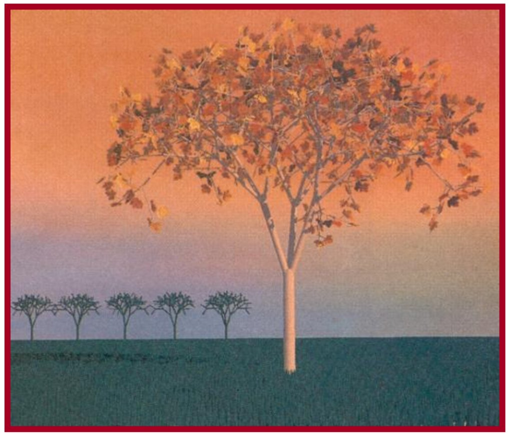
Modeling

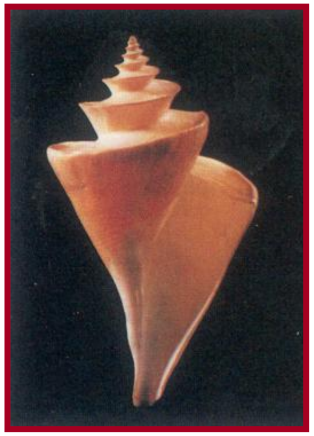
Michael Kazhdan

(601.457/657)

Modeling

- How do we construct 3D models quickly and/or automatically with a computer?


H&B Figure 10.79


Fowler

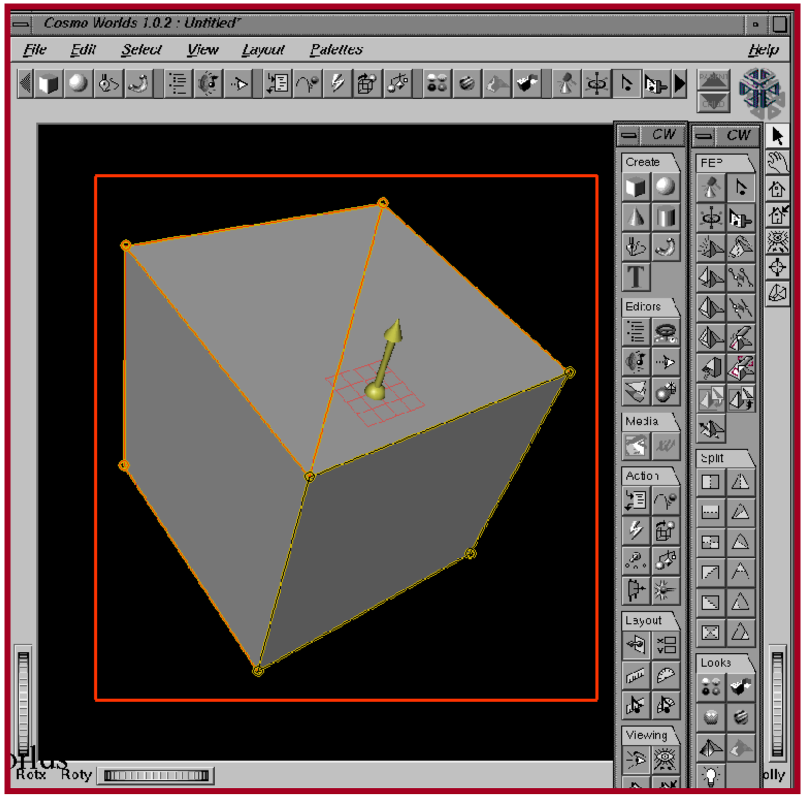
H&B Figure 10.83b

Model Construction

- Interactive modeling tools
  - CAD programs
- Active scanners
  - Light probes, triangulation/LiDAR/CAT/MRI scanners
- Passive scanners
  - Stereo, motion, etc.
- Procedural generation
  - Sweeps, fractals, grammars

Interactive Modeling Tools

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



Cosmoworlds, SGI

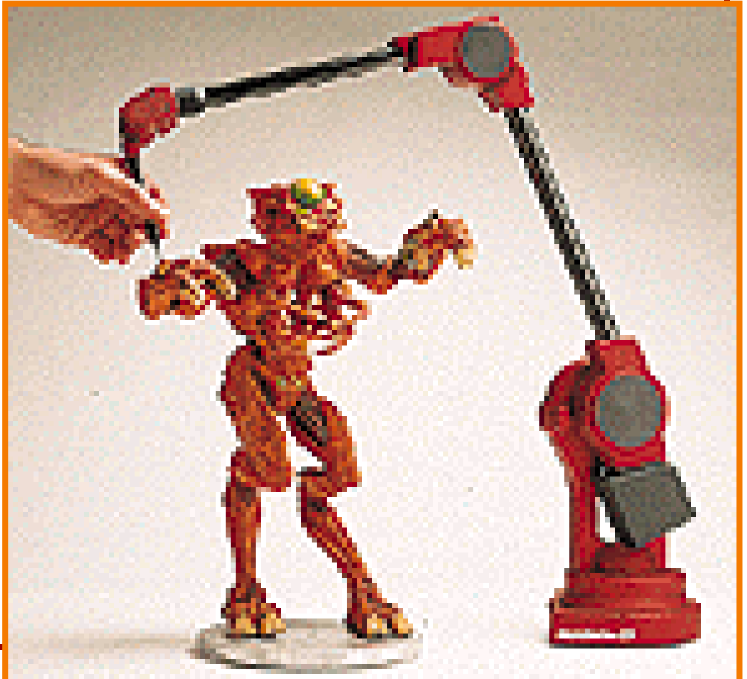
Model Construction

- Interactive modeling tools
  - CAD programs
- Active scanners
  - Light probes, triangulation/LiDAR/CAT/MRI scanners
- Passive scanners
  - Stereo, motion, etc.
- Procedural generation
  - Sweeps, fractals, grammars

Active Scanners: Touch Scanner

- Articulated arm with:
  - Angular sensors at the joints
  - Touch probe at the end
- Angles are recorded on contact, and use to compute the position of the head

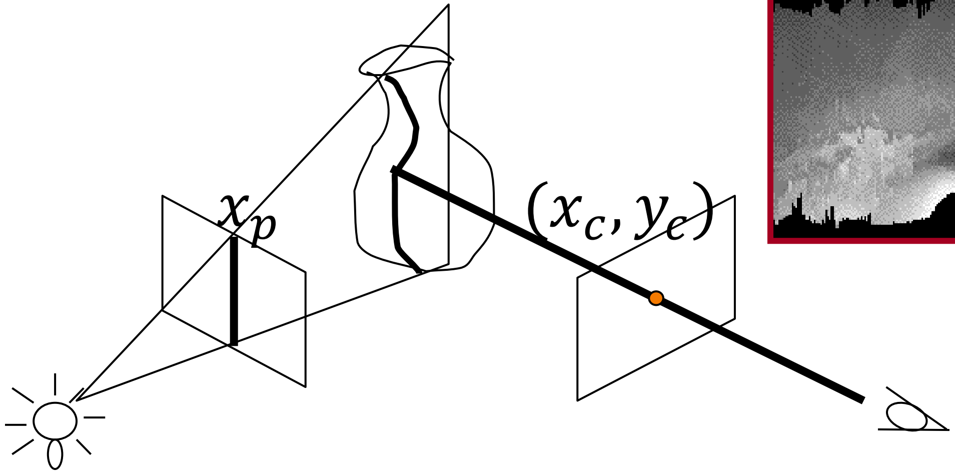
⇒ Position of the probe (relative to the base of the scanner)

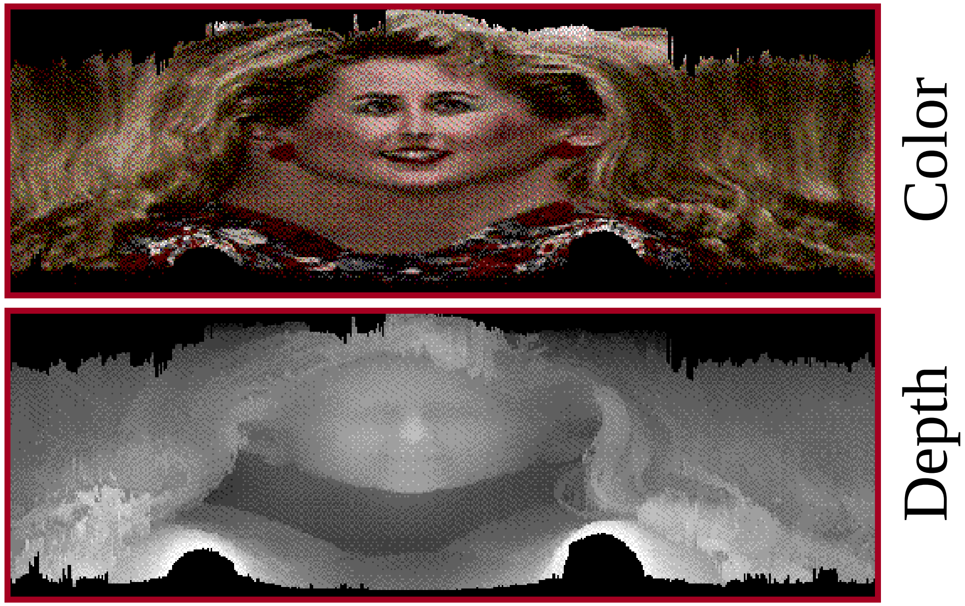


Active Scanners: Laser Scanner

- Emits a laser from one position
- Images the laser-illuminated surface from another (calibrated position)

⇒ Triangulation

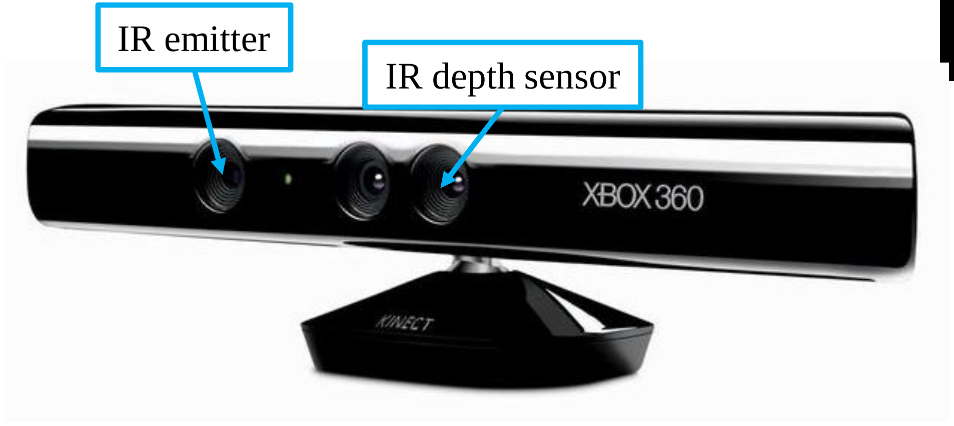




Active Scanners: IR Scanner

- Emits IR light from one position
- Images the IR-illuminated surface from another (calibrated position)

⇒ Triangulation

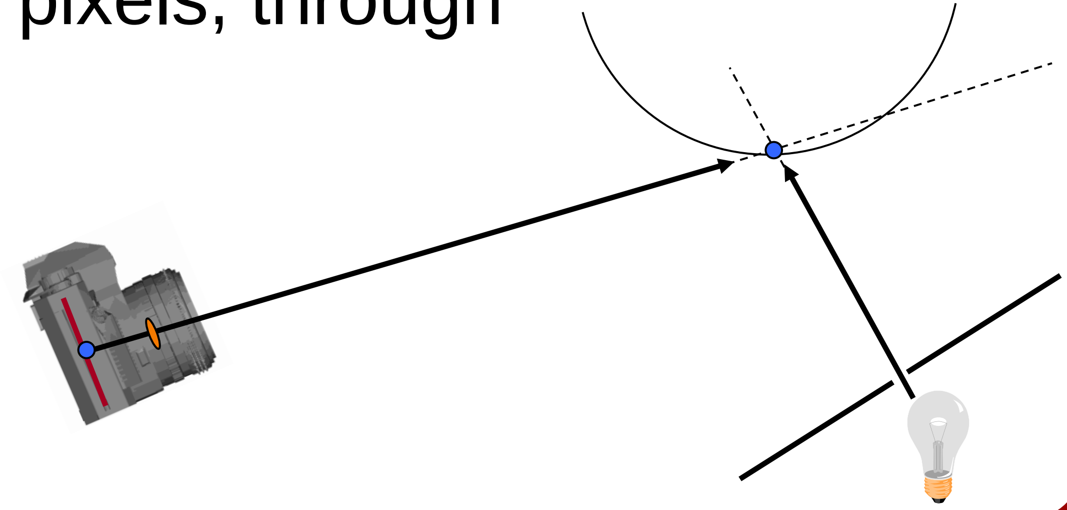


Active Scanners: Triangulation (2D)

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 through a slit onto the surface…
Find where the lit points project
onto the camera…
Cast rays from the lit pixels, through
the camera…



Active Scanners: Triangulation 2D

To figure out the position of a point we need:

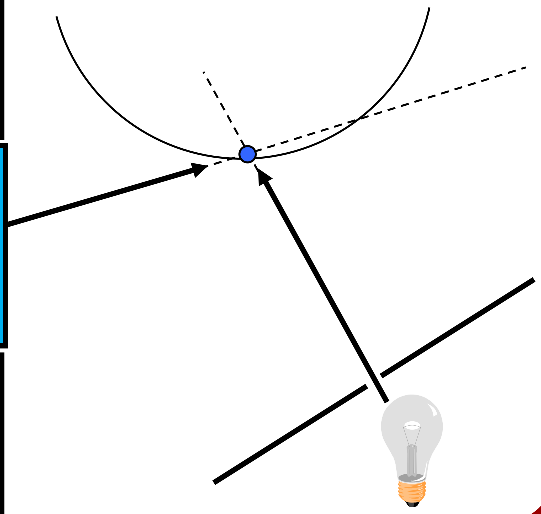
1. The position of the camera
2. The position of the light source

Project the light through a slit onto the surface…
Find where the lit points project
onto the camera…

For the lit point to project onto the pixel, it has to lie along the ray.

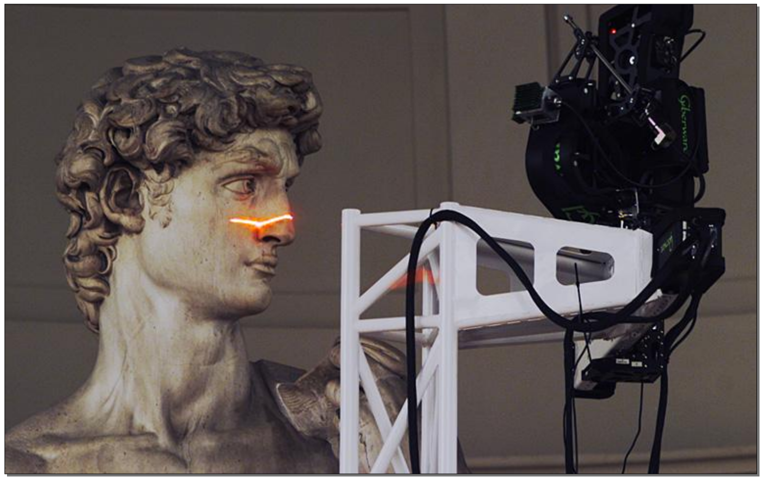
The lit point is also constrained to lie on the plane from the light source.

The position of the lit point has to be at the intersection of the ray and the plane.

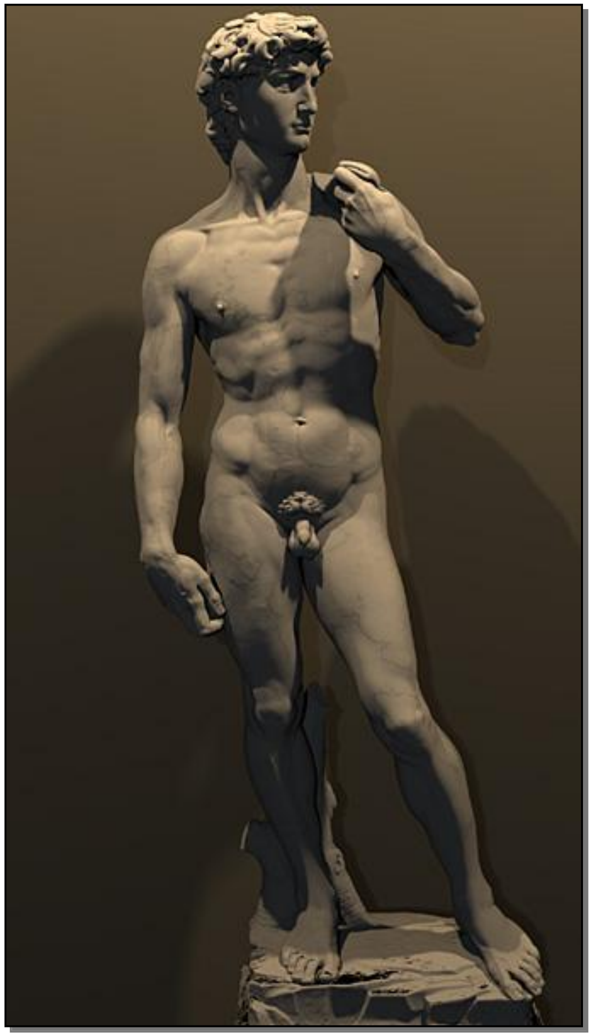


Laser Range Scanning

- Example: Digital Michelangelo Project
  - 480 individually aimed scans



Stanford Graphics Laboratory



Active Scanners: LiDAR

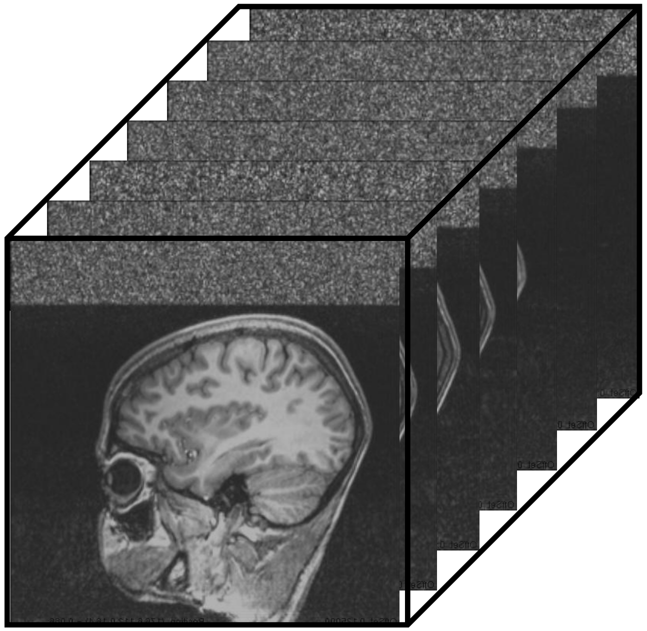
- Emits laser light
- Measures the time it takes the light to reflect back to a sensor.

⇒ Knowing the speed of light gives (twice) the distance travelled.

Image courtesy of Wikipedia

Active Scanners: MRI Scanners

- …

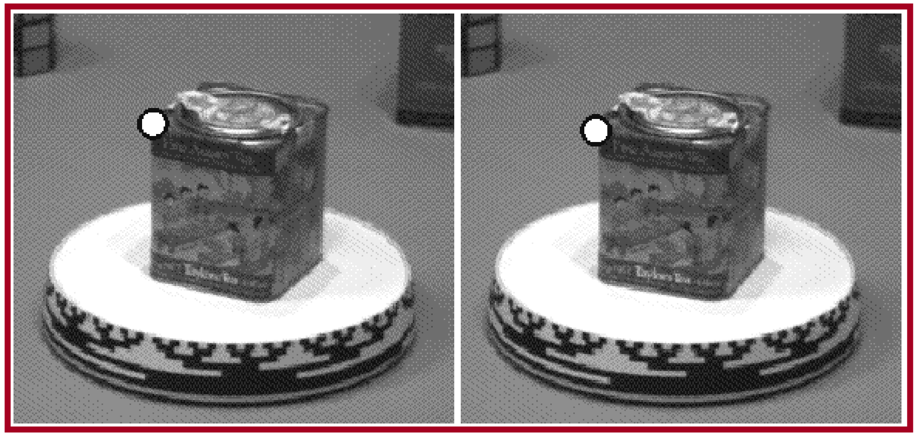


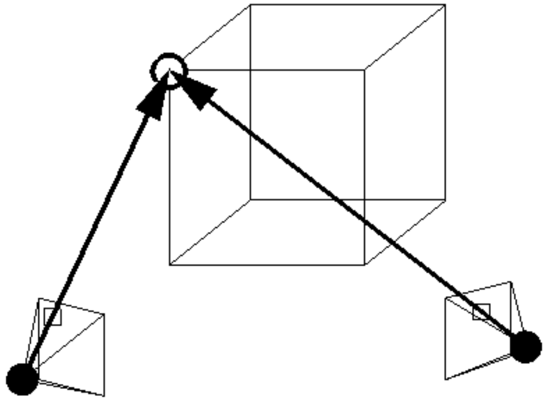
Model Construction

- Interactive modeling tools
  - CAD programs
- Active scanners
  - Light probes, triangulation/LiDAR/CAT/MRI scanners
- Passive scanners
  - Stereo, motion, etc.
- Procedural generation
  - Sweeps, fractals, grammars

Passive Scanners: Stereo

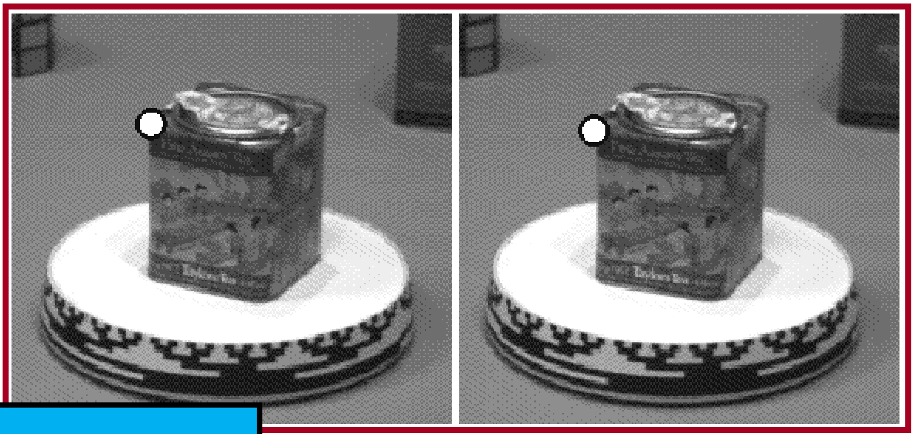
- Image the scene from multiple calibrated locations





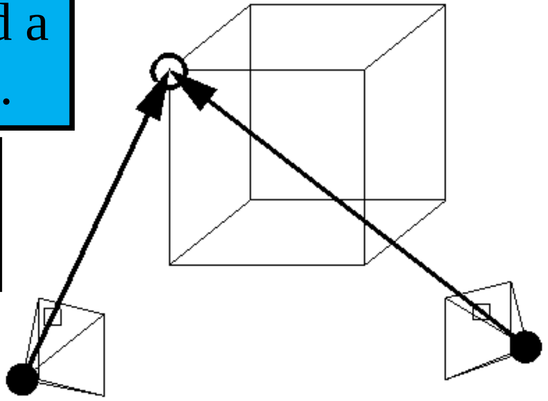
Passive Scanners: Stereo

- Image the scene from multiple calibrated locations



This is similar to the approach for 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.



Passive Scanners: Motion

- Image the scene from multiple uncalibrated locations

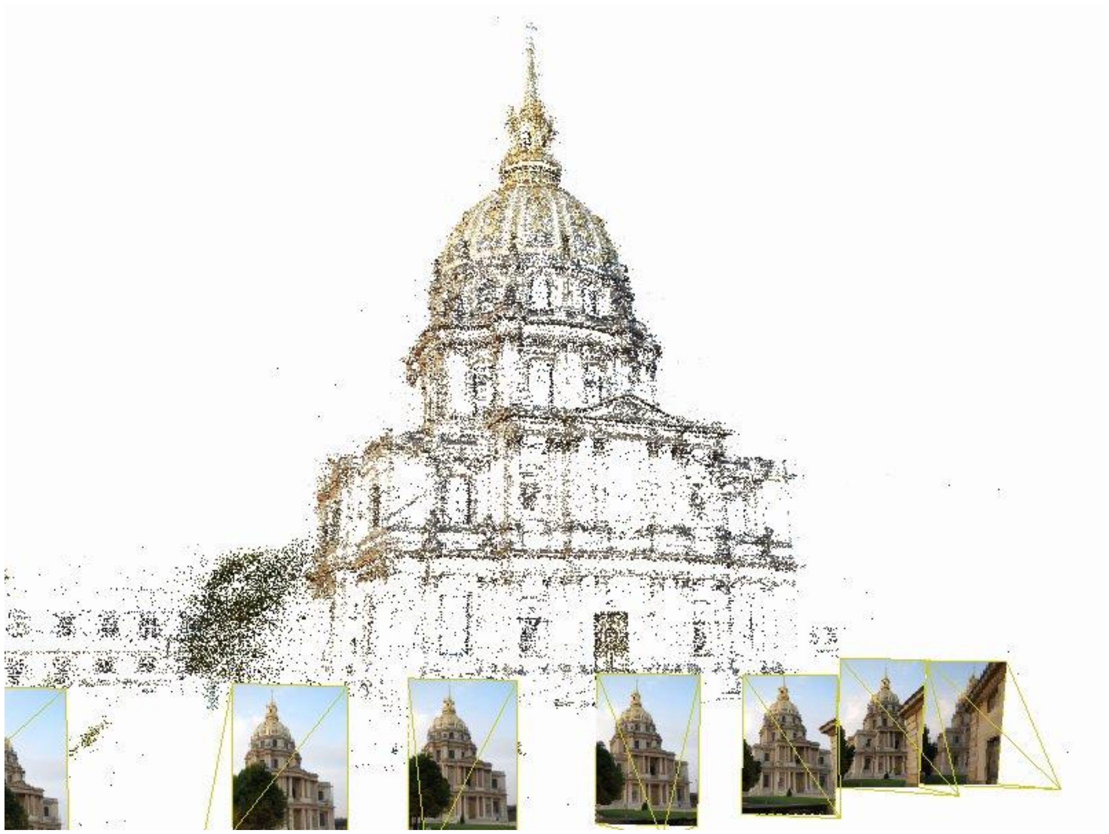
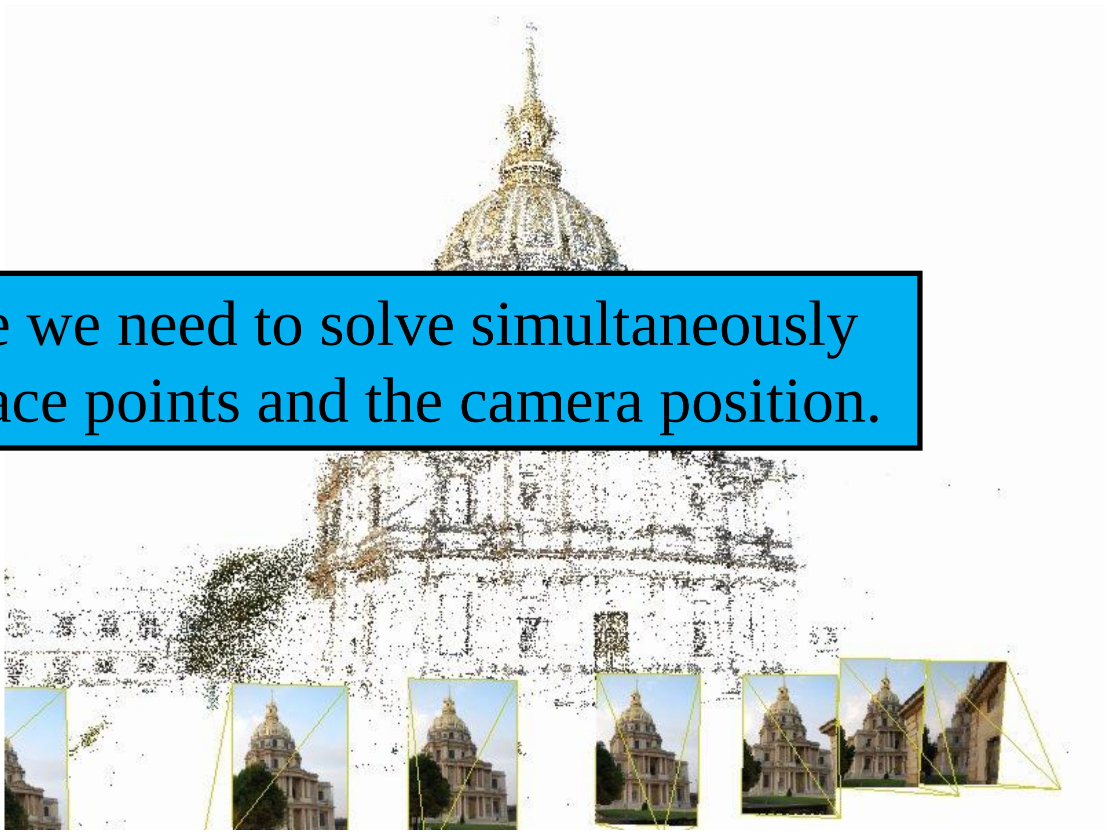


Image courtesy of http://www.maths.lth.se/matematiklth/personal/calle/

Passive Scanners: Motion

- Image the scene from multiple uncalibrated locations



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

Image courtesy of http://www.maths.lth.se/matematiklth/personal/calle/

Model Construction

- Interactive modeling tools
  - CAD programs
- Active scanners
  - Light probes, triangulation/LiDAR/CAT/MRI scanners
- Passive scanners
  - Stereo, motion, etc.
- Procedural generation
  - Sweeps, fractals, grammars

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

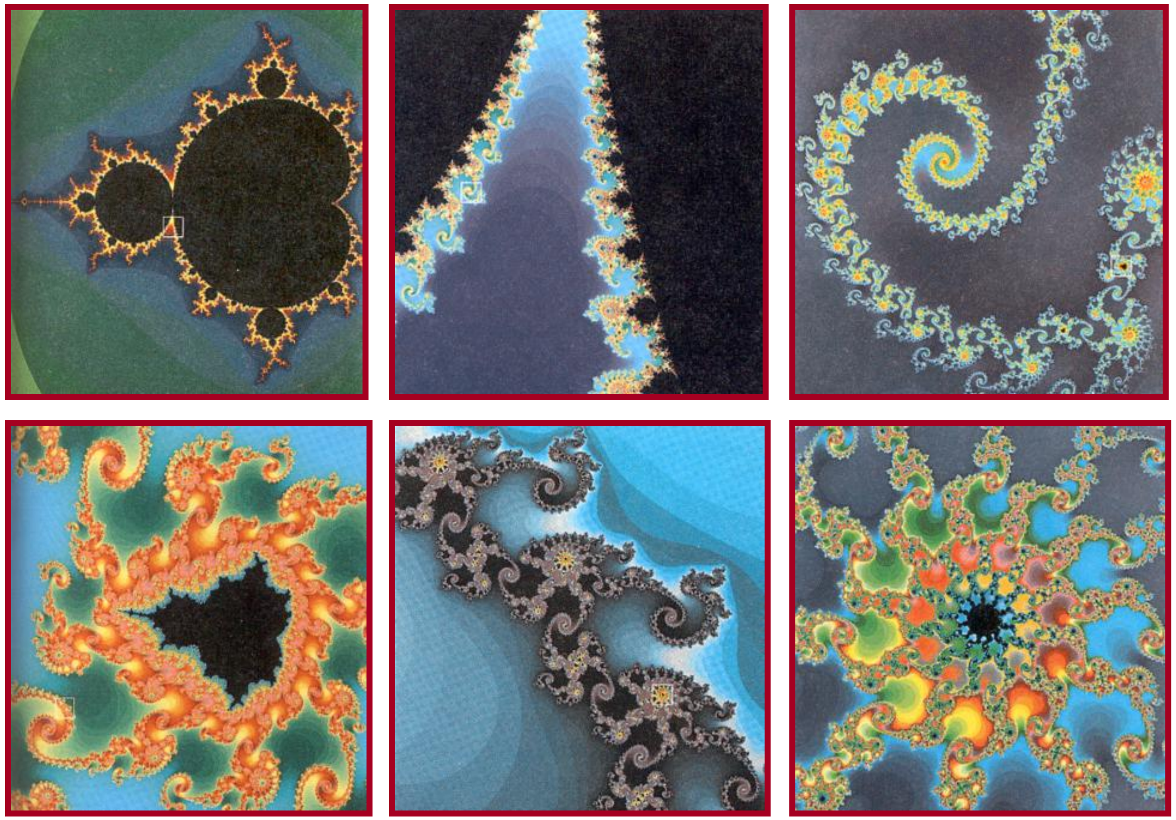
Similar to Perlin noise

Procedural Modeling: Sweeps

- Transform a generating curve along a sweep curve

Procedural Modeling: Fractals

- Defining property:
  - Self-similar with infinite resolution



Mandelbrot Set

H&B Figure 10.100

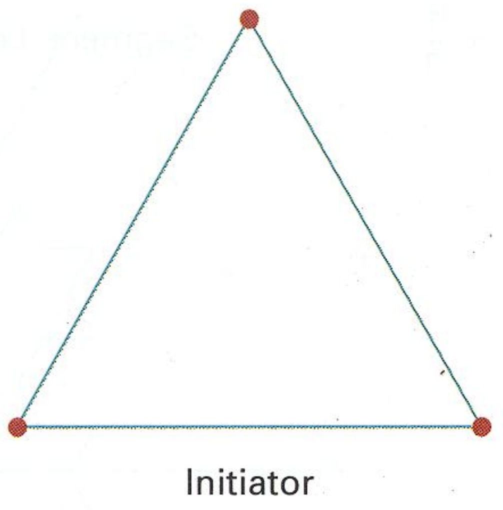
Procedural Modeling: Fractals

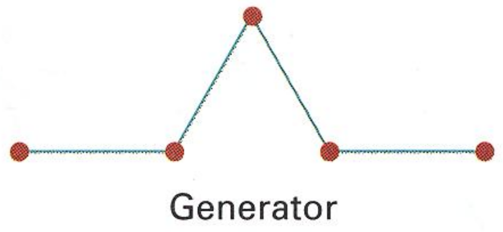
- Deterministically self-similar fractals
  - Parts are scaled copies of original

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

Procedural Modeling: Fractals

- Deterministic fractal generation:
  - Initiator: start with a shape
  - Generator: replace subparts with scaled copy of original
  - Repeat

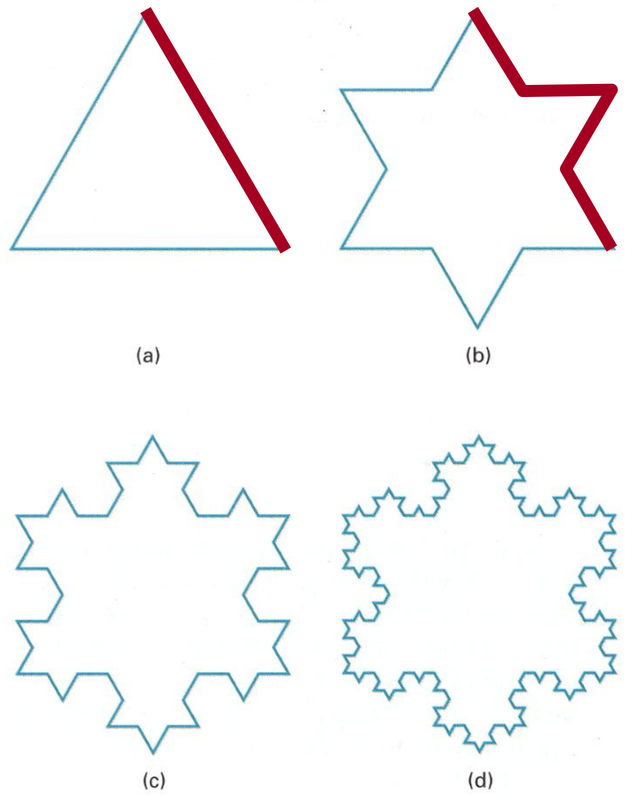




H&B Figure 10.68

Procedural Modeling: Fractals

- Deterministic fractal generation:
  - Initiator: start with a shape
  - Generator: replace subparts with scaled copy of original
  - Repeat

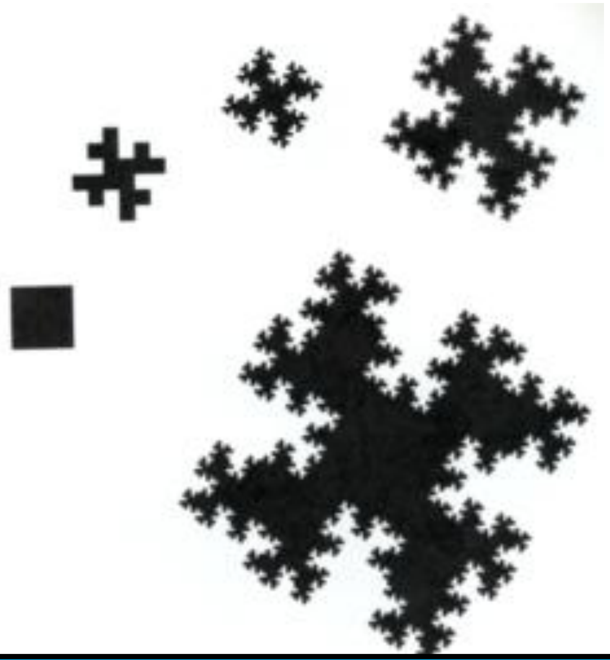


Koch Curve

H&B Figure 10.69

Procedural Modeling: Fractals

- Deterministic fractal generation:
  - Initiator: start with a shape
  - Generator: replace subparts with scaled copy of original
  - Repeat



Useful for creating "interesting" shapes.

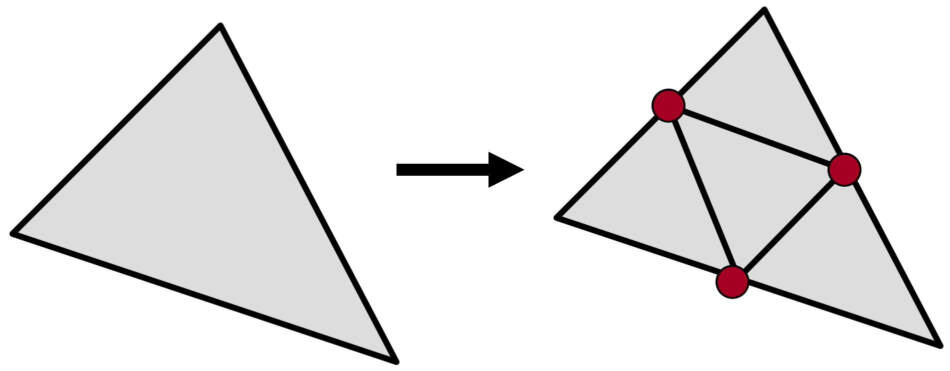
Mandelbrot Figure X

Procedural Modeling: Fractals

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

Procedural Modeling: Fractals

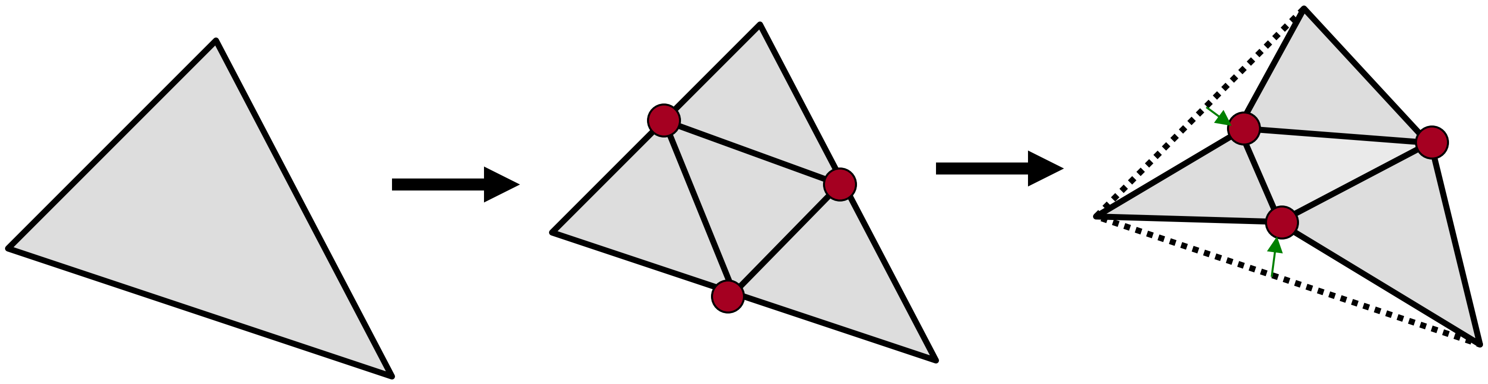
- Statistical fractal generation:
  - Initiator: start with a shape
  - Generator: replace subparts with a scaled copy
  - Repeat



Procedural Modeling: Fractals

- Statistical fractal generation:
  - Initiator: start with a shape

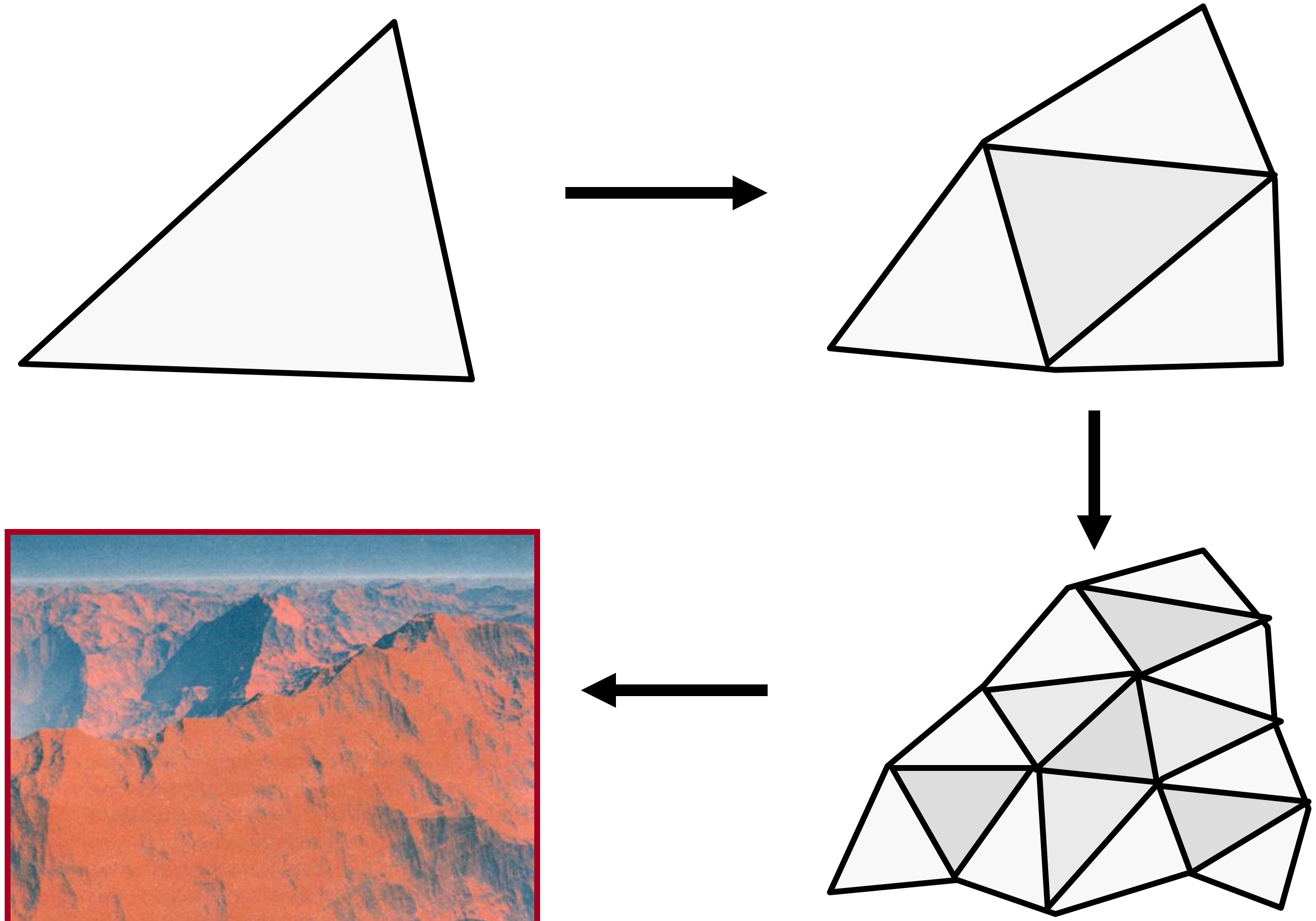
    statistically self-similar
  - Generator: replace subparts with^a scaled copy
  - Repeat

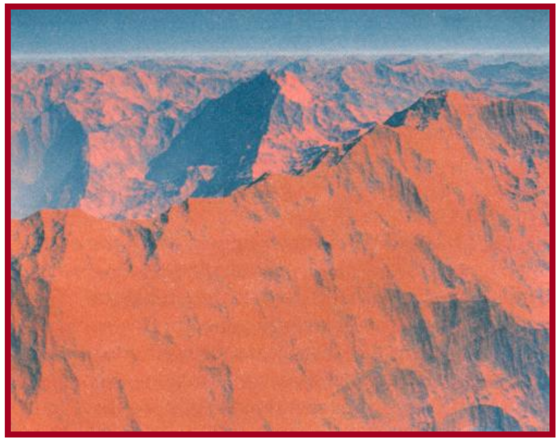


Random Midpoint Displacement

Procedural Modeling: Fractals

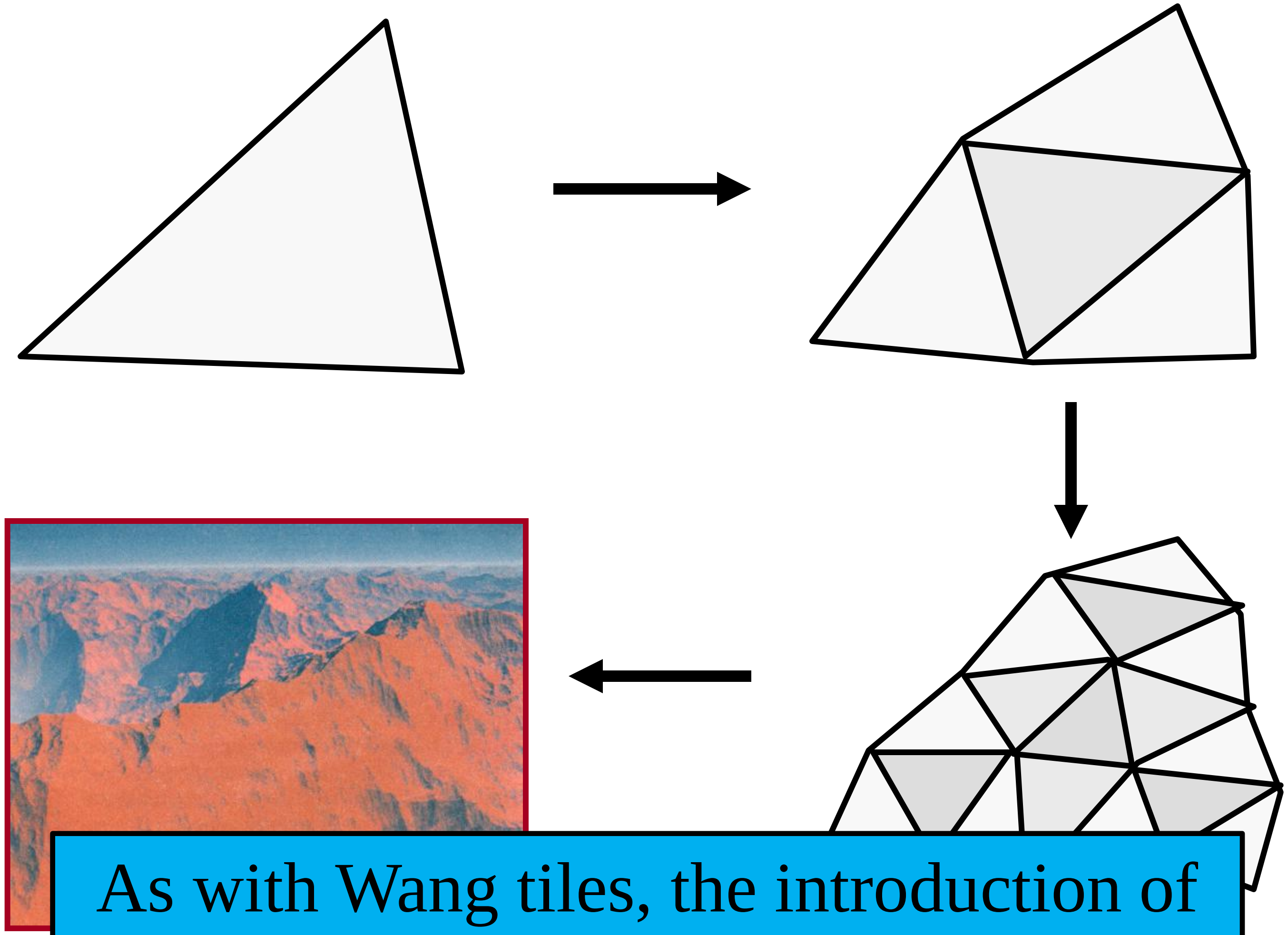
- Example: terrain




H&B Figure 10.83b

Procedural Modeling: Fractals

- Example: terrain



As with Wang tiles, the introduction of randomness removes repetitiveness

Statistical Fractal Generation

- Useful for creating mountains

H&B Figure 10.83a

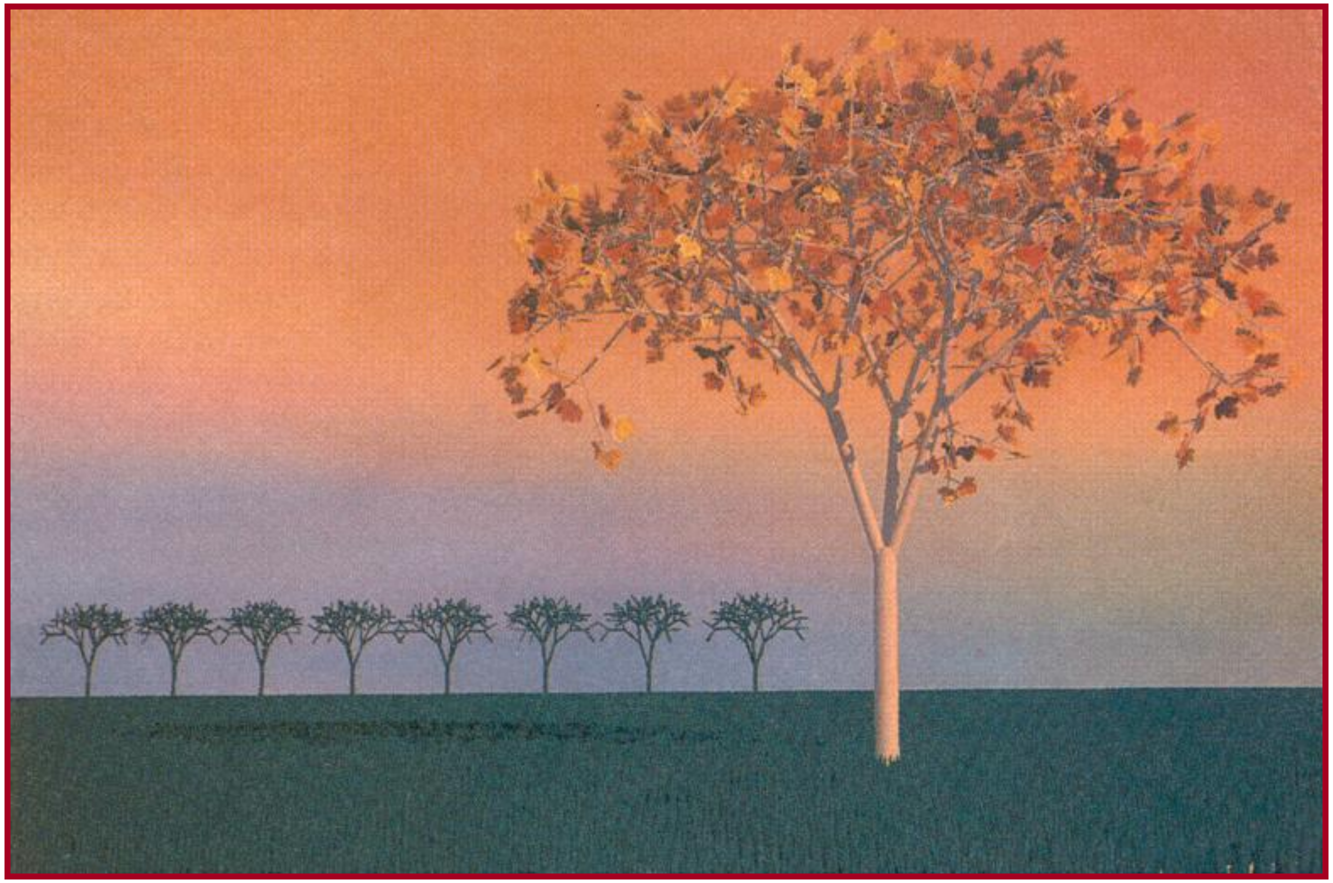
Statistical Fractal Generation

- 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: Grammars

- Generate description of geometric model by applying production rules

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

Procedural Modeling: Grammars

- Generate description of geometric model by applying production rules

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

AB

Procedural Modeling: Grammars

- Generate description of geometric model by applying production rules

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

$\boxed{A}B$

BaB

Procedural Modeling: Grammars

- Generate description of geometric model by applying production rules

$$S \longrightarrow AB$$
$$A \longrightarrow Ba \;|\; a$$
$$\boxed{B \longrightarrow Ab \;|\; b}$$

AB

$Ba\boxed{B}$

BaAb

Procedural Modeling: Grammars

- Generate description of geometric model by applying production rules

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

AB
BaB
$\boxed{\text{B}}$aAb
AbaAb

Procedural Modeling: Grammars

- Generate description of geometric model by applying production rules

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

AB
BaB
BaAb
AbaAb
⋮

Procedural Modeling: Grammars

- Useful for creating plants

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge

🟢 = leaf

| = edge

🔵 = Junction

◌ = [Root]

Procedural Modeling: Grammars

- Useful for creating plants [R]

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge

🟢	= leaf
|	= edge
🔵	= Junction
◌	= [Root]

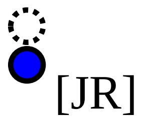
Procedural Modeling: Grammars

- Useful for creating plants

$$\begin{tikzcd}[R] \arrow[d] \\ JR\end{tikzcd}$$

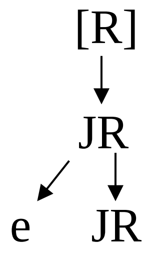
Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge

🟢	= leaf
┃	= edge
🔵	= Junction
◌	= [Root]

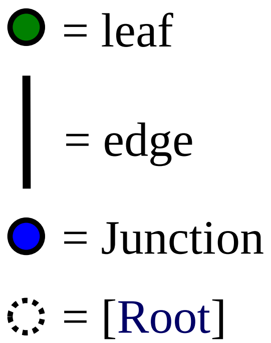


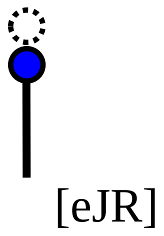
Procedural Modeling: Grammars

- Useful for creating plants



Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge





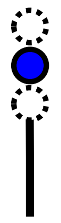
Procedural Modeling: Grammars

- Useful for creating plants

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge

$$\begin{tikzcd}[R] \arrow[d] & \\ JR \arrow[d] \arrow[dl] & \\ e & JR \arrow[dl] \arrow[d] \\ [R] & JR\end{tikzcd}$$

●	= leaf
|	= edge
●	= Junction
◌	= [Root]



[e[R]JR]

Procedural Modeling: Grammars

- Useful for creating plants

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge

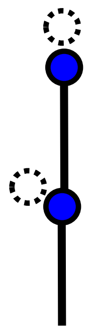
$$\begin{tikzcd}
 & & {[R]} \arrow[d] & \\
 & & JR \arrow[dl] \arrow[d] & \\
 & e & JR \arrow[dl] \arrow[d] & \\
 & {[R]} \arrow[d] & JR \arrow[d] \arrow[dr] & \\
 & JR & e & JR
\end{tikzcd}$$

● = leaf

| = edge

● = Junction

◌ = [Root]


[e[JR]eJR]

Procedural Modeling: Grammars

- Useful for creating plants

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge

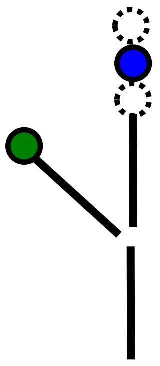
$$\begin{tikzcd}
 & & \text{[R]} \arrow[d] & & & & \\
 & & \text{JR} \arrow[dl] \arrow[dr] & & & & \\
 & \text{e} & & \text{JR} \arrow[dl] \arrow[dr] & & & \\
 & & \text{[R]} \arrow[d] & & \text{JR} \arrow[d] \arrow[dr] & & \\
 & & \text{JR} \arrow[dl] \arrow[dr] & & \text{e} & \text{JR} \arrow[d] \arrow[dr] & \\
 & \text{e} & & \text{l} & & \text{[R]} & \text{JR}
\end{tikzcd}$$

● = leaf

| = edge

● = Junction

◌ = [Root]


[e[el]e[R]JR]

Procedural Modeling: Grammars

- Useful for creating plants

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge

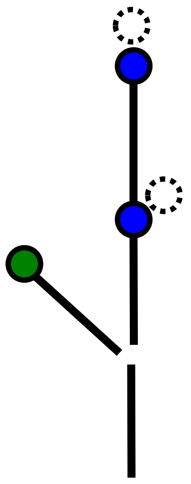
$$\begin{tikzcd}
 & & & {[R]} \arrow[d] & & & \\
 & & & JR \arrow[dl] \arrow[d] & & & \\
 & & e & JR \arrow[dl] \arrow[d] & & & \\
 & & {[R]} \arrow[d] & JR \arrow[d] \arrow[dr] & & & \\
 & & JR \arrow[dl] \arrow[dr] & e & JR \arrow[d] \arrow[dr] & & \\
 & e & & l & {[R]} \arrow[d] & JR \arrow[d] \arrow[dr] & \\
 & & & & JR & e & JR
\end{tikzcd}$$

● = leaf

| = edge

● = Junction

◌ = [Root]

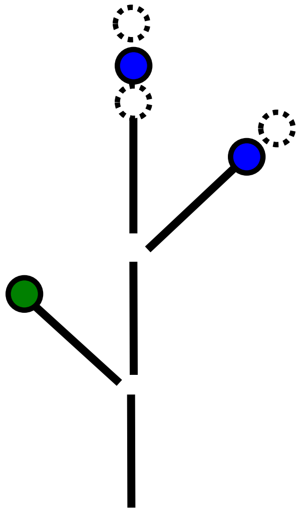


[e[el]e[JR]eJR]

Procedural Modeling: Grammars

- Useful for creating plants

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge



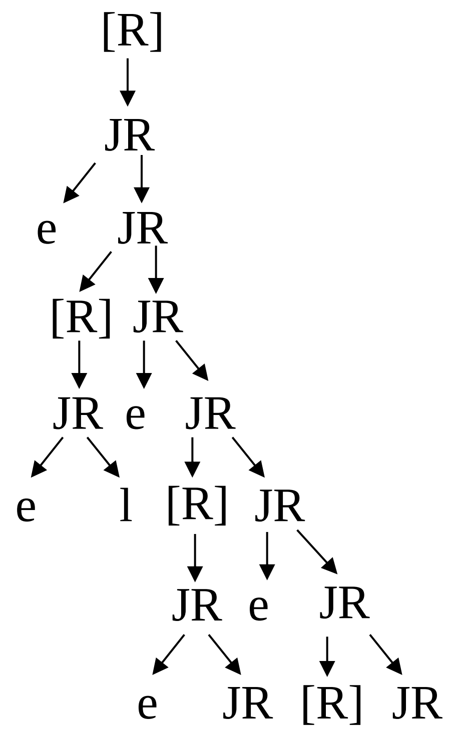
[e[el]e[eJR]e[R]JR]

● = leaf

| = edge

● = Junction

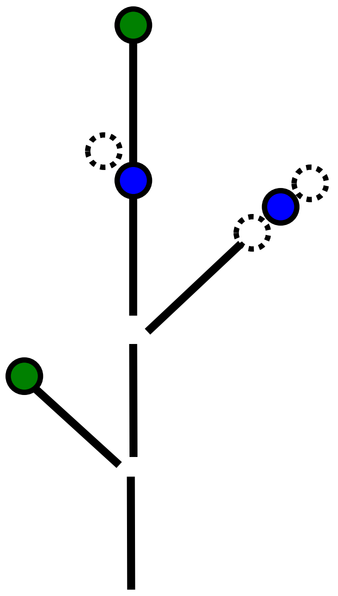
◌ = [Root]



Procedural Modeling: Grammars

- Useful for creating plants

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge



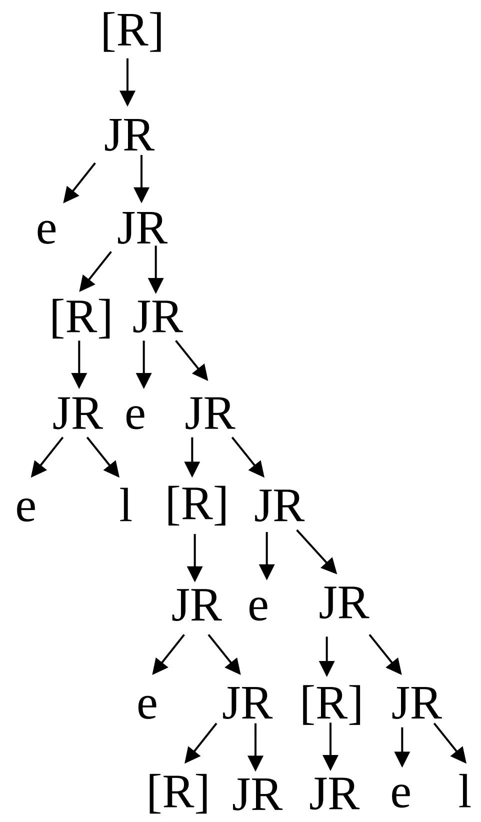
● = leaf

| = edge

● = Junction

◌ = [Root]

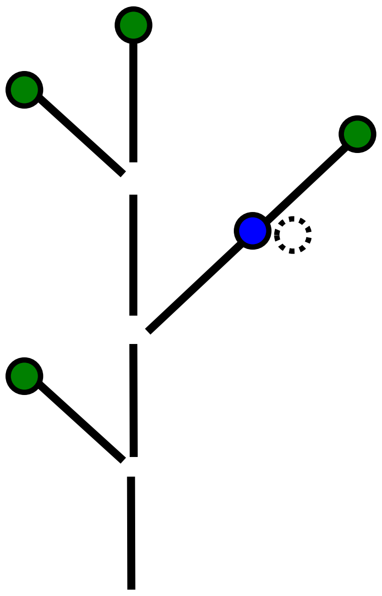
[e[el]e[e[R]JR]e[JR]el]



Procedural Modeling: Grammars

- Useful for creating plants

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge



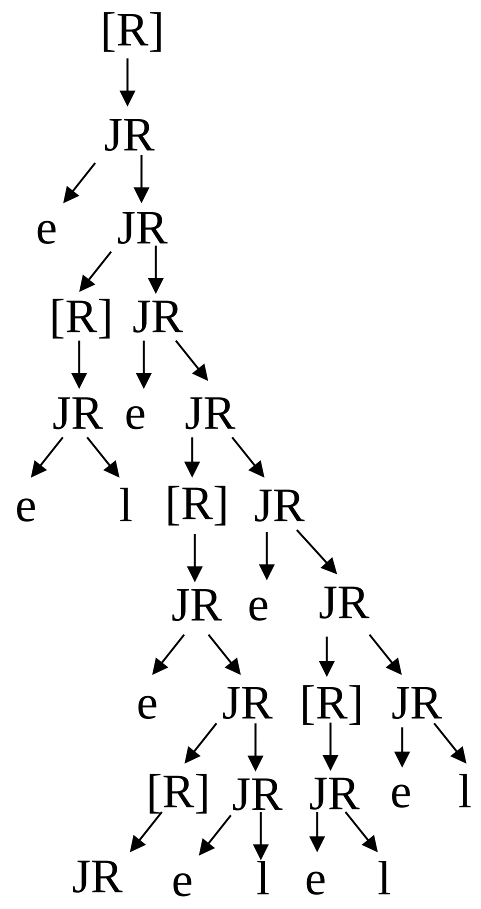
[e[el]e[e[JR]el]e[el]el]

🟢 = leaf

| = edge

🔵 = Junction

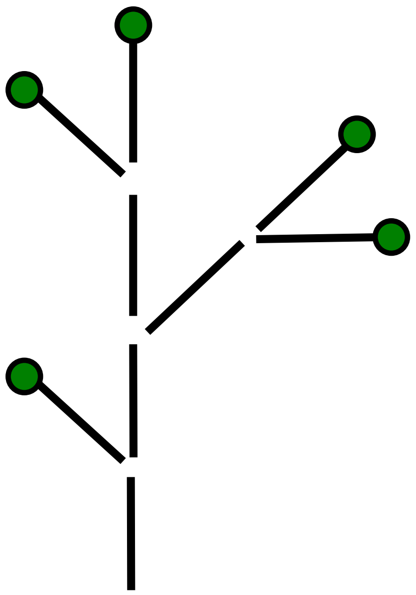
◌ = [Root]



Procedural Modeling: Grammars

- Useful for creating plants

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge



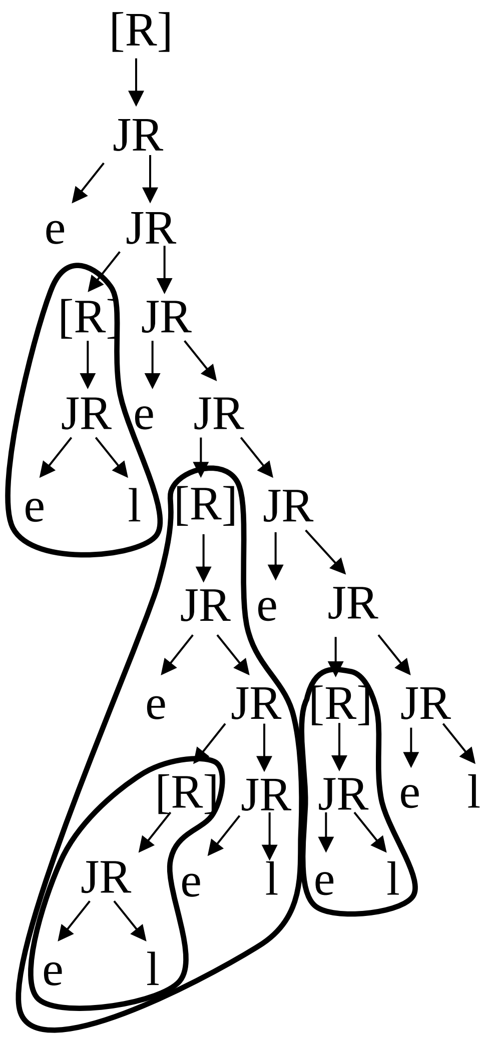
[e[el]e[e[el]el]e[el]el]

● = leaf

| = edge

● = Junction

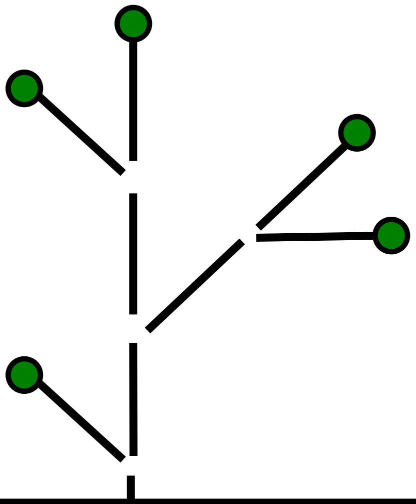
◌ = [Root]



Procedural Modeling: Grammars

- Useful for creating plants

Start → [Root]
Root → Junction Root | leaf
Junction → [Root] | edge



● = leaf

| = edge

● = Junction

◌ = [Root]

As with Wang tiles, the ability to make a choice creates a variety and removes periodicity.

Procedural Modeling: Grammars

- Useful for creating plants

Start → [End]
End → Branch End | leaf
Branch → [End] | edge

🟢 = leaf

| = edge

🔵 = Branch

◌ = [End]

Procedural Modeling: Grammars

- Useful for creating plants

[E]

Start → [End]
End → Branch End | leaf
Branch → [End] | edge

● = leaf
| = edge
● = Branch
◌ = [End]

Procedural Modeling: Grammars

- Useful for creating plants

$$\begin{tikzcd}[E] \arrow[d] \\ BE\end{tikzcd}$$

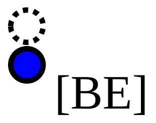
Start → [End]		
End → Branch End	|	leaf
Branch → [End]	|	edge

● = leaf

| = edge

● = Branch

◌ = [End]



Procedural Modeling: Grammars

- Useful for creating plants

Start → [End]
End → Branch End | leaf
Branch → [End] | edge

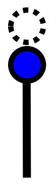
$$\begin{tikzcd}
 & {[E]} \arrow[d] \\
 & BE \arrow[dl] \arrow[d] \\
e & BE
\end{tikzcd}$$

● = leaf

| = edge

● = Branch

◌ = [End]



Procedural Modeling: Grammars

- Useful for creating plants

Start → [End]
End → Branch End | leaf
Branch → [End] | edge

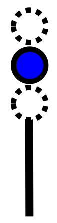
$$\begin{tikzcd}
 & \text{[E]} \arrow[d] \\
 & \text{BE} \arrow[dl] \arrow[d] \\
\text{e} & \text{BE} \arrow[dl] \arrow[d] \\
\text{[E]} & \text{BE}
\end{tikzcd}$$

● = leaf

| = edge

● = Branch

◌ = [End]


[e[E]BE]

Procedural Modeling: Grammars

- Useful for creating plants

Start → [End]
End → Branch End | leaf
Branch → [End] | edge

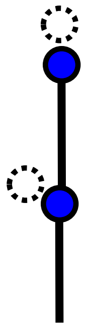
$$\begin{tikzcd}[column sep=tiny]
 & & \text{[E]} \arrow[d] & \\
 & & \text{BE} \arrow[dl] \arrow[d] & \\
 & \text{e} & \text{BE} \arrow[dl] \arrow[d] & \\
 & \text{[E]} \arrow[d] & \text{BE} \arrow[d] \arrow[dr] & \\
 & \text{BE} & \text{e} & \text{BE}
\end{tikzcd}$$

● = leaf

| = edge

● = Branch

◌ = [End]



[e[BE]eBE]

Procedural Modeling: Grammars

- Useful for creating plants

Start → [End]
End → Branch End | leaf
Branch → [End] | edge

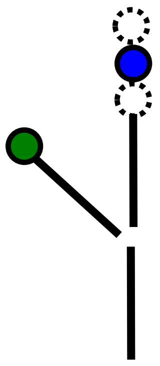
$$\begin{tikzcd}
 & & {[R]} \arrow[d] & & \\
 & & BE \arrow[dl] \arrow[d] & & \\
 & e & BE \arrow[dl] \arrow[d] & & \\
 & {[E]} \arrow[d] & BE \arrow[d] \arrow[dr] & & \\
 & BE \arrow[dl] \arrow[dr] & e & BE \arrow[d] \arrow[dr] & \\
e & & l & {[E]} & BE
\end{tikzcd}$$

🟢 = leaf

| = edge

🔵 = Branch

◌ = [End]


[e[el]e[E]BE]

Procedural Modeling: Grammars

- Useful for creating plants

Start → [End]		
End → Branch End	|	leaf
Branch → [End]	|	edge

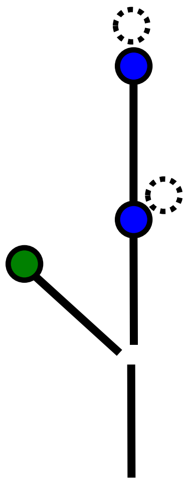
$$\begin{tikzcd}
 & & {[E]} \arrow[d] & & & & & \\
 & & BE \arrow[dl] \arrow[dr] & & & & & \\
 & e & & BE \arrow[dl] \arrow[dr] & & & & \\
 & & {[E]} \arrow[d] & & BE \arrow[d] \arrow[dr] & & & \\
 & & BE \arrow[dl] \arrow[dr] & & e & BE \arrow[d] \arrow[dr] & & \\
 & e & & l & & {[E]} \arrow[d] & BE \arrow[d] \arrow[dr] & \\
 & & & & & BE & e & BE
\end{tikzcd}$$

● = leaf

| = edge

● = Branch

◌ = [End]



[e[el]e[BE]eBE]

Procedural Modeling: Grammars

- Useful for creating plants

Start → [End]		
End → Branch End	|	leaf
Branch → [End]	|	edge

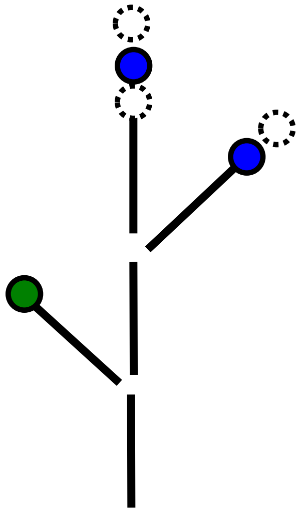
$$\begin{tikzcd}
 & & & {[E]} \arrow[d] & & & & \\
 & & & BE \arrow[dl] \arrow[d] & & & & \\
 & & e & BE \arrow[dl] \arrow[d] \arrow[dr] & & & & \\
 & & {[E]} \arrow[d] & & & & & \\
 & & BE \arrow[dl] \arrow[dr] & e & BE \arrow[d] \arrow[dr] & & & \\
 & e & & l & {[E]} \arrow[d] & BE \arrow[d] \arrow[dr] & & \\
 & & & & BE \arrow[dl] \arrow[dr] & e & BE \arrow[d] \arrow[dr] & \\
 & & & e & & BE & {[E]} & BE
\end{tikzcd}$$

🟢 = leaf

| = edge

🔵 = Branch

◌ = [End]

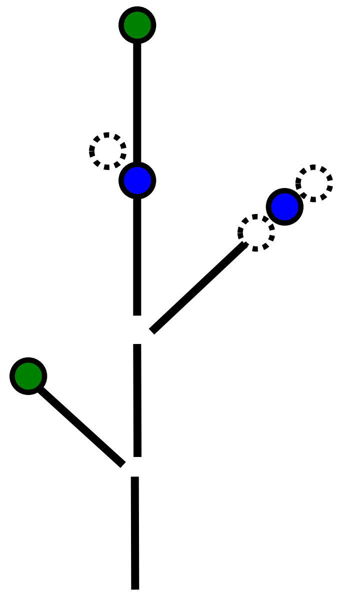


[e[el]e[eBE]e[E]BE]

Procedural Modeling: Grammars

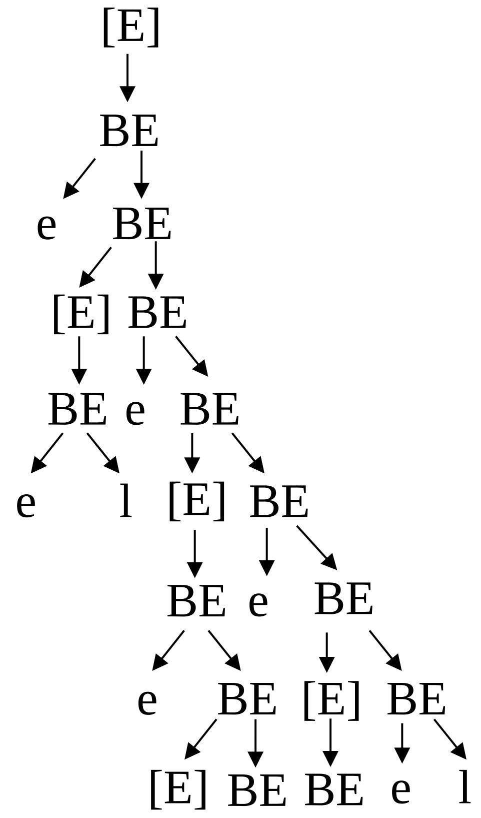
- Useful for creating plants

Start → [End]
End → Branch End | leaf
Branch → [End] | edge



[e[el]e[e[E]BE]e[BE]el]

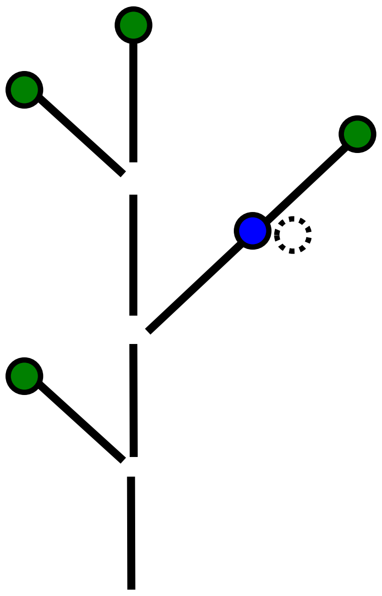
🟢 = leaf
| = edge
🔵 = Branch
◌ = [End]



Procedural Modeling: Grammars

- Useful for creating plants

Start → [End]
End → Branch End | leaf
Branch → [End] | edge



● = leaf

| = edge

● = Branch

◌ = [End]

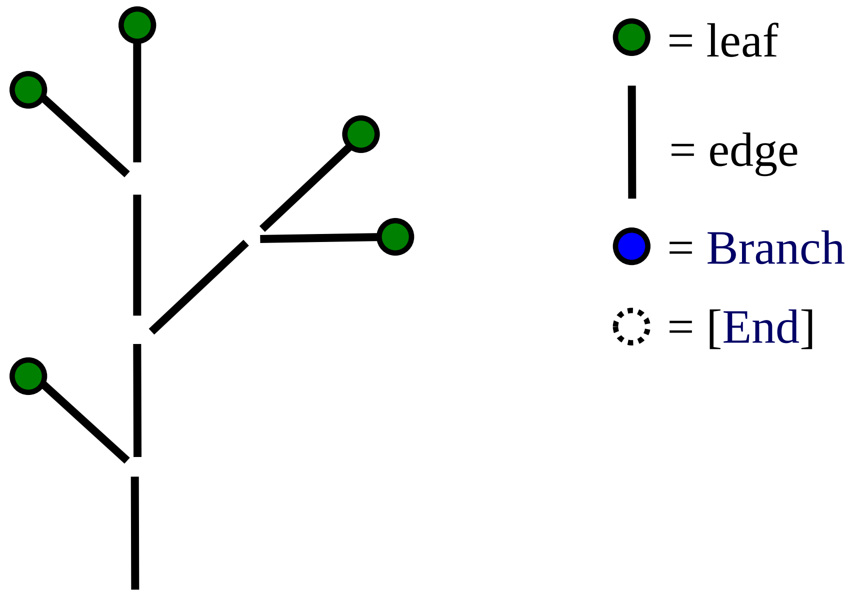
[e[el]e[e[BE]el]e[el]el]

$$\begin{tikzcd}
 & & & {[E]} \arrow[d] & & & & & \\
 & & & BE \arrow[dl] \arrow[d] & & & & & \\
 & & e & BE \arrow[dl] \arrow[d] & & & & & \\
 & & {[E]} \arrow[d] & BE \arrow[d] \arrow[dr] & & & & & \\
 & & BE \arrow[dl] \arrow[d] & e & BE \arrow[d] \arrow[dr] & & & & \\
 & e & l & & {[E]} \arrow[d] & BE \arrow[d] \arrow[dr] & & & \\
 & & & & BE \arrow[dl] \arrow[d] & e & BE \arrow[d] \arrow[dr] & & \\
 & & & e & BE \arrow[dl] \arrow[d] & & {[E]} \arrow[d] & BE \arrow[d] \arrow[dr] & \\
 & & & {[E]} \arrow[dl] & BE \arrow[dl] \arrow[d] & & BE \arrow[d] \arrow[dr] & e & l \\
 & & BE & e & l & & e & l &
\end{tikzcd}$$

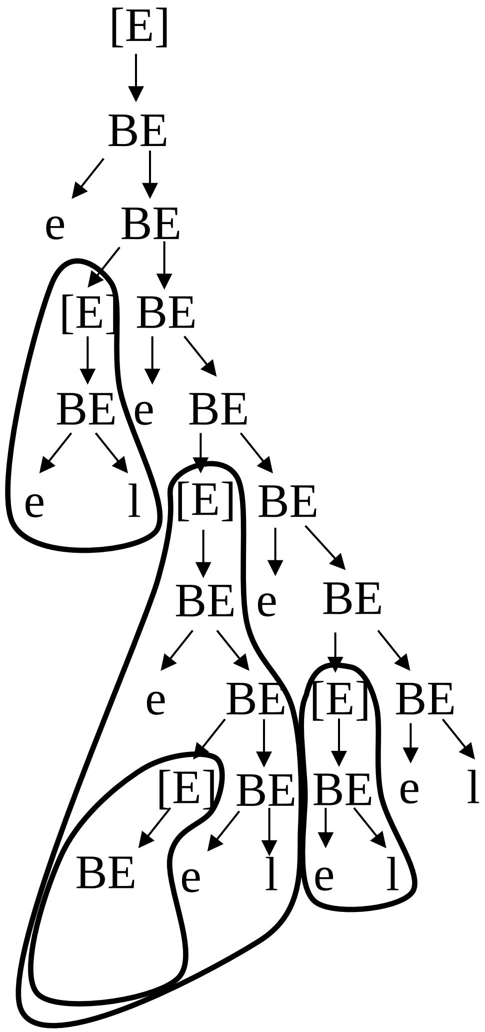
Procedural Modeling: Grammars

- Useful for creating plants

Start → [End]
End → Branch End | leaf
Branch → [End] | edge



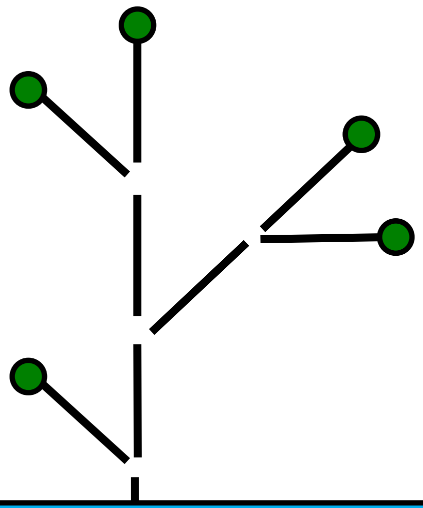
[e[el]e[e[el]el]e[el]el]



Procedural Modeling: Grammars

- Useful for creating plants

Start → [End]
End → Branch End | leaf
Branch → [End] | edge



🟢 = leaf
| = edge
🔵 = Branch
◌ = [End]

As with Wang tiles, the ability to make a choice creates a variety and removes periodicity.