find road map data online that contains maps of many key metropolitan areas. Often, the road map data is defined as a graph of connected lines with additional information for each line segment such as the number of lanes and the address range

Roads

- Roads can be placed on height fields by placing the control points of the road curves on the height field
- This will still require some local flattening for the road and the area to the side of the road
- This can be achieved by essentially rendering road triangles onto the height field, where a low detail road is extruded and ‘rendered’ into the cells of the height field to set their heights. Sides of roads can be blended using techniques similar to alpha blending
- These operations can be used to modify the existing shape the height field in a very similar way to how real roads are constructed

Roads

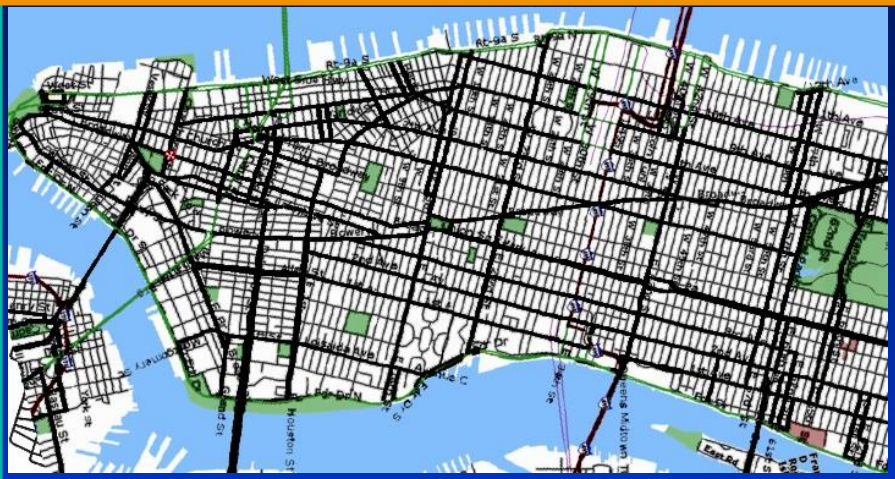
- Ideally, the road surface would be an extrusion and the open terrain would be a height field
- They can be sewn together into a single triangle mesh in a similar way to how trim curves are used in patch tessellation



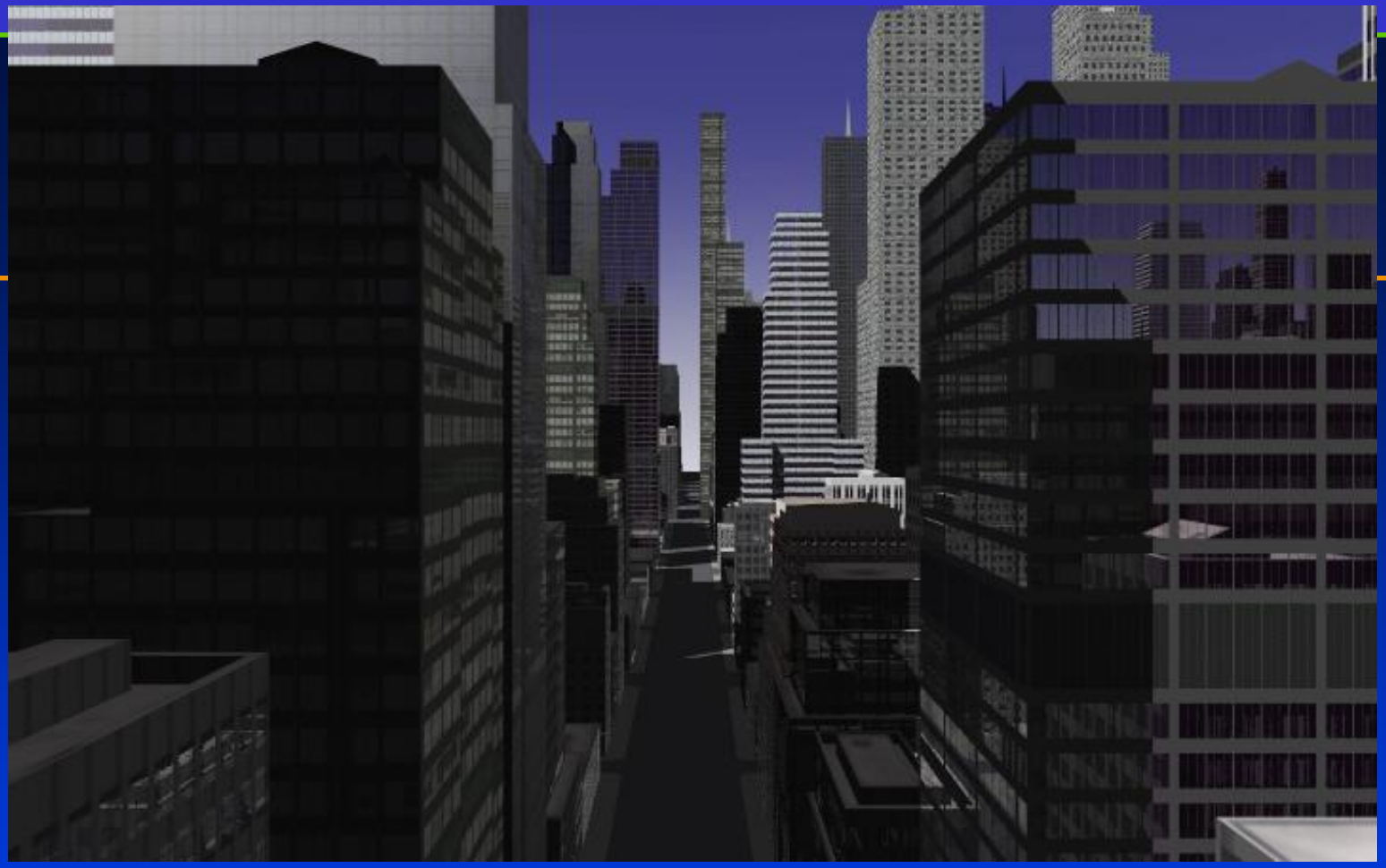
Buildings and Cities

- Buildings and other man-made structures can also be procedurally modeled
- In addition to the overall shapes of buildings, there have been papers for details such as exact brick placement including a variety of patterns
- There have also been research papers on automatically generating city road map layouts based on terrain height fields
- From the road maps, city blocks are then subdivided into lots, which have procedural buildings placed on them
- Details like street lights, trees, etc. can be placed along the roads
- In this way, entire cities can be build automatically
- Cities (and other complex models) can either be generated completely randomly, or as a mix of random and non-random processes
- Additional data exists for cities that describe locations and overall outlines of buildings, placement of power & telephone lines, train tracks, and other data

L-Systems for Cities [Parish01]



- **Start with a single street**
- **Branch & extend w/ parametric L-System**
- **Parameters of the string are tweaked by goals/constraints**
- **Goals control street direction, spacing**
- **Constraints allow for parks, bridges, road loops**



- **Once we have streets, we can form buildings with another L-System**
- **Building shapes are represented as CSG operations on simple shapes**

Proper Placement

- There have been various research papers that have addressed the issue of prop placement as a procedural modeling tool
- To place plants, for example, we do not just want to randomly scatter them around
- Models have been designed that take the shape of the terrain into consideration and determine plant locations based on properties such as wetness, light exposure, and other geographic properties
- In addition, simulations can be run that model the changes in the ecosystem over time and allow for different plant groups to spread about the terrain
- Man-made objects can also be automatically placed in terrains. Objects such as street lights, traffic lights, houses, street signs, and more can be placed automatically in a city based on the basic road map and terrain layouts

Using Noise in 3-D to Animate 2-D Flows

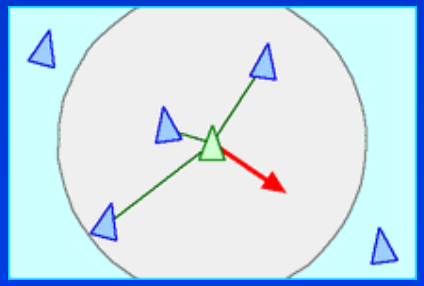
- Treating time as another spatial dimension
- Examples
  - Corona [K. Perlin]
    - http://www.noisemachine.com/talk1/imgs/flame500.html
  - Clouds [K. Perlin]
    - http://www.noisemachine.com/talk1/imgs/clouds500.html

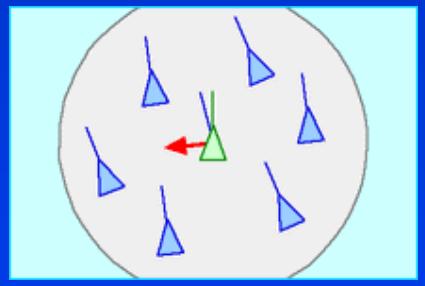
Procedural Animation

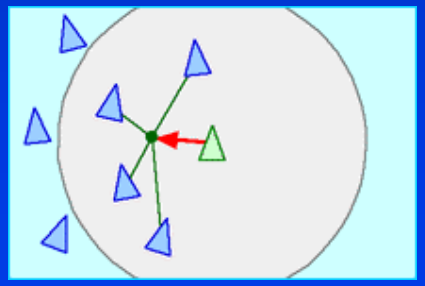
- Fluid simulation
- Particle systems
- Flocking/crowd simulations

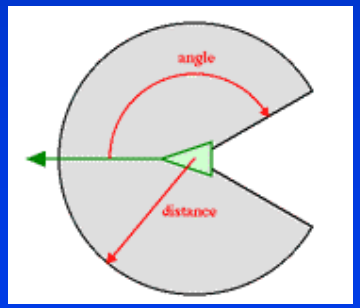
Procedural Flocking (Boids)

- Simulate the movement of a flock of birds in 3-space
- Separation: move to avoid crowding local neighbors
- Alignment: steer towards average heading of neighbors
- Cohesion: steer towards average position of neighbors
- Limited senses: only neighbors in forward-facing arc are observable



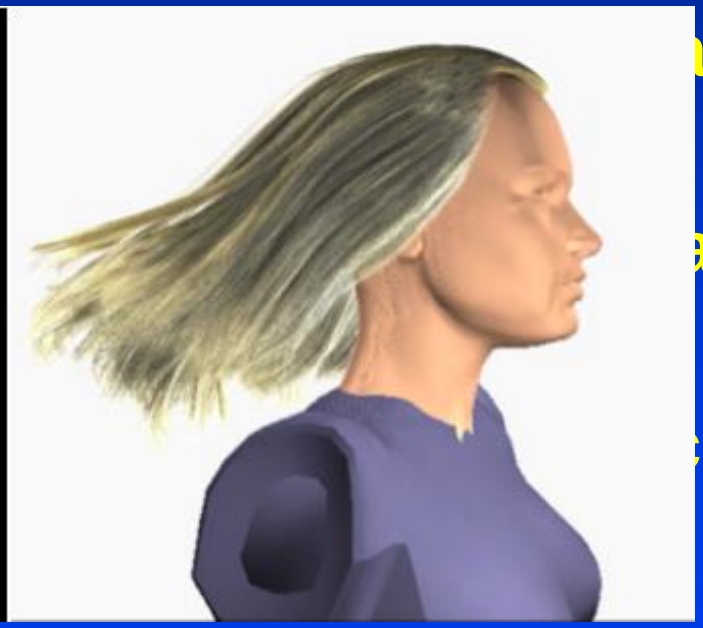






Procedural Hair [Chang02]

- **Generate a model with a few hundred guide hairs**
- **Each hair is a rigid chain w/ revolute joints**
- **Us irs to s**
- **Cre airs to s**
- **Int e ma**

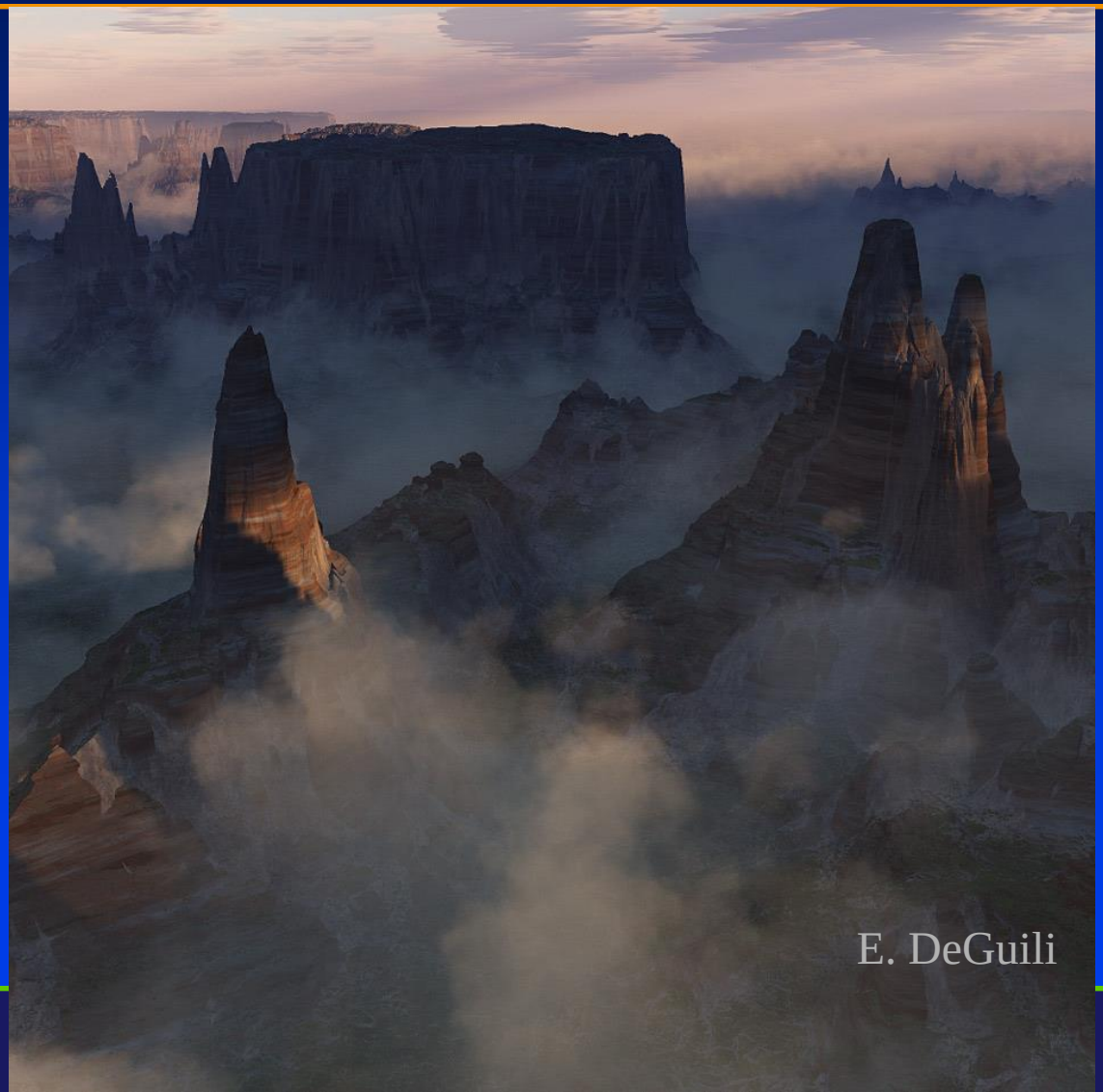


Procedural Hair (Examples)

Flow-Based Video Synthesis And Editing [Bhat04]

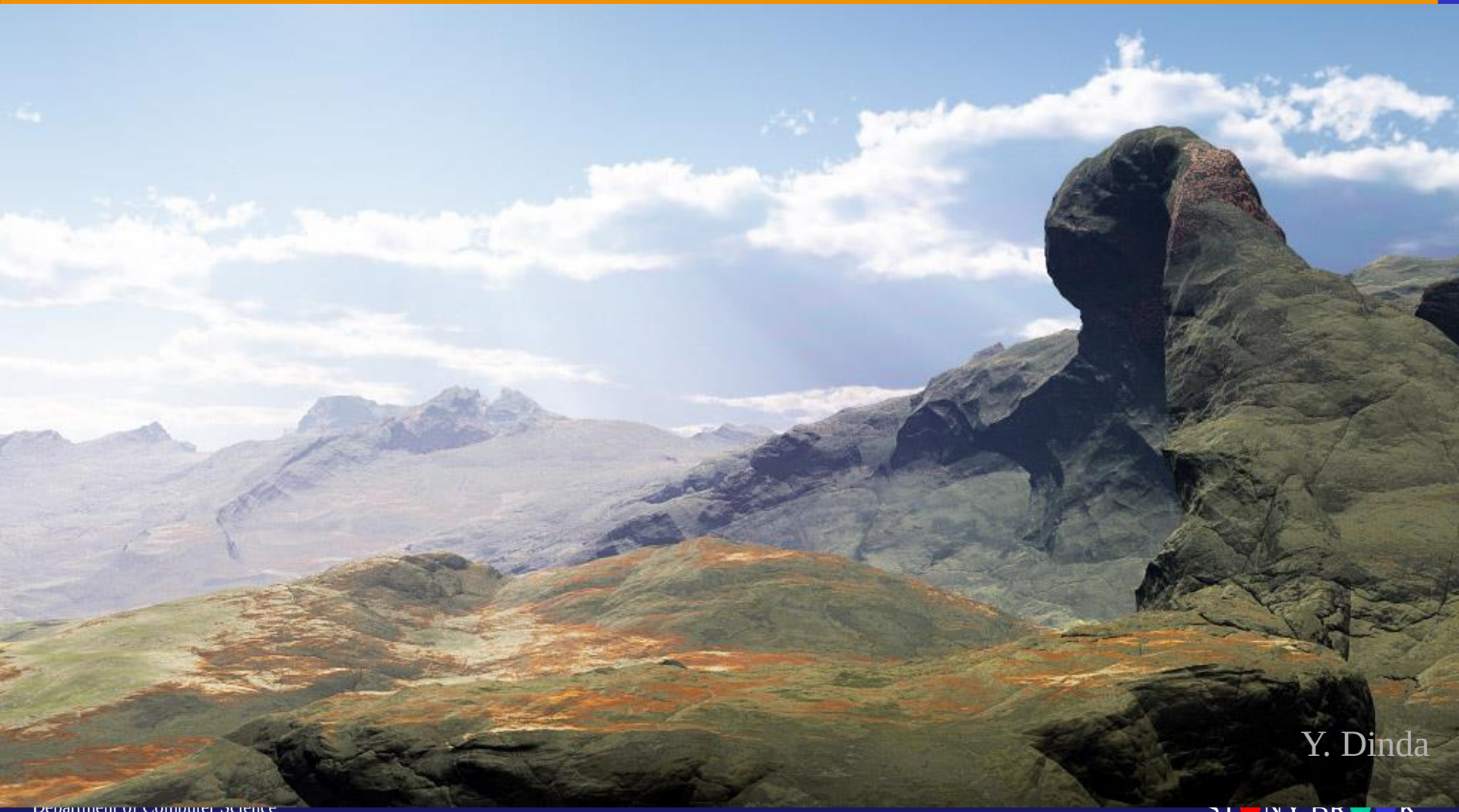
- **Allows animator to easily create loops and variants of flowing natural phenomena (water, smoke, etc)**
- **Artist draws a set of flow lines on the original image**
- **Algorithm computes textures for a particle system that uses these flow lines**
- **Sc**
  pr
- **Artist can then draw additional flow lines to create new variants**

Procedural Planets



Procedural Planets

Procedural Planets



F.K. Musgrave

Department of Computer Science
Center for Visual Computing
STONY BROOK
STATE UNIVERSITY OF NEW YORK

Gameplay Snapshots

- **Multiple sub-games of creature/civilization gameplay**
- **Editors for creatures, buildings, vehicles**
- **Procedural behavior, animation, and texturing (driven by player-created models)**

Procedural Modeling

Procedural Modeling: Summary

- Procedural techniques are very powerful
- Use with care
  - Physical validation is rare
- For some objects, procedure is the answer
  - Plants
- Can complement physics-based methods
  - Adding high frequency details
- General recommendation: add noise to your models to make them more “natural”

Cellular Automata

- Simple example - game of life
- The rules:
  - Binary state on a 2D grid: cell / no cell
  - If too few or too many cells surround a cell, it dies
  - If two cells surround empty space, a cell is born
- Very simple rules produce complex behavior
  - Stable patterns
  - Moving stable patterns
  - Oscillations

Complex Cellular Automata

- Can have more than one cell system interacting
- Rules can be arbitrary
  - Model the needed behavior
- The result is (usually) a distribution of some property
  - Binary result, need extra step
- The art is in creating the rules
  - Take from differential equations
  - Take from general intuition
    - Too many cell -> overcrowding -> dies

Lattice Boltzman Machine

- Extension of cellular automata
- Can simulate more natural phenomena besides clouds

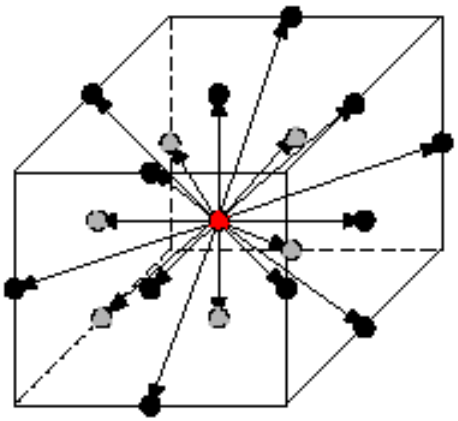


Figure 3: The D3Q19 lattice geometry

Summary

- Interactive modeling tools
  - CAD programs
  - Subdivision surface editors

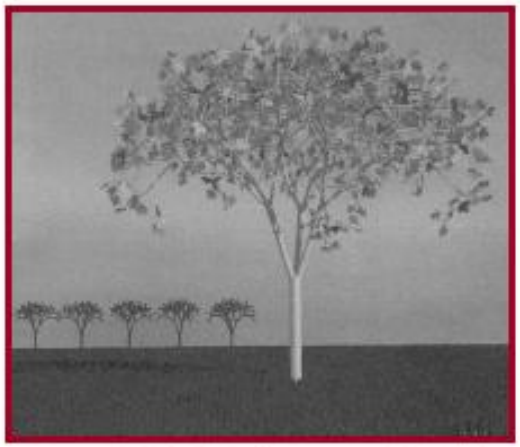
Constructing 3D models is hard!

- Scanning tools
  - CAT, MRI, Laser, magnetic, robotic arm, etc.
- Computer vision
  - Stereo, motion, etc.
- Procedural generation
  - Sweeps, fractals, grammars

Jurassic Park

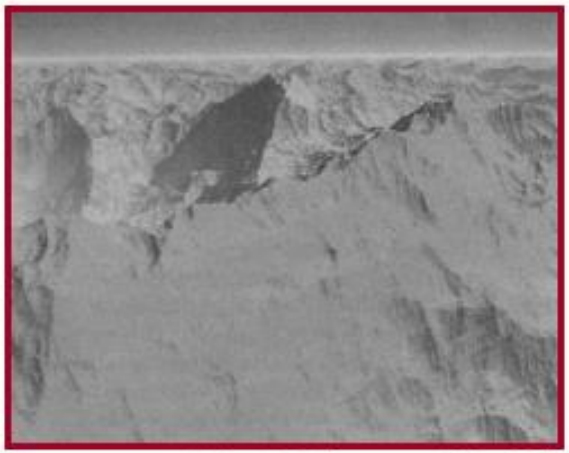
Modeling

- How do we …
  - Represent 3D objects in a computer?
  - Construct such representations quickly and/or automatically with a computer?
  - Manipulate 3D objects with a computer?



H&B Figure 10.79

Fowler



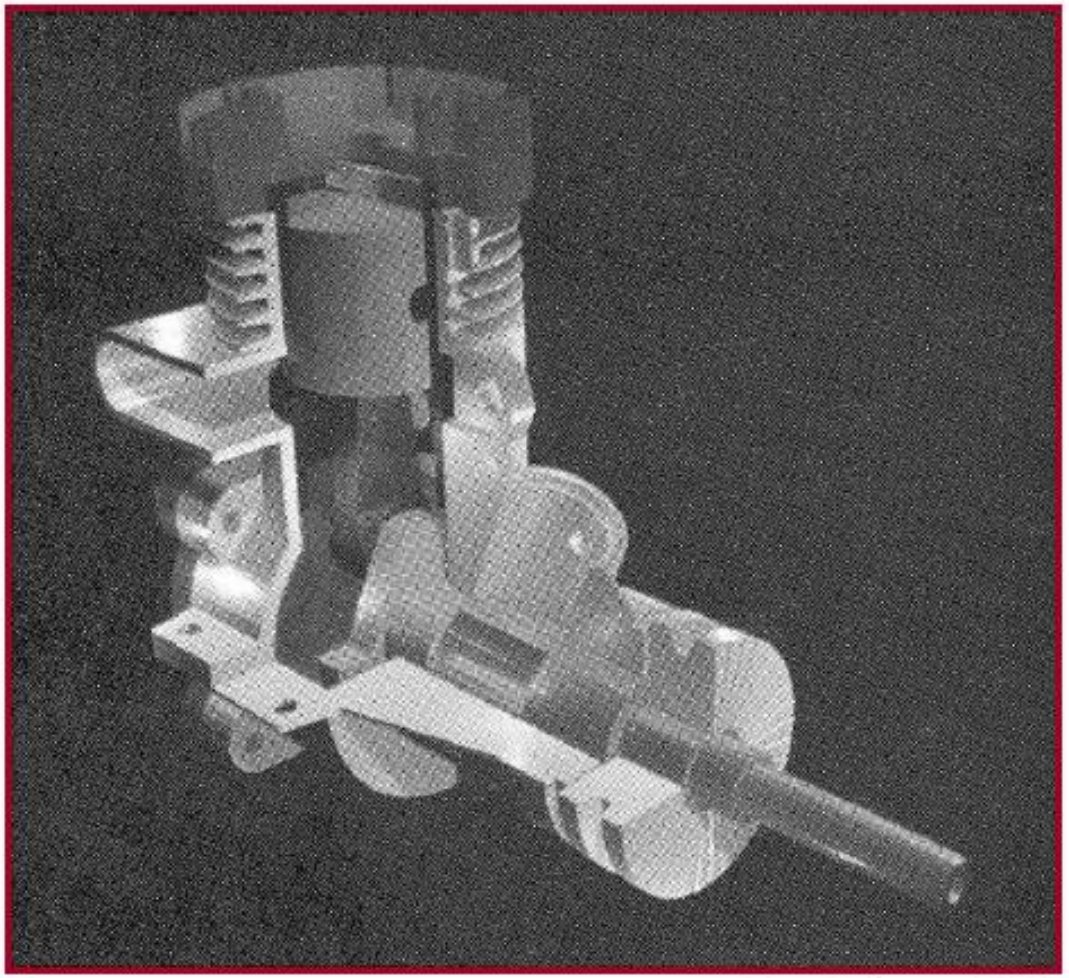
H&B Figure 10.83b

Model Construction

- Interactive modeling tools
  - CAD programs
  - Subdivision surface editors
- Scanning tools
  - CAT, MRI, laser, magnetic, robotic arm, etc.
- Computer vision
  - Stereo, motion, etc.
- Procedural generation
  - Sweeps, fractals, grammars

Interactive Modeling Tools

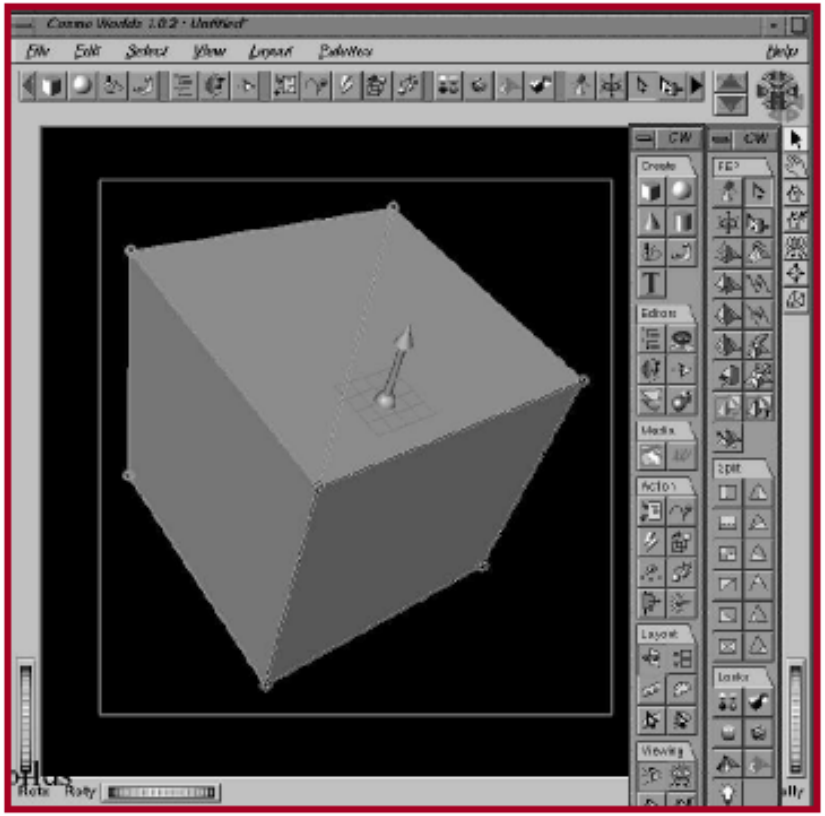
- Example: Mechanical CAD



H&B Figure 9.9

Interactive Modeling Tools

- User constructs objects with drawing program
  - Menu commands, direct manipulation, etc.
  - CSG, parametric surfaces, quadrics, etc.



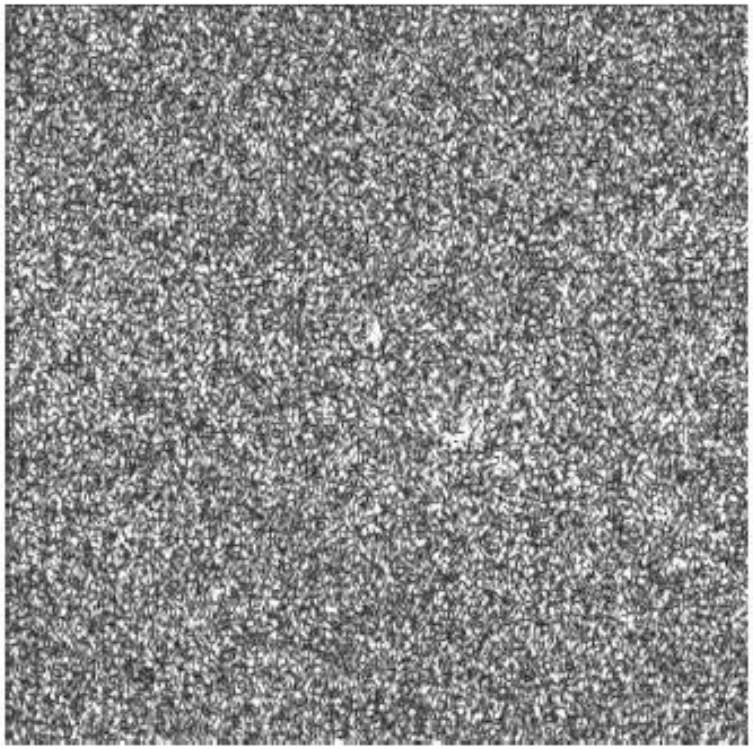
Cosmoworlds, SGI

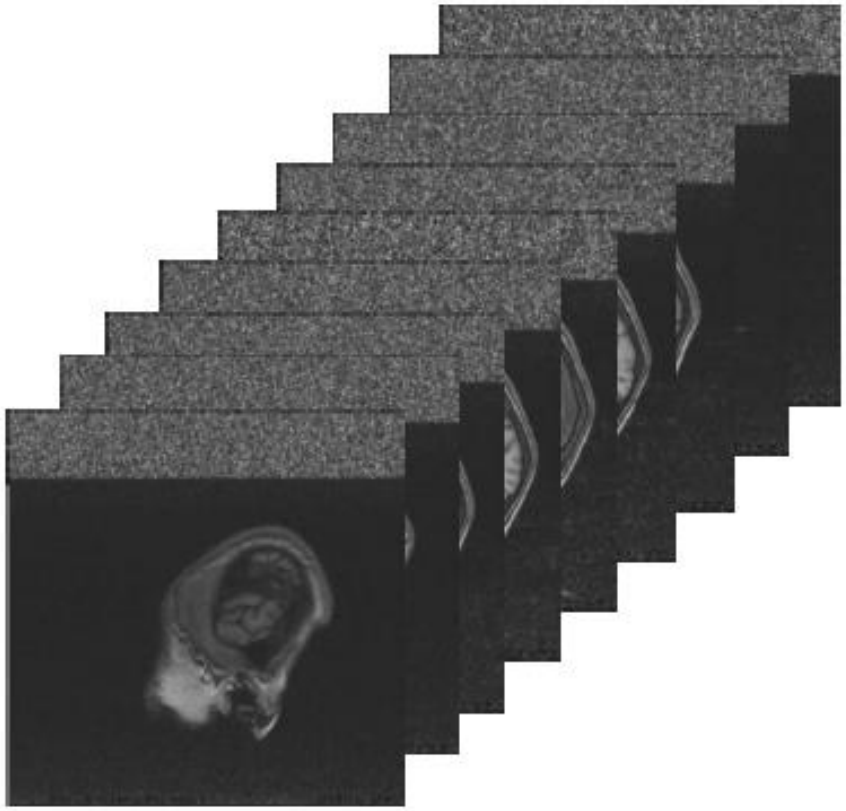
Model Construction

- Interactive modeling tools
  - CAD programs
  - Subdivision surface editors
- Scanning tools
  - CAT/MRI, Laser, robotic arm, etc.
- Computer vision
  - Stereo, motion, etc.
- Procedural generation
  - Sweeps, fractals, grammars

Scanning tools

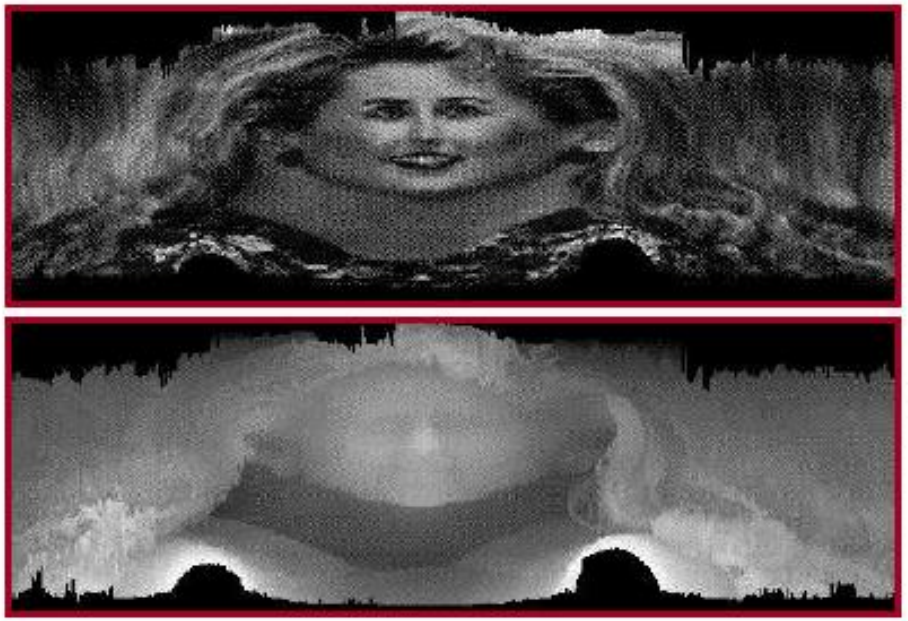
- Acquire geometry of objects with active sensors
  - CAT/MRI
  - Laser range scanner
  - Robotic arm

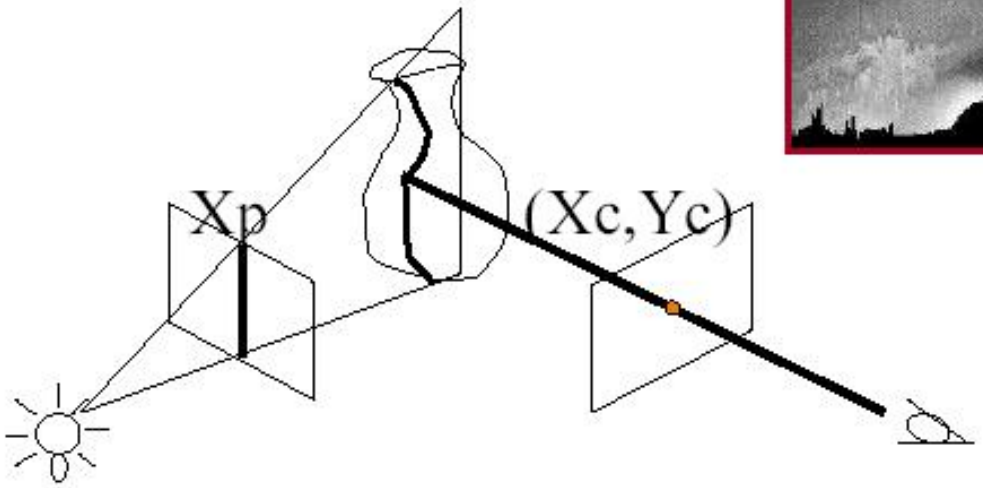




Scanning tools

- Acquire geometry of objects with active sensors
  - CAT/MRI
  - Laser range scanner
  - Robotic arm
  - etc.



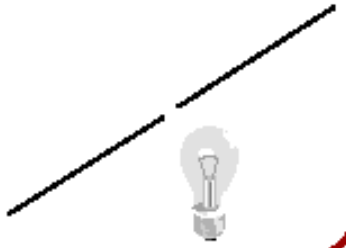


Scanning tools

Triangulation (in 2D):

To figure out the position of a point we need:

1. The position of the camera.
2. The position of the light source



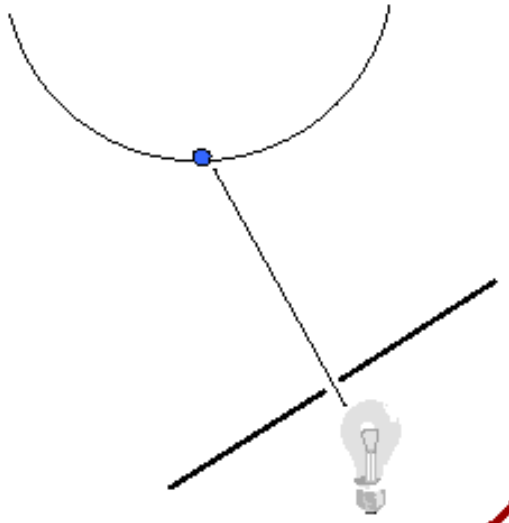
Scanning tools

Triangulation (in 2D):

To figure out the position of a point we need:

1. The position of the camera
2. The position of the light source

Project the light onto the surface…



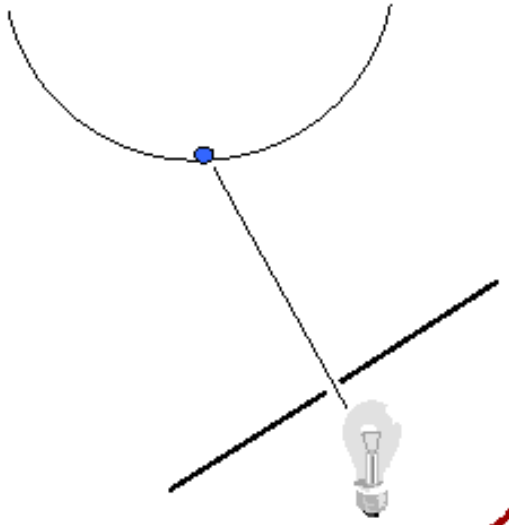
Scanning tools

Triangulation (in 2D):

To figure out the position of a point we need:

1. The position of the camera
2. The position of the light source

Project the light onto the surface…
Find where the lit point projects
onto the camera…



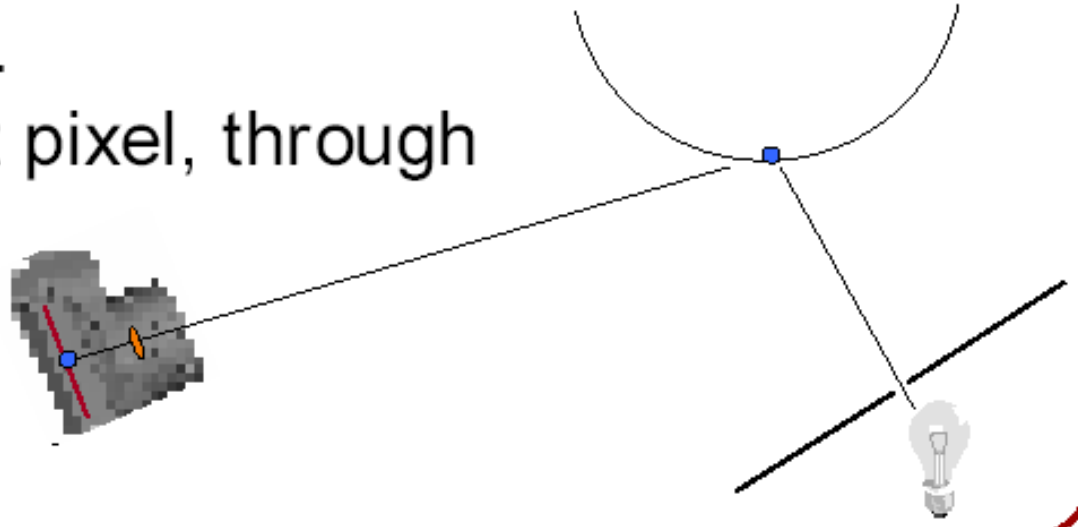
Scanning tools

Triangulation (in 2D):

To figure out the position of a point we need:

1. The position of the camera
2. The position of the light source

Project the light onto the surface…
Find where the lit point projects
onto the camera…
Cast a ray from the lit pixel, through
the camera…



Scanning tools

Triangulation (in 2D):

To figure out the position of a point we need:

1. The position of the camera
2. The position of the light source

Project the light onto the surface…

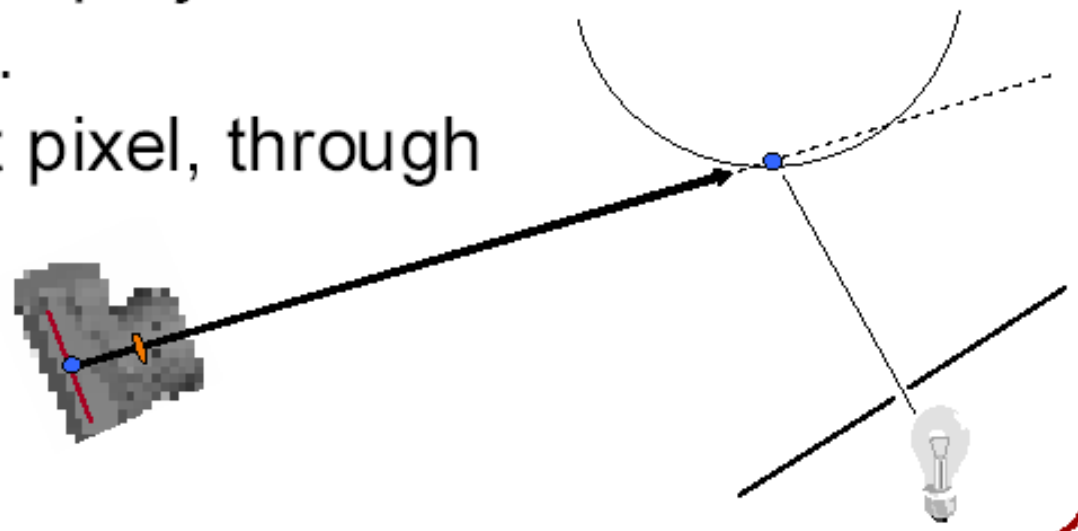
…int projects

…

…it pixel, through

the camera…

For the lit point to project onto the appropriate pixel, it has to lie somewhere on the ray.



Scanning tools

Triangulation (in 2D):

To figure out the position of a point we need:

1. The position of the camera
2. The position of the light source

Project the light onto the surface…

int projects

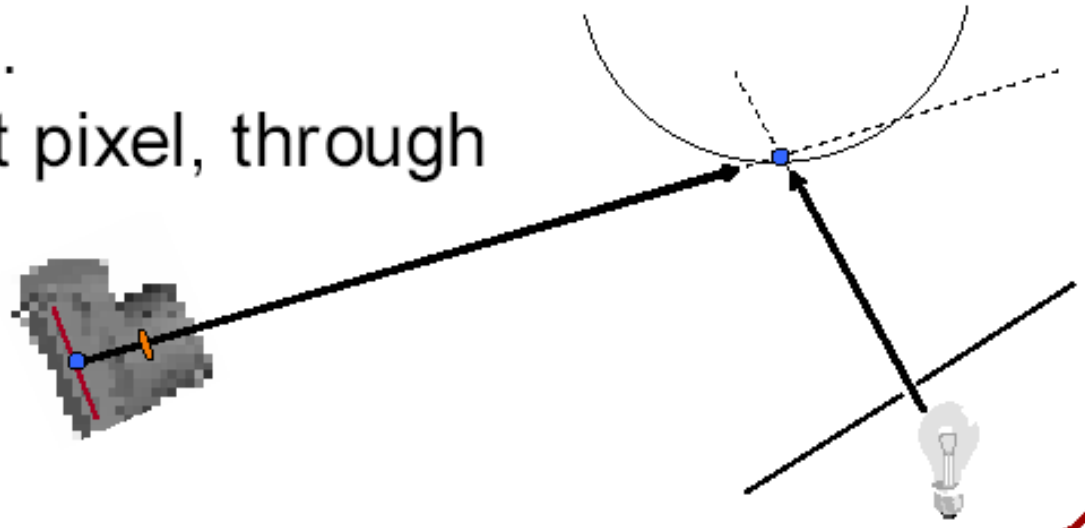
…

it pixel, through

the camera

For the lit point to project onto the appropriate pixel, it has to lie somewhere on the ray.

The lit point is also constrained to lie on the ray from the light source.



Scanning tools

Triangulation (in 2D):

To figure out the position of a point we need:

1. The position of the camera
2. The position of the light source

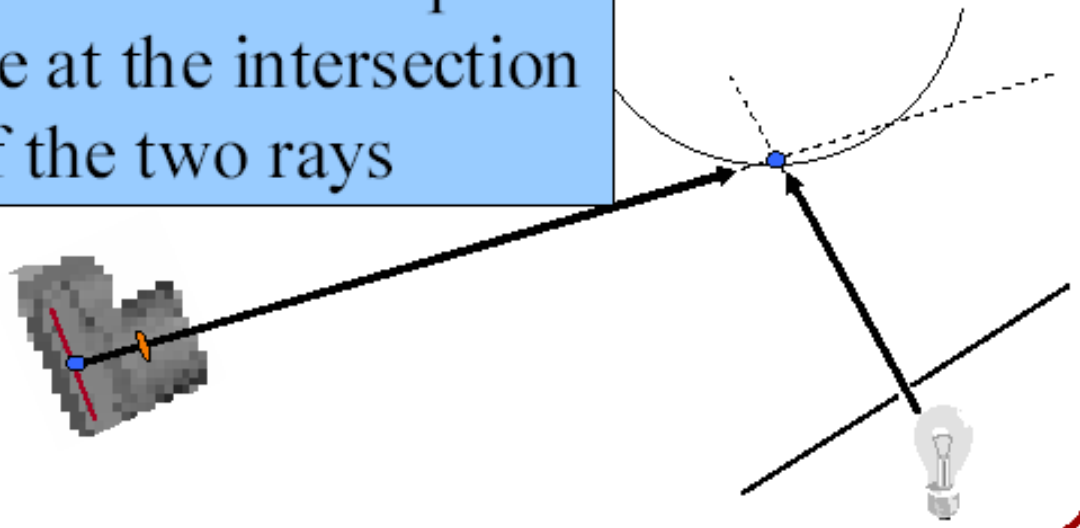
Project the light onto the surface…

For the lit point to
onto the appropria
it has to lie somev
the ray.

The position of the lit point has to be at the intersection of the two rays

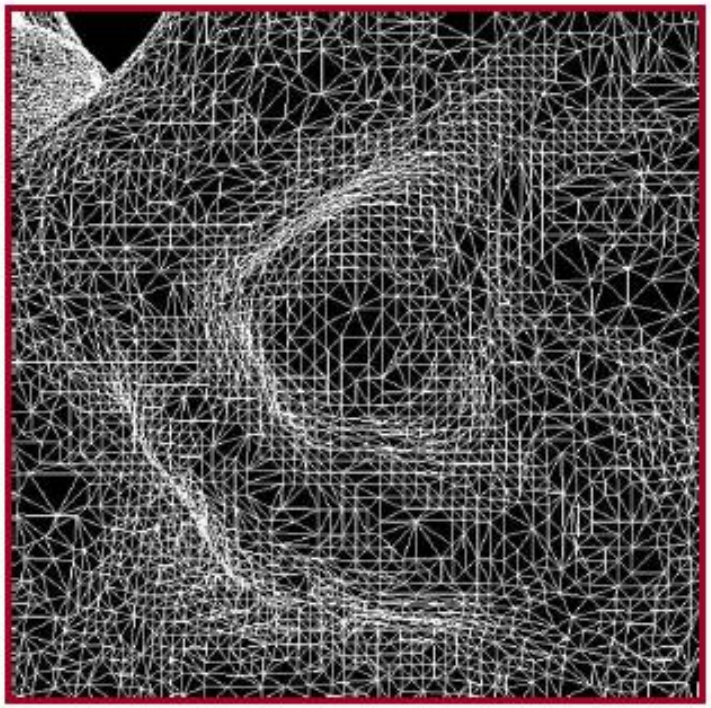
the camera

The lit point is also constrained to lie on the ray from the light source.



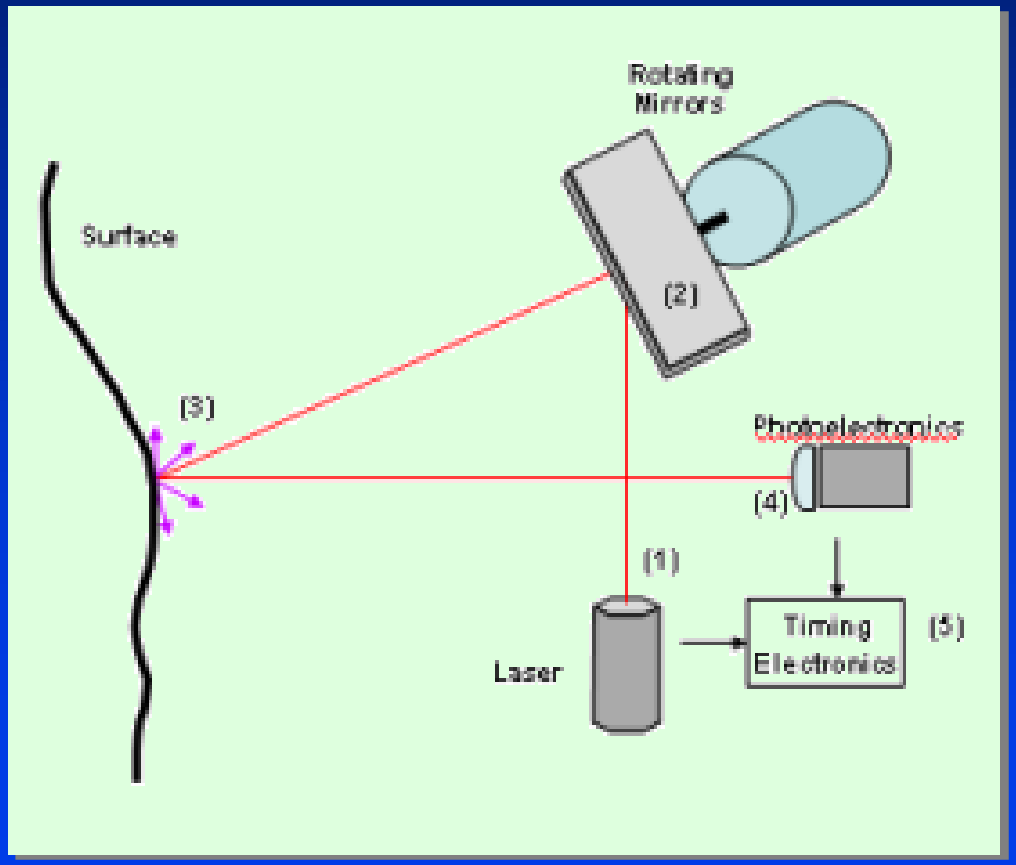
Laser Range Scanning

- Example: 70 scans



Stanford Graphics Laboratory

Other Range Scanning Tool



Scanning tools

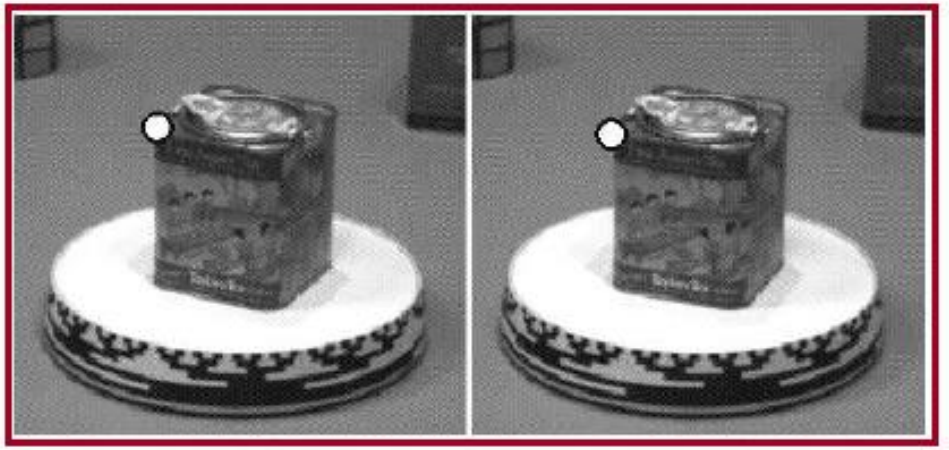
- Acquire geometry of objects with active sensors
  - CAT/MRI
  - Laser range scanner
  - Magnetic sensor
  - Robotic arm
  - etc.

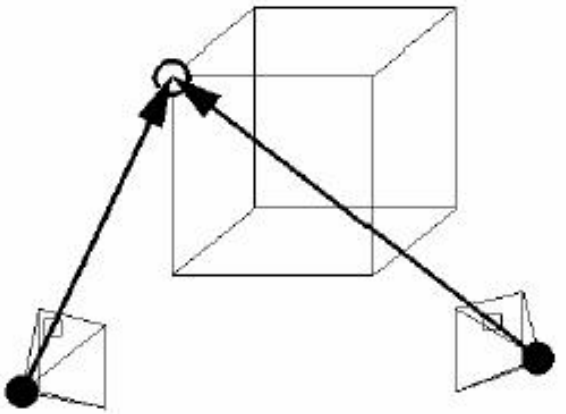
Model Construction

- Interactive modeling tools
  - CAD programs
  - Subdivision surface editors :)
- Scanning tools
  - Laser, magnetic, robotic arm, etc.
- Computer vision
  - Stereo, motion, etc.
- Procedural generation
  - Sweeps, fractals, grammars

Computer Vision

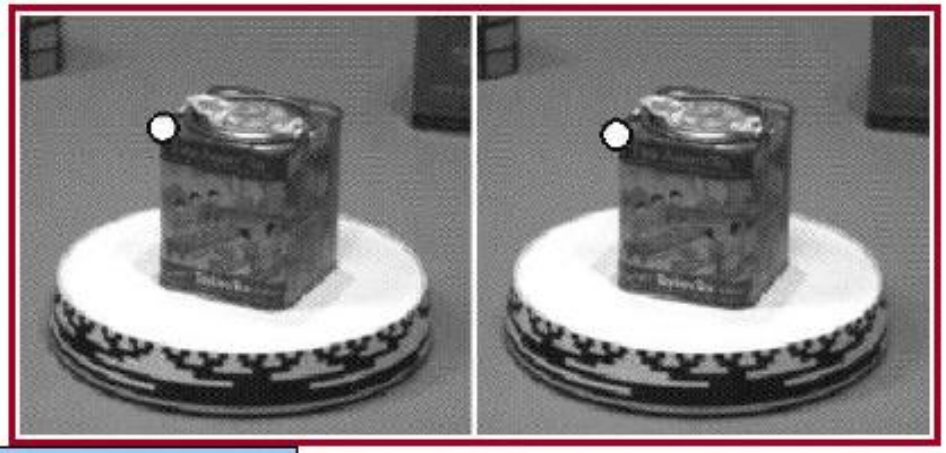
- Infer 3D geometry from images
  - Stereo
  - Motion
  - etc.



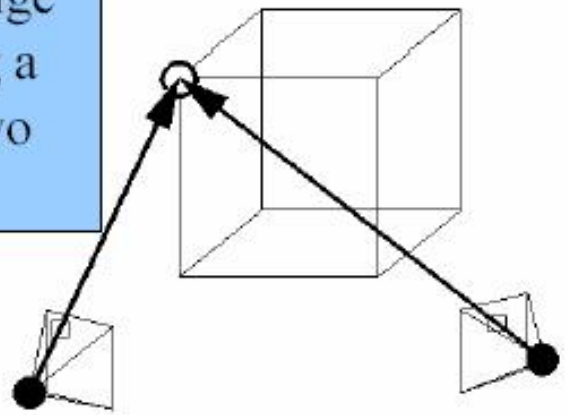


Computer Vision

- Infer 3D geometry from images
  - Stereo
  - Motion
  - etc.

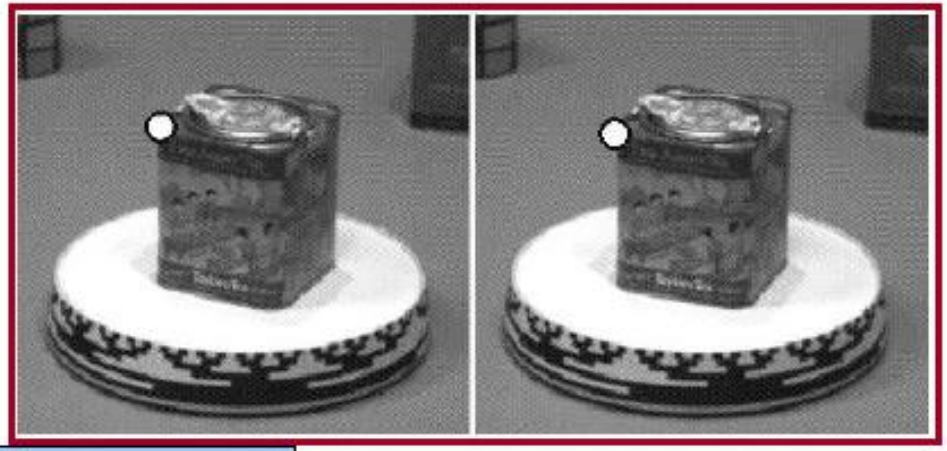


This is similar to the approach for laser range scanners, but instead of triangulating using a light and a camera, we triangulate using two cameras.



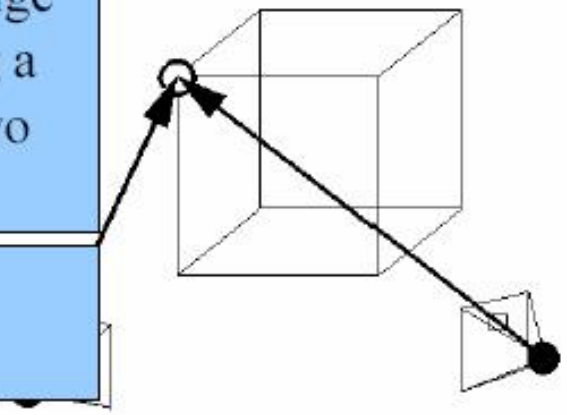
Computer Vision

- Infer 3D geometry from images
  - Stereo
  - Motion
  - etc.



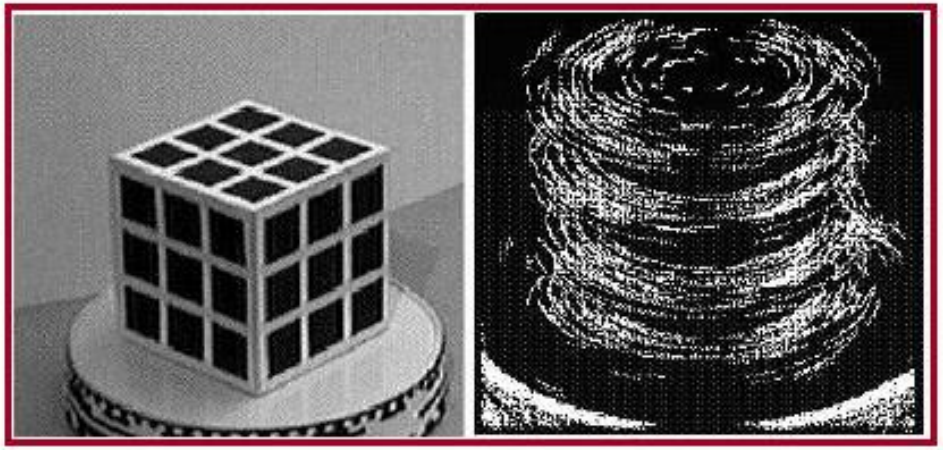
This is similar to the approach for laser range scanners, but instead of triangulating using a light and a camera, we triangulate using two cameras.

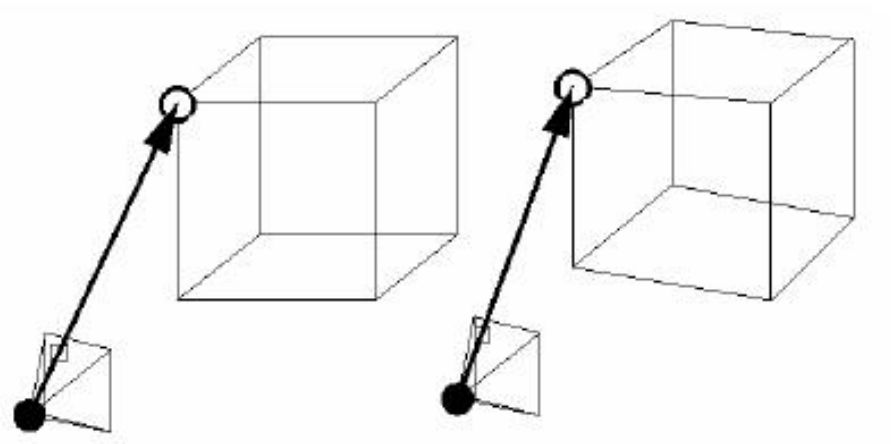
The challenge is to determine pixel pair correspondences across the two images.



Computer Vision

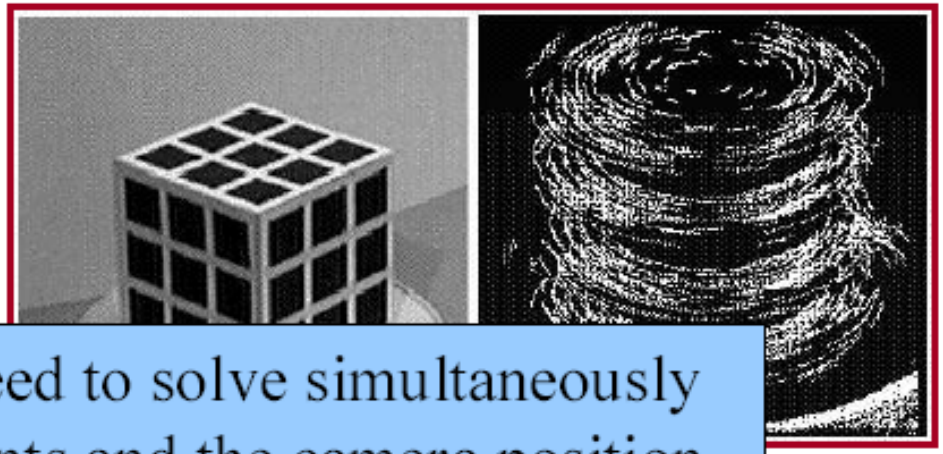
- Infer 3D geometry from images
  - Stereo
  - Motion
  - etc.





Computer Vision

- Infer 3D geometry from images
  - Stereo
  - Motion
  - etc.



In this case we need to solve simultaneously for the surface points and the camera position.