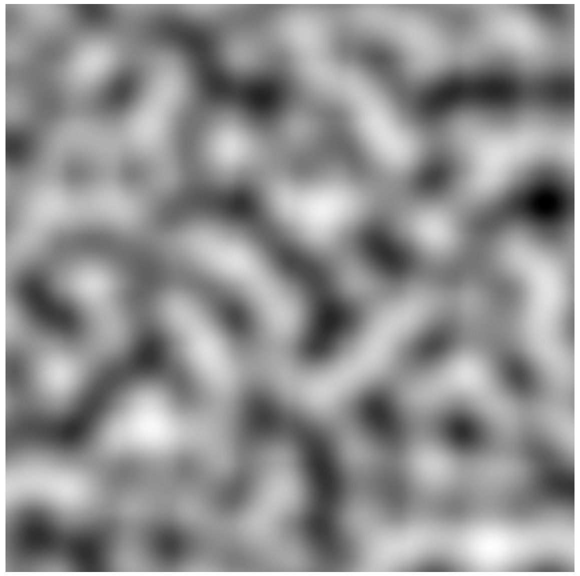
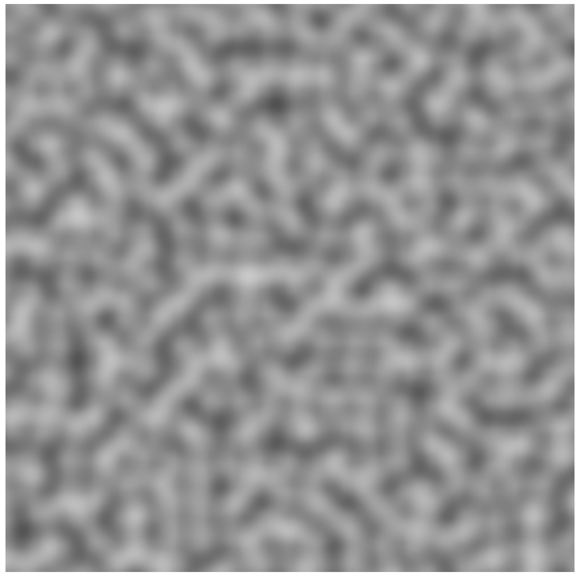
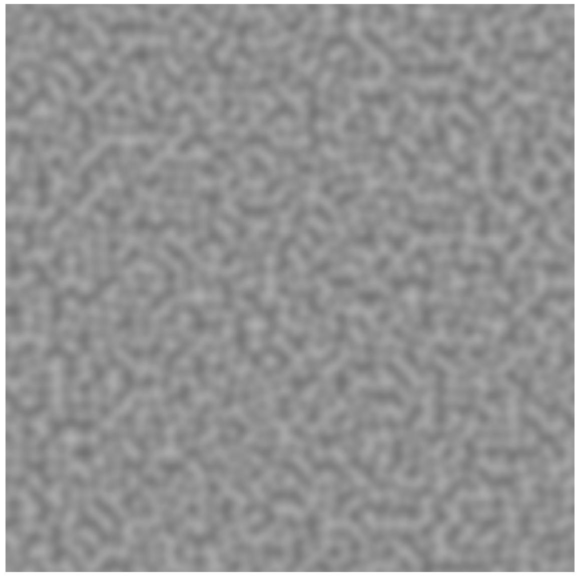
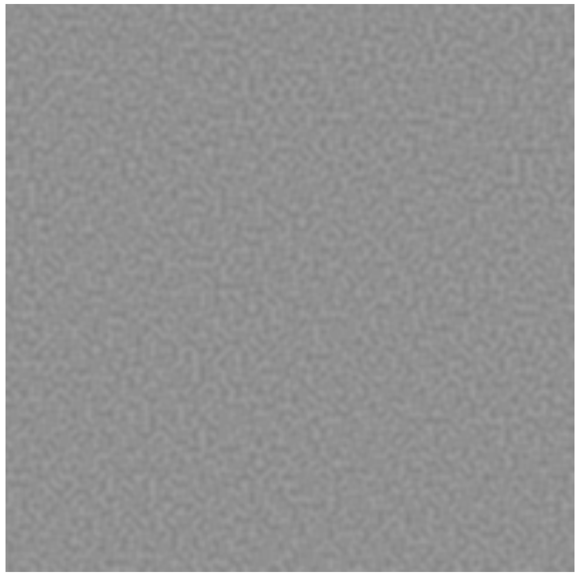
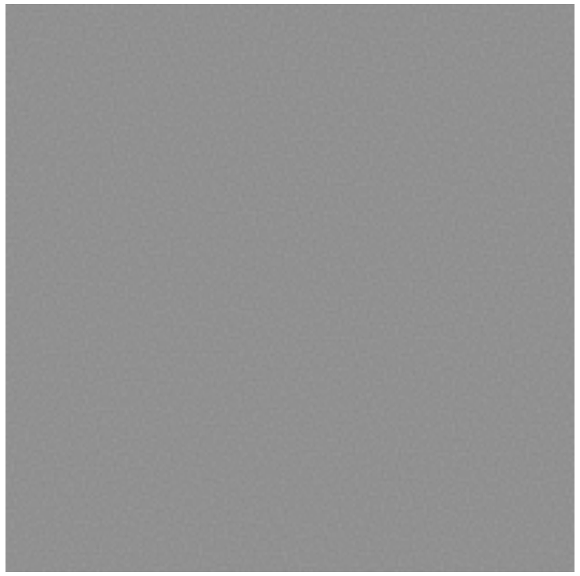
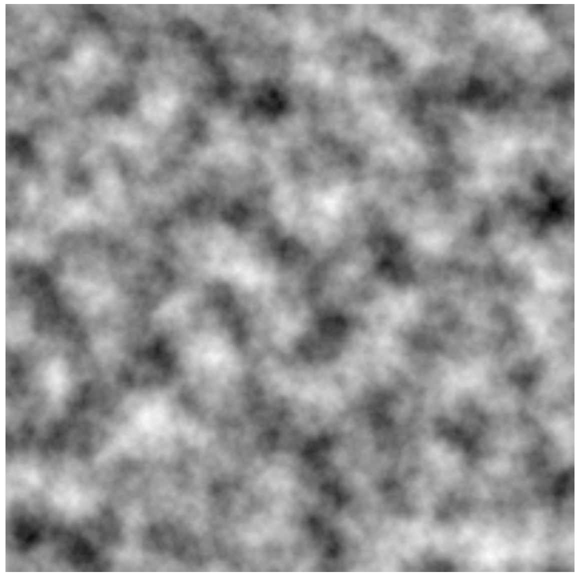
Procedural Modeling

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

Perlin Noises in 2-D

Terrain Example

Example: Seashells

- Create 3D polygonal surface models of seashells

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.

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

$$\Theta_{i+1} = \Theta_i + \Delta\Theta$$
$$\lambda_{i+1} = r_i\lambda_r$$
$$z_{i+1} = z_i\lambda_z$$

Example: Seashells

- Sweep generating curve around helico-spiral axis

Helico-spiral definition:

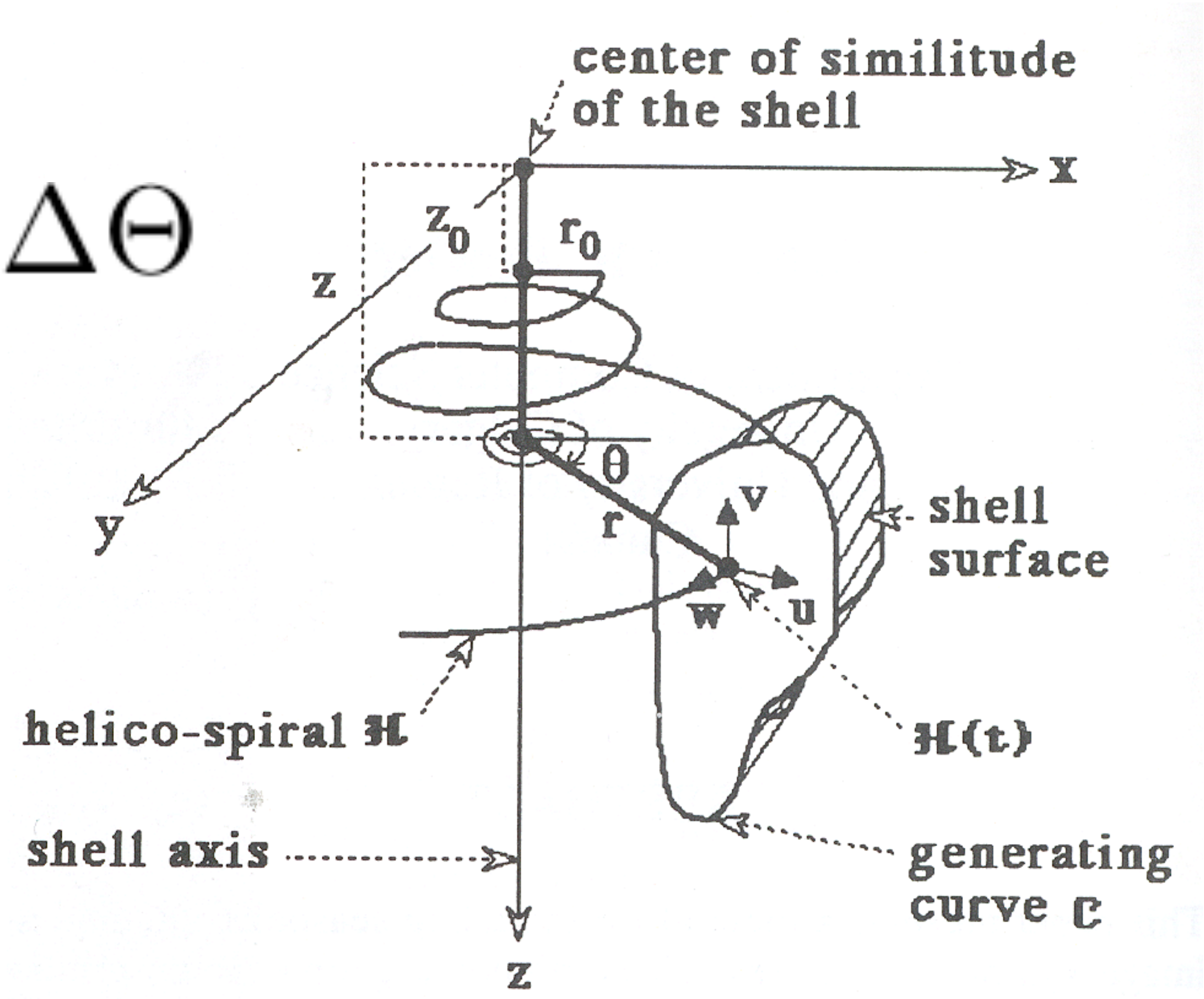
$$\Theta_{i+1} = \Theta_i + \Delta\Theta$$
$$\lambda_{i+1} = r_i \lambda_r$$
$$z_{i+1} = z_i \lambda_z$$

Example: Seashells

- Sweep generating curve around helico-spiral axis

Helico-spiral definition:

$$\Theta_{i+1} = \Theta_i + \Delta\Theta$$
$$\lambda_{i+1} = r_i \lambda_r$$
$$z_{i+1} = z_i \lambda_z$$



Fowler et al. Figure 1

Example: Seashells

- Generate different shells by varying parameters

Example: Seashells

- Generate different shells by varying parameters

Different helico-spirals

Fowler et al. Figure 2

Example: Seashells

- Generate different shells by varying parameters

Example: Seashells

- Generate different shells by varying parameters

Different generating curves

Fowler et al. Figure 3

Example: Seashells

Example: Seashells

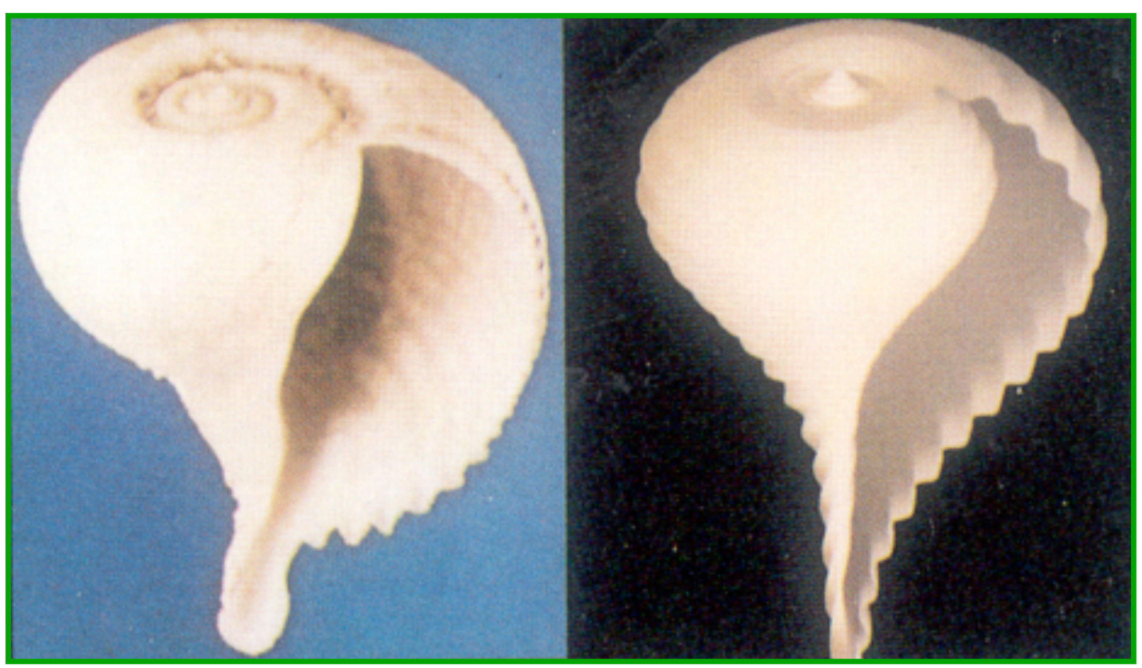
Generate many interesting shells with a simple procedural model!

Example: Seashells

Generate many interesting shells with a simple procedural model!

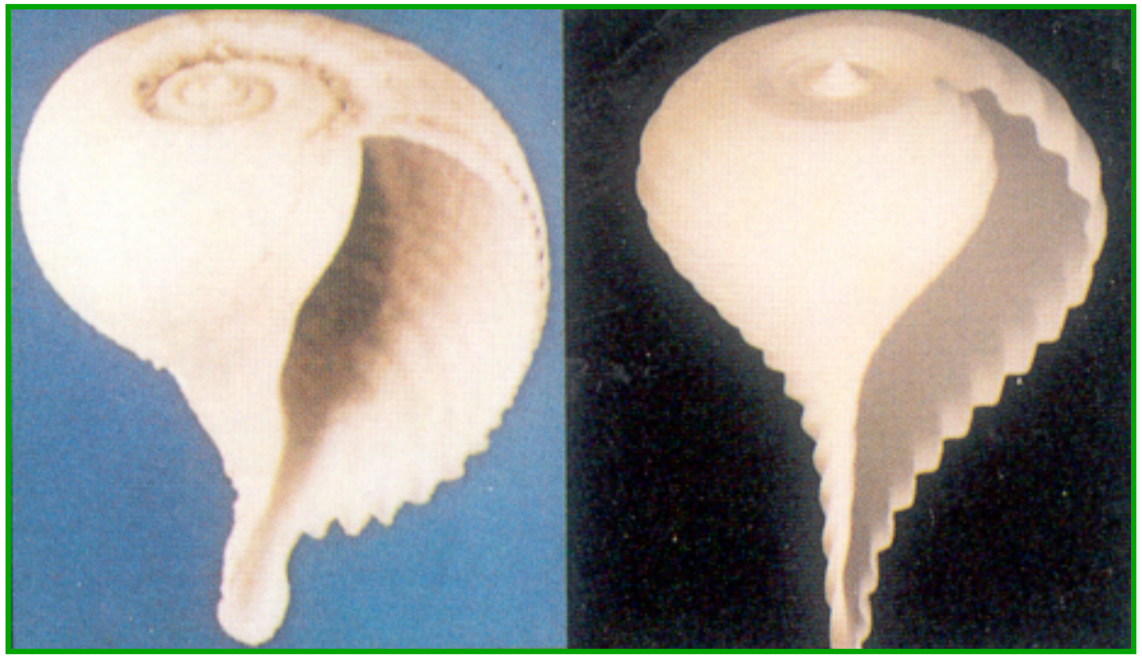
Example: Seashells

Generate many interesting shells with a simple procedural model!



Example: Seashells

Generate many interesting shells with a simple procedural model!



Fowler et al. Figures 4,5,7

Fractals

Fractals

- Useful for describing natural 3D phenomenon

Fractals

- Useful for describing natural 3D phenomenon
  - Terrain

Fractals

- Useful for describing natural 3D phenomenon
  - Terrain
  - Plants

Fractals

- Useful for describing natural 3D phenomenon
  - Terrain
  - Plants
  - Clouds

Fractals

- Useful for describing natural 3D phenomenon
  - Terrain
  - Plants
  - Clouds
  - Water

Fractals

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

Fractals

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

Fractals

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

Fractals

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

H&B Figure 10.80

Fractal Generation

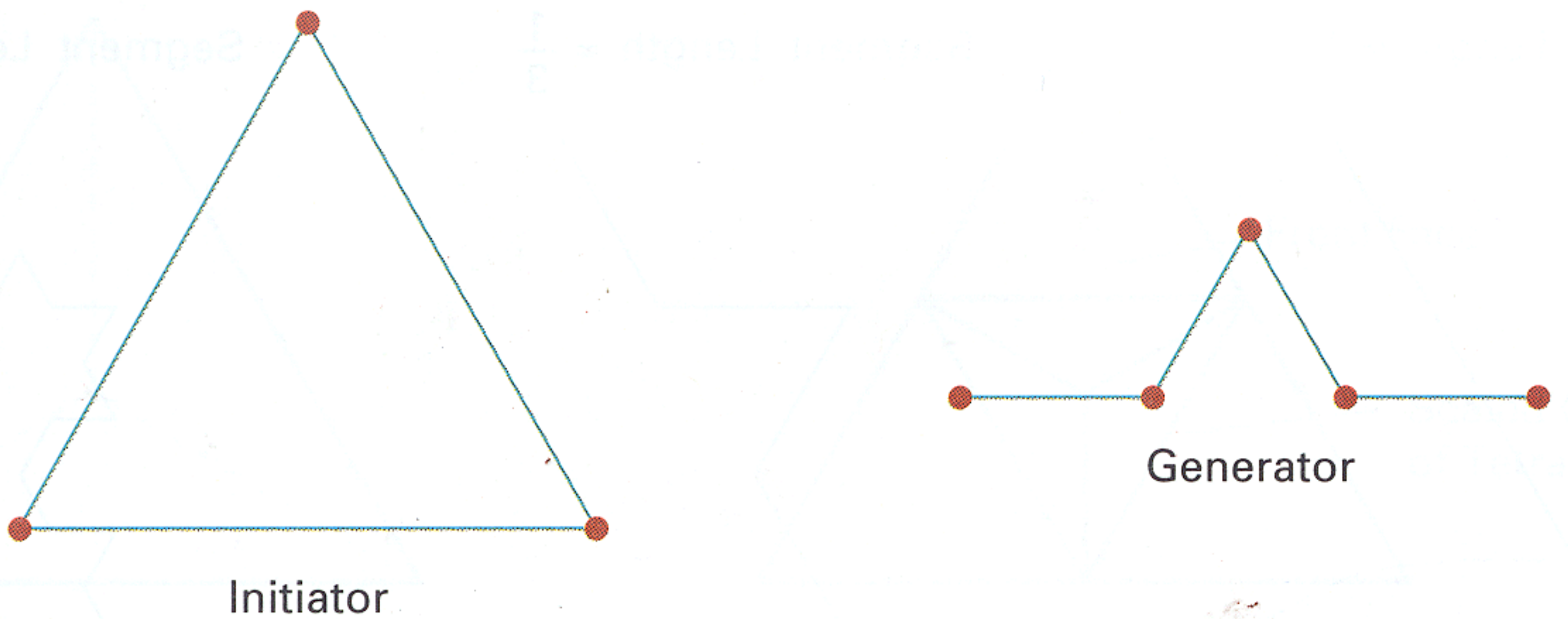
- Deterministically self-similar fractals
  - Parts are scaled copies of original
- Statistically self-similar fractals
  - Parts have same statistical properties as original

Deterministic Fractal Generation

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

Deterministic Fractal Generation

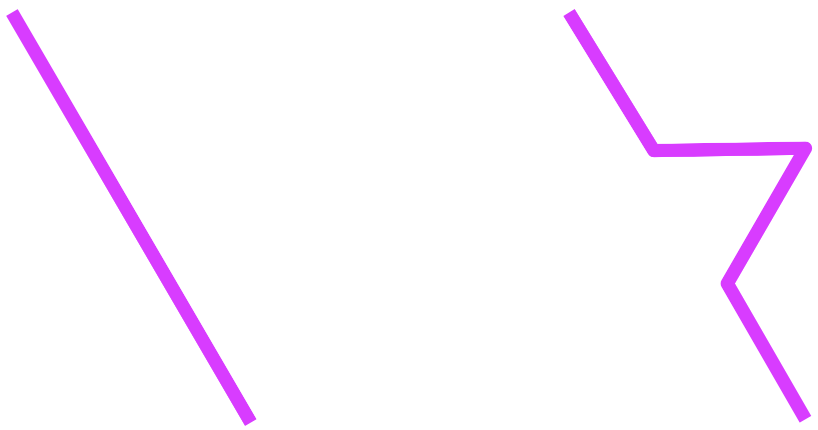
- General procedure:
  - Initiator: start with a shape
  - Generator: replace subparts with scaled copy of original



H&B Figure 10.68

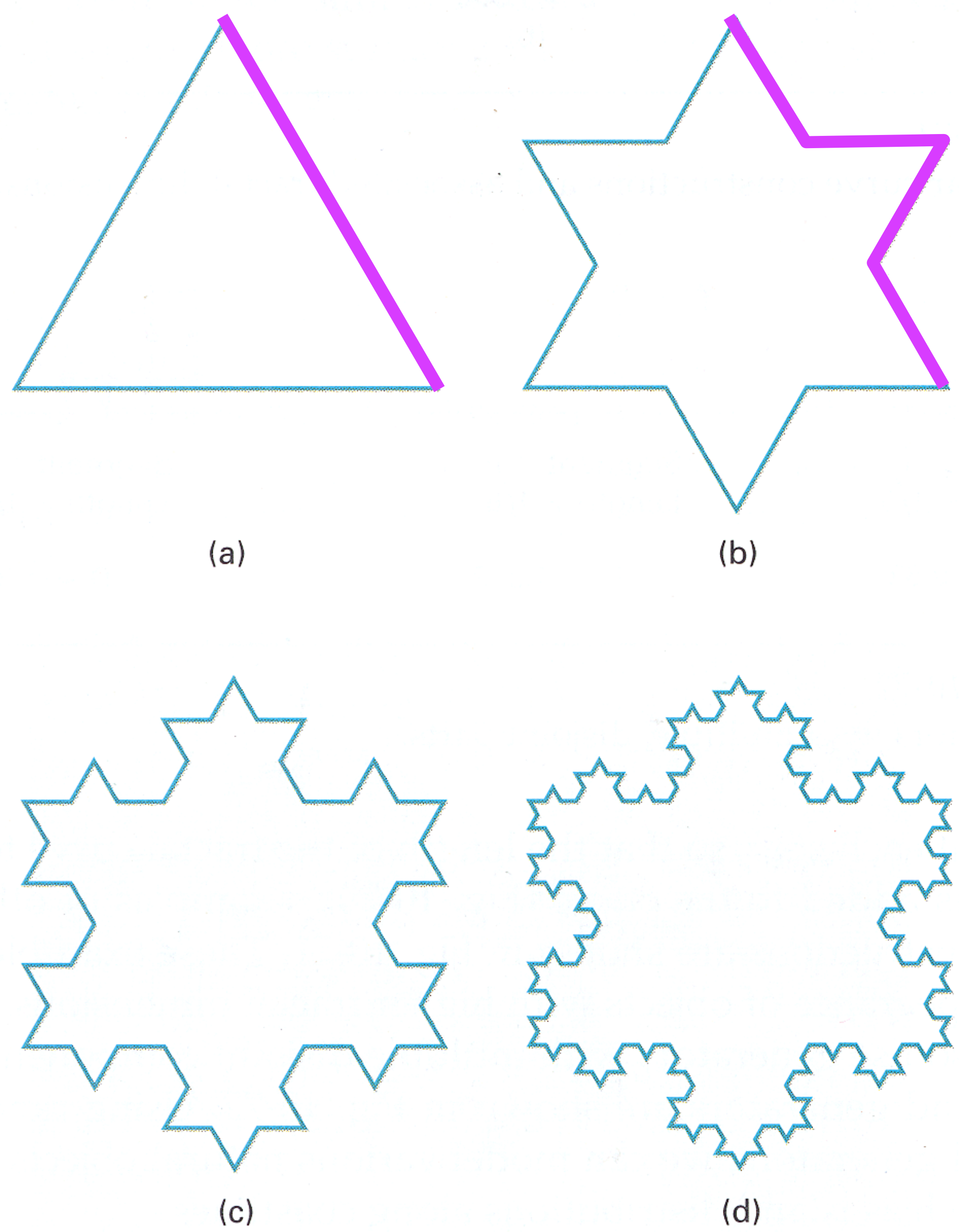
Deterministic Fractal Generation

- Apply generator repeatedly



Deterministic Fractal Generation

- Apply generator repeatedly



Koch Curve

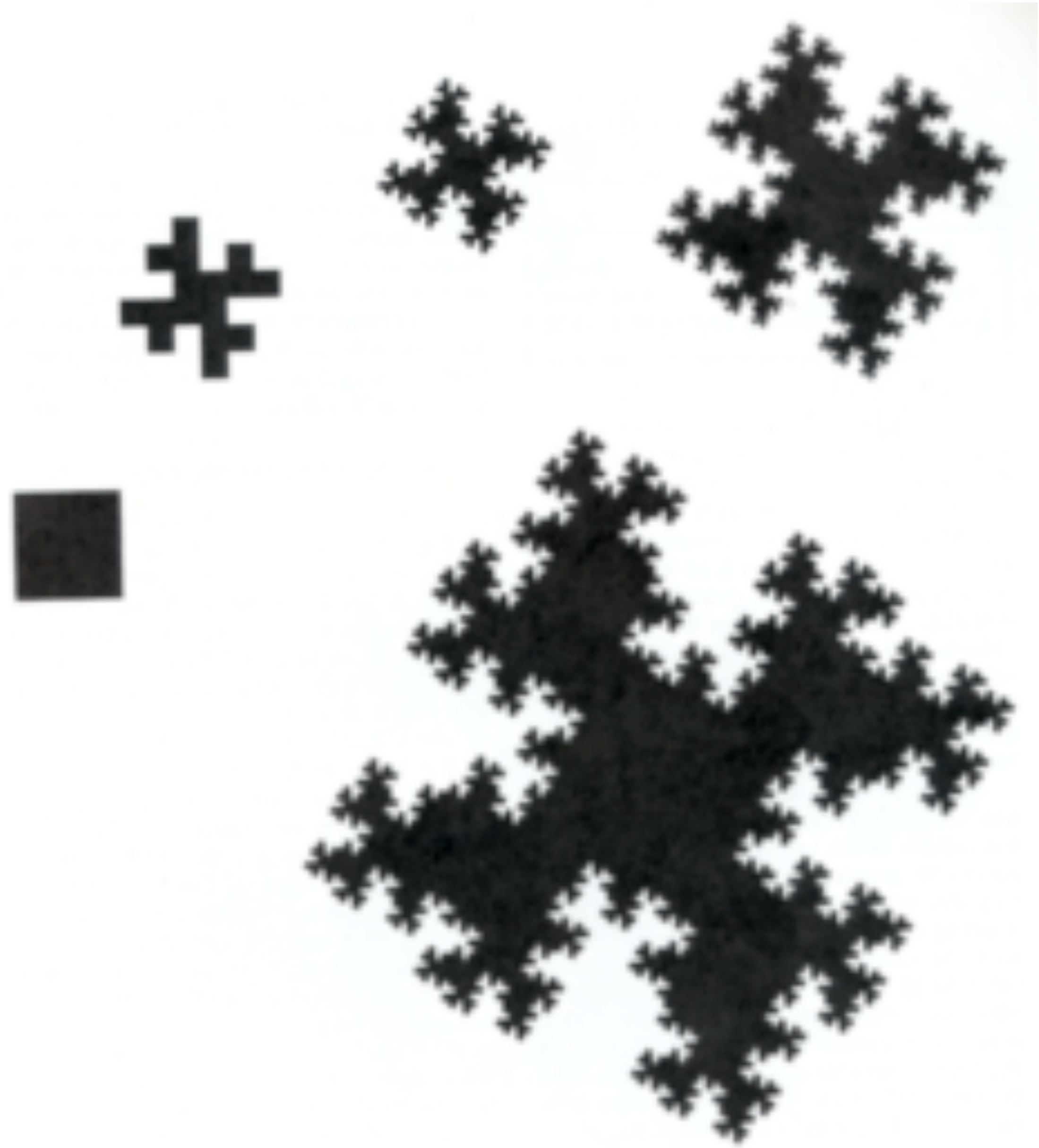
H&B Figure 10.69

Deterministic Fractal Generation

- Useful for creating interesting shapes!

Deterministic Fractal Generation

- Useful for creating interesting shapes!



Mandelbrot Figure X

Deterministic Fractal Generation

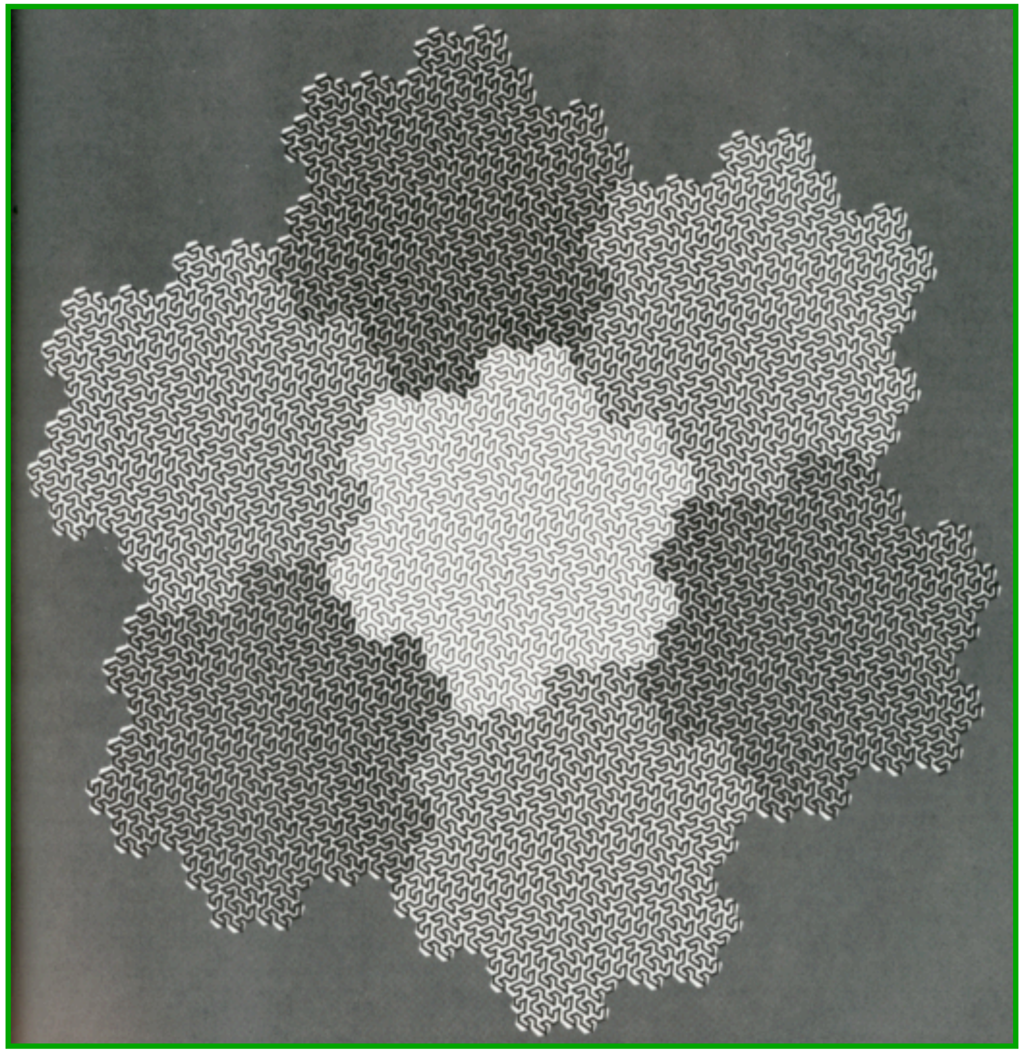
- Useful for creating interesting shapes!

Deterministic Fractal Generation

- Useful for creating interesting shapes!

Deterministic Fractal Generation

- Useful for creating interesting shapes!



Mandelbrot Figure 46

Deterministic Fractal Generation

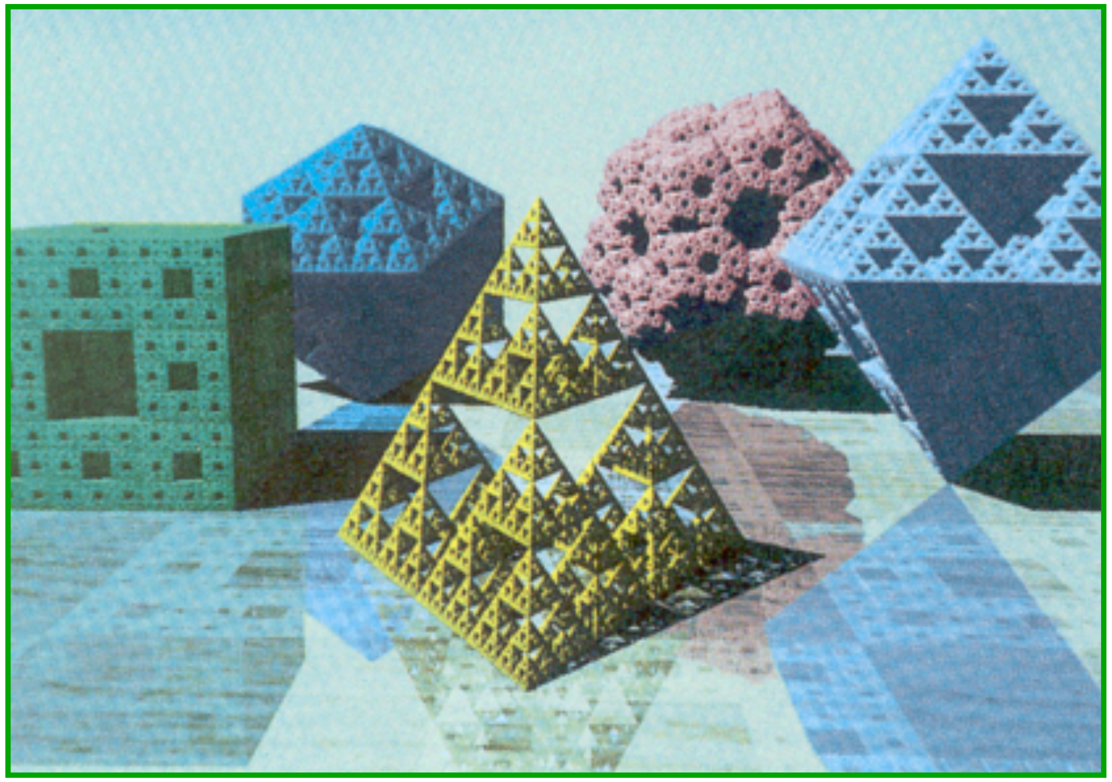
- Useful for creating interesting shapes!

Deterministic Fractal Generation

- Useful for creating interesting shapes!

Deterministic Fractal Generation

- Useful for creating interesting shapes!



H&B Figures 75 & 109

Fractal Generation

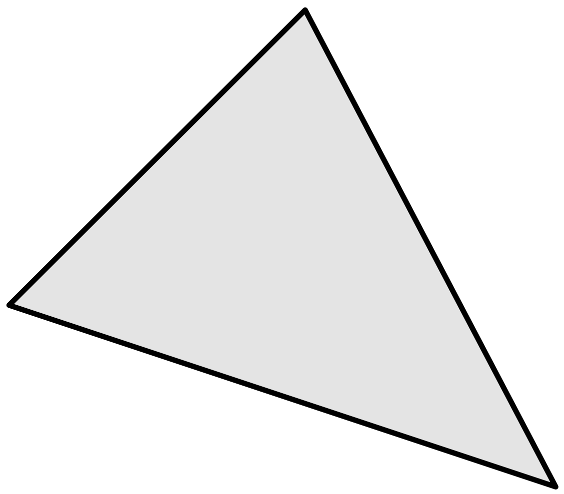
- Deterministically self-similar fractals
  - Parts are scaled copies of original
- Statistically self-similar fractals
  - Parts have same statistical properties as original

Statistical Fractal Generation

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

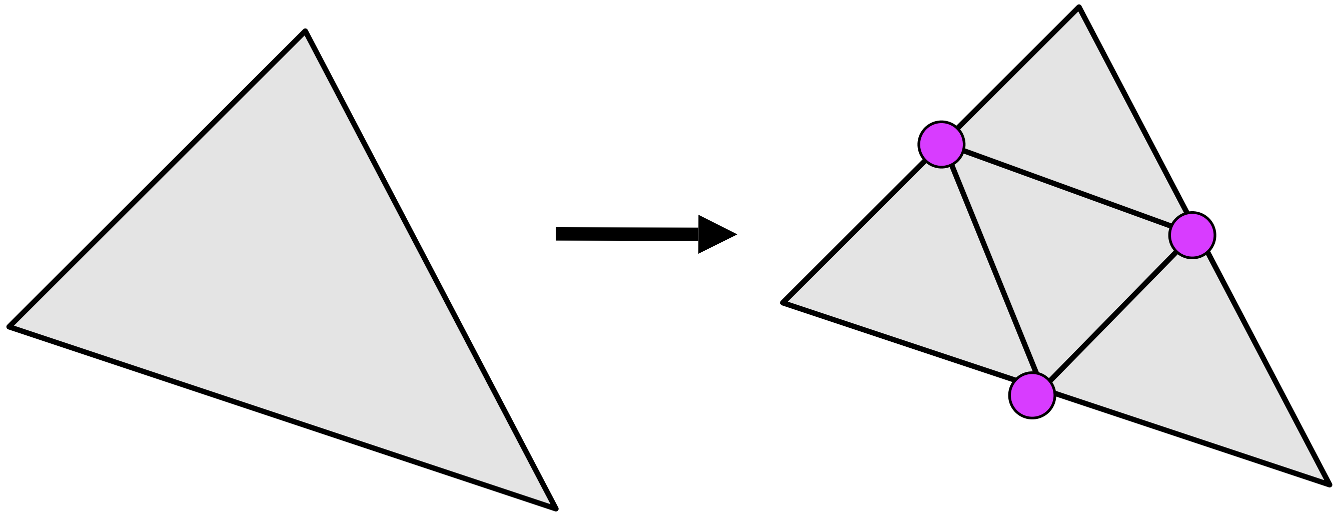
Statistical Fractal Generation

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



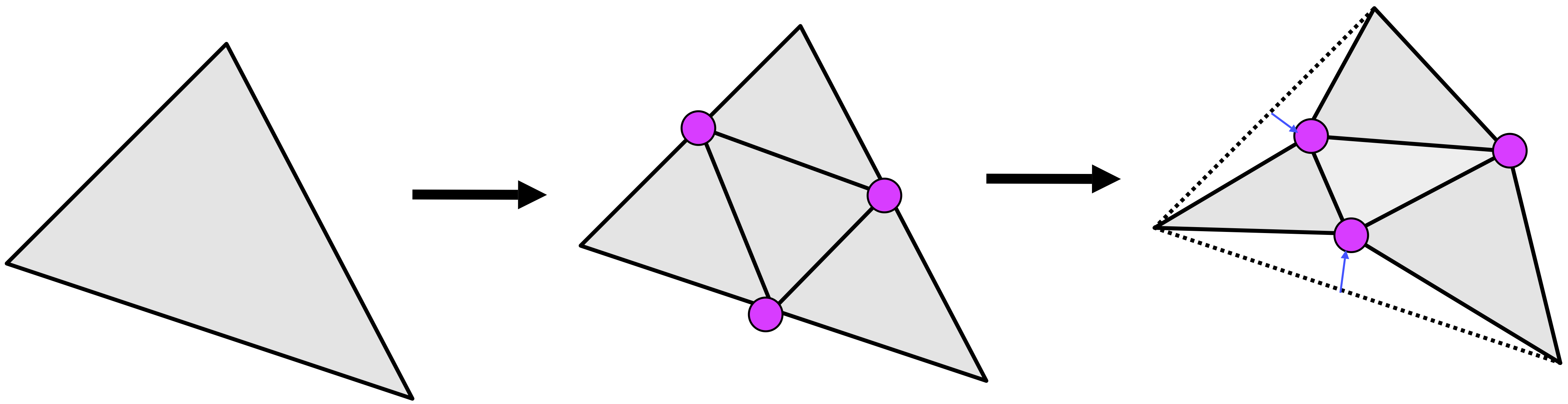
Statistical Fractal Generation

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



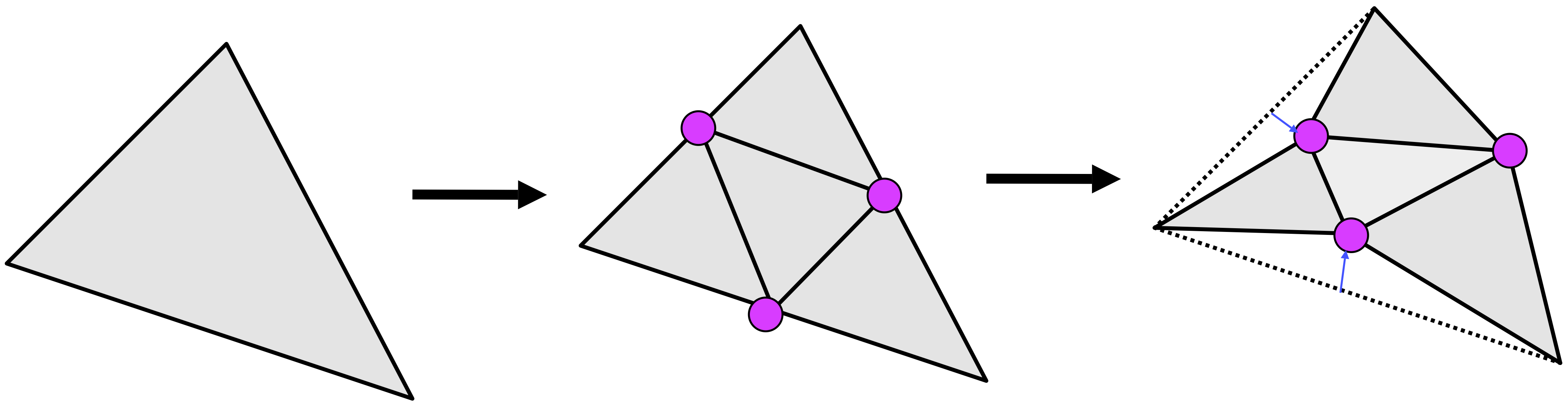
Statistical Fractal Generation

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



Statistical Fractal Generation

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



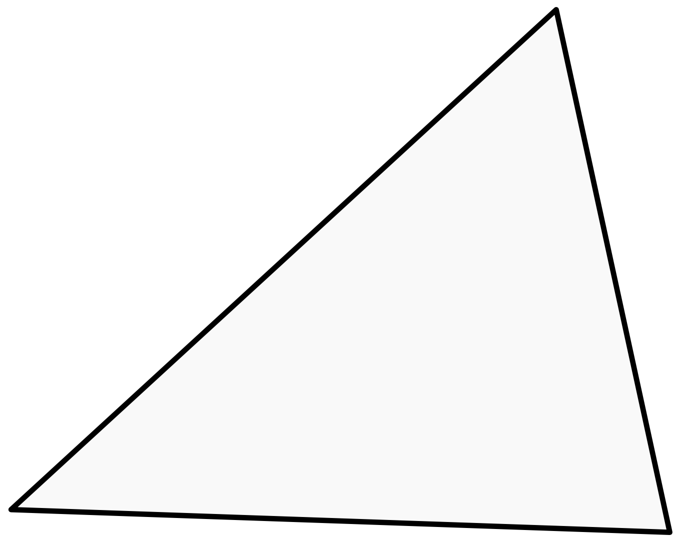
Random Midpoint Displacement

Statistical Fractal Generation

- Example: terrain

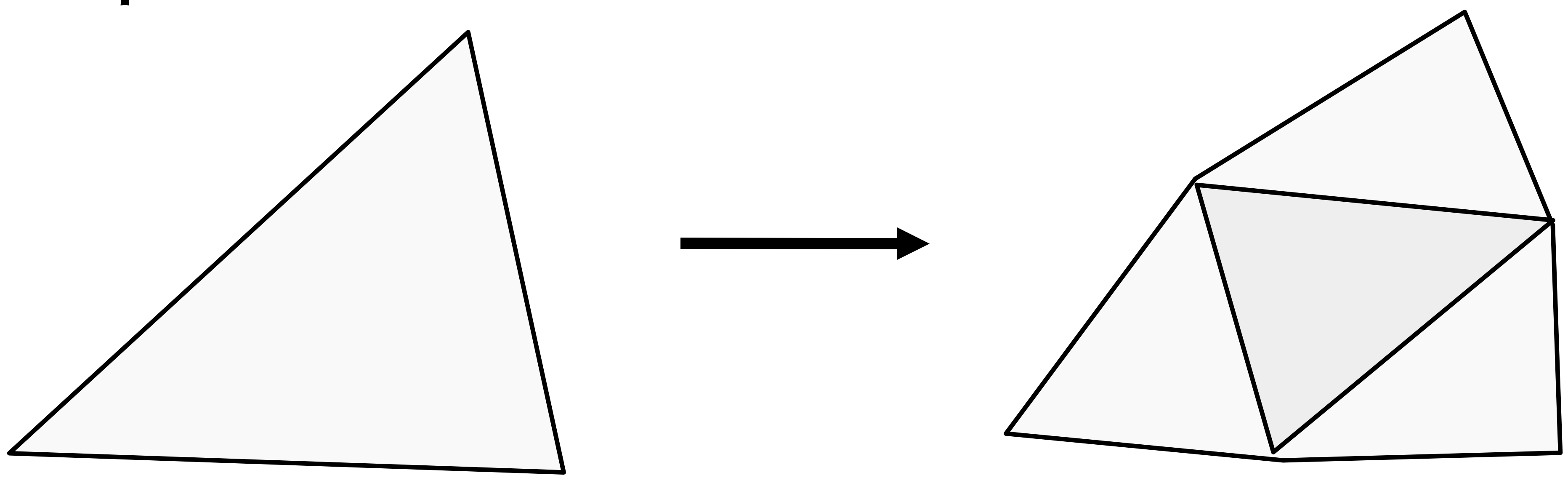
Statistical Fractal Generation

- Example: terrain



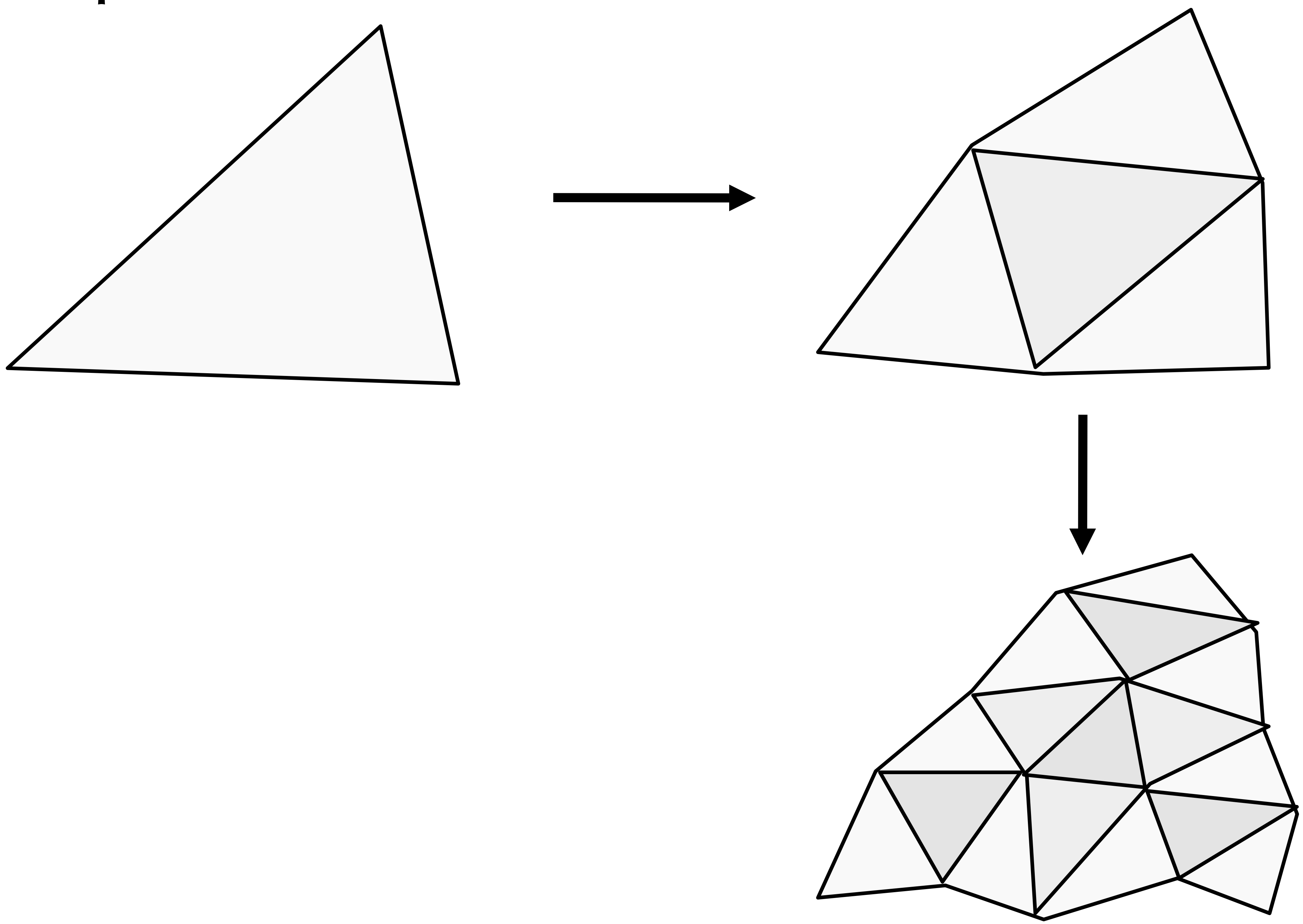
Statistical Fractal Generation

- Example: terrain



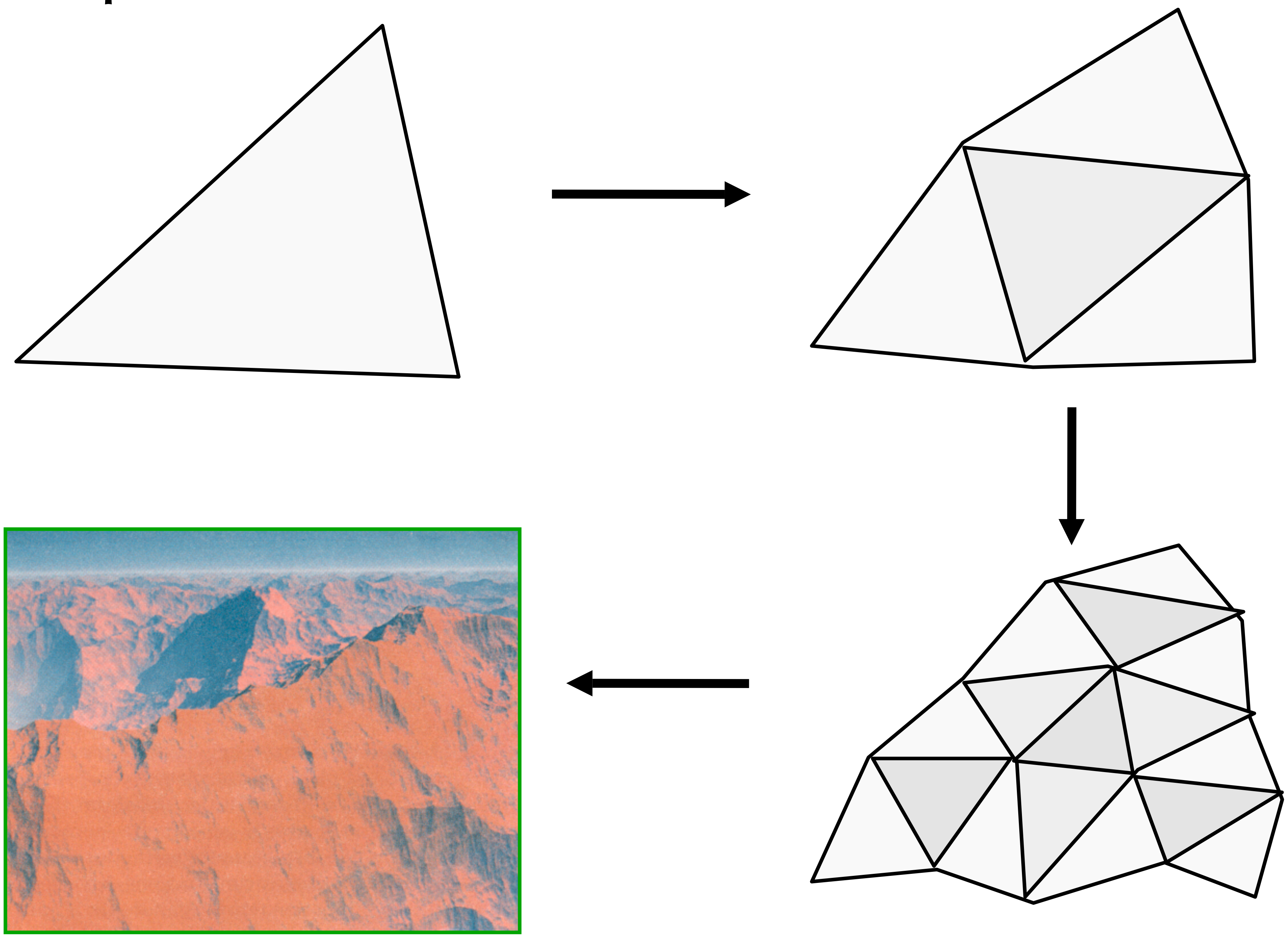
Statistical Fractal Generation

- Example: terrain



Statistical Fractal Generation

- Example: terrain



H&B Figure 10.83b

Statistical Fractal Generation

- Useful for creating mountains

Statistical Fractal Generation

- Useful for creating mountains

H&B Figure 10.83a

Statistical Fractal Generation

- Useful for creating 3D plants

Statistical Fractal Generation

- Useful for creating 3D plants

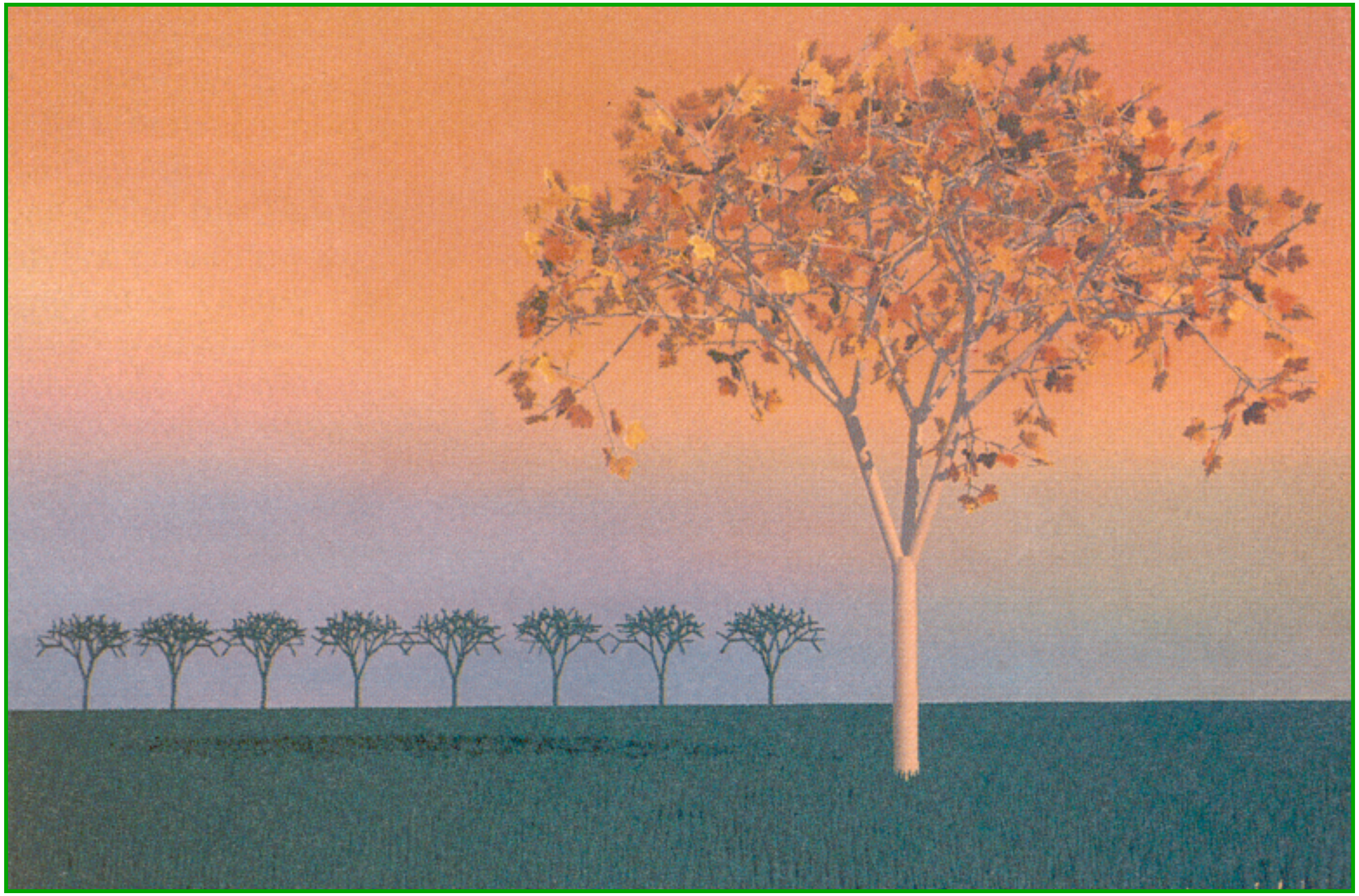
H&B Figure 10.82

Statistical Fractal Generation

- Useful for creating 3D plants

Statistical Fractal Generation

- Useful for creating 3D plants



H&B Figure 10.79

L-Systems

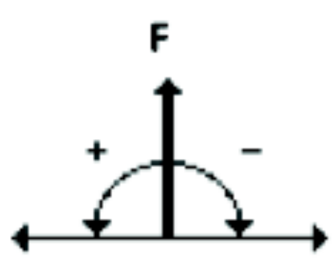
- Developed by Aristid Lindenmayer to model the development of plants
- Based on parallel string-rewriting rules
- Excellent for modeling organic objects and fractals

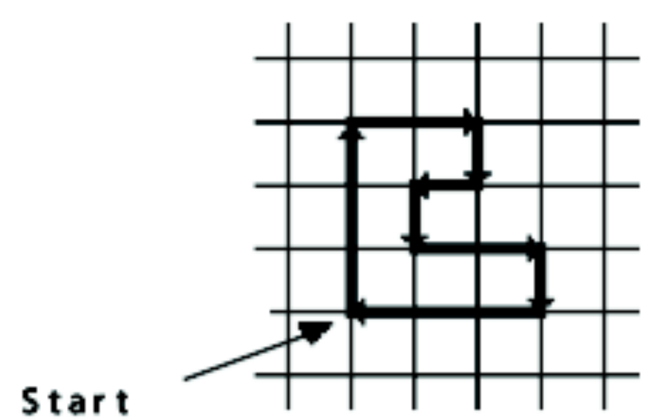
L-Systems Grammar

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

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 ∂

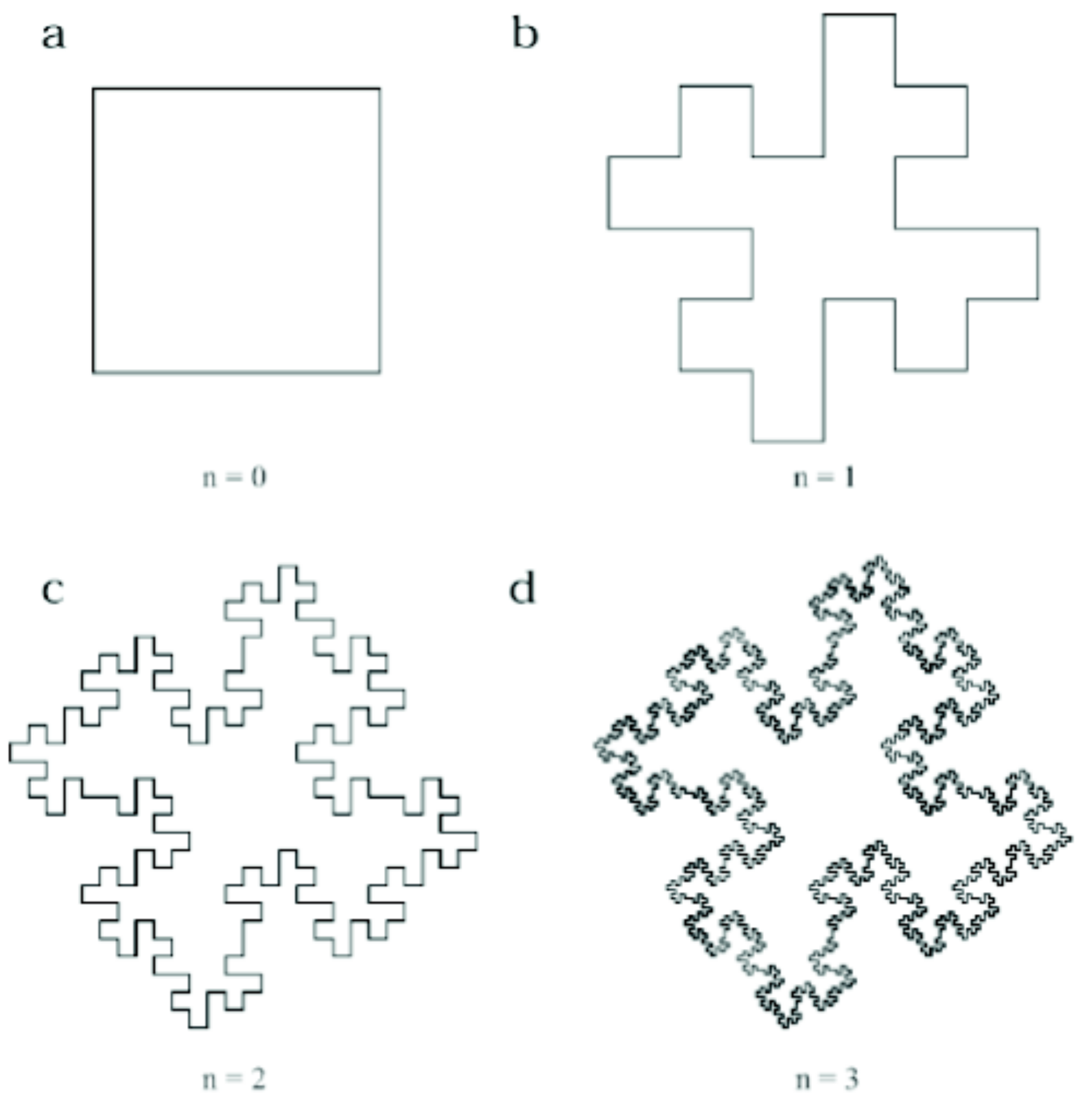




FFF-FF-F-F+F+FF-F-FFF

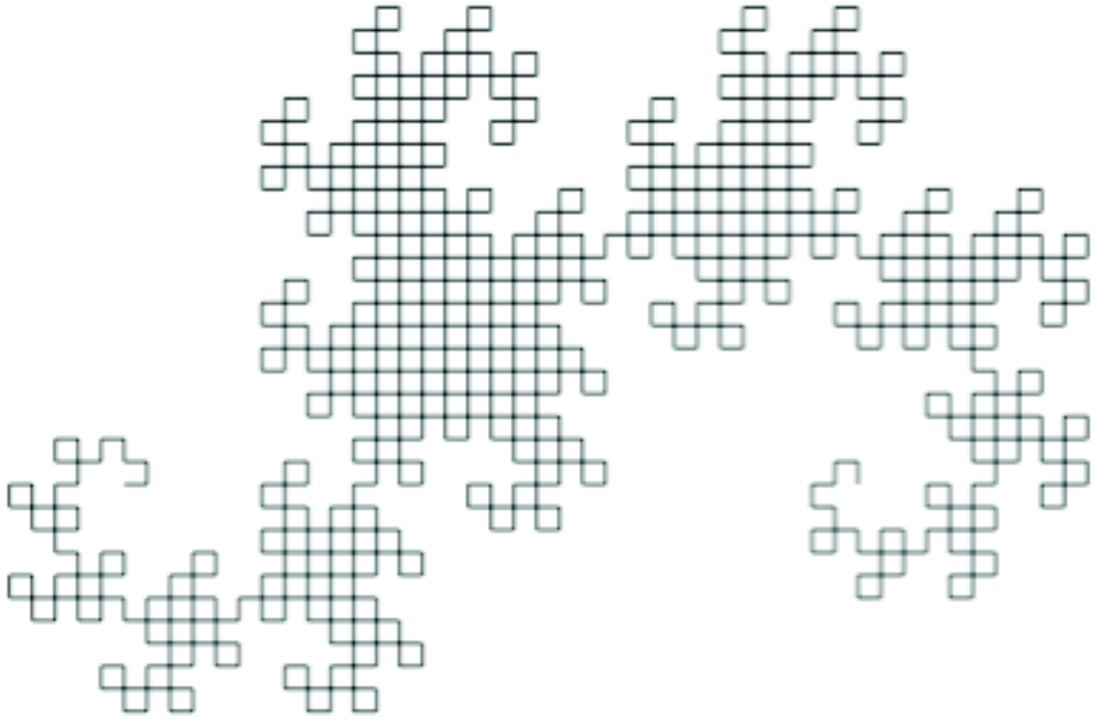
L-Systems Example: Koch Snowflake

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

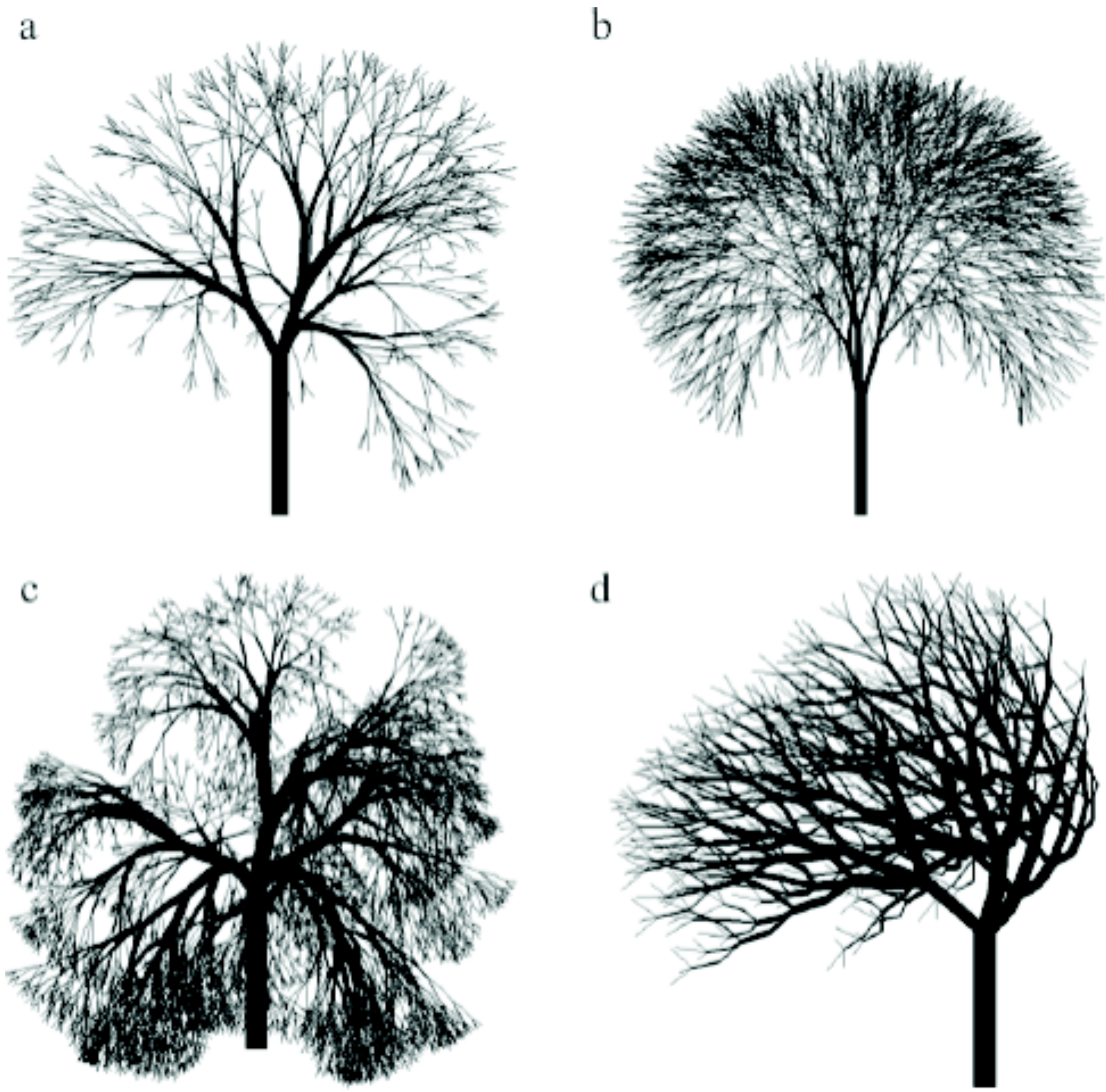


L-Systems Example: Dragon Curve

- Axiom: F_l ∂ :90 degrees n:10 iterations
- F_l -> F_l+F_r+
- F_r -> F_l-F_r-



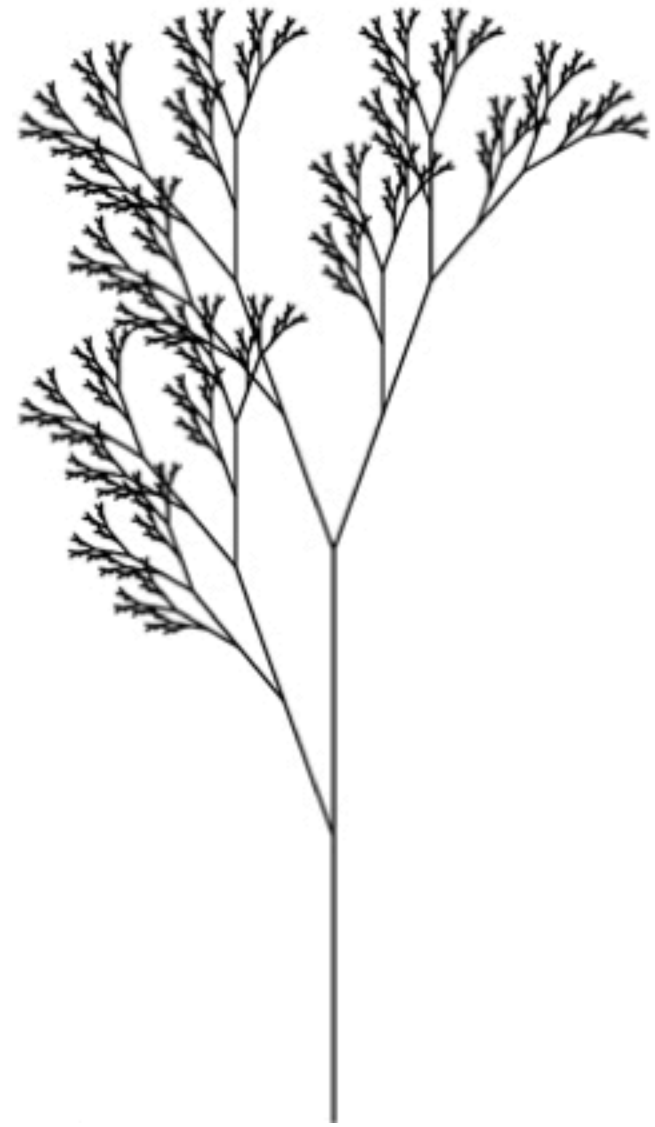
L-Systems for Plants



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

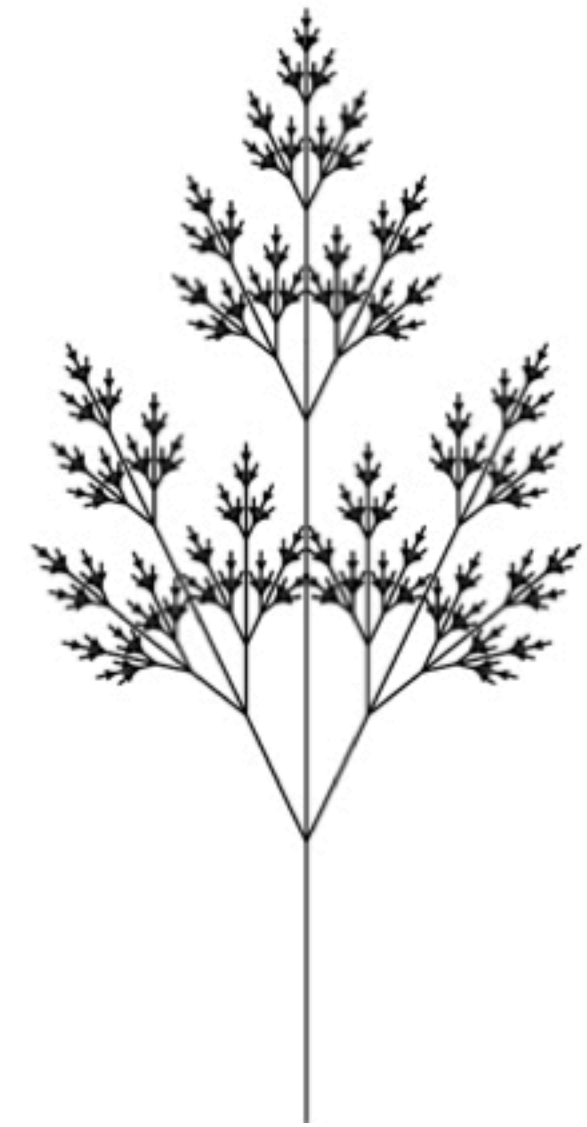
L-system

- alphabet: {a,b}
- initiator: a
- production rules:
  - a -> b
  - b -> ba
- generations:
  - a
  - b
  - ba
  - bab
  - babba
  - babbabab
  - babbababbabba
  - babbababbabbababbabab



d

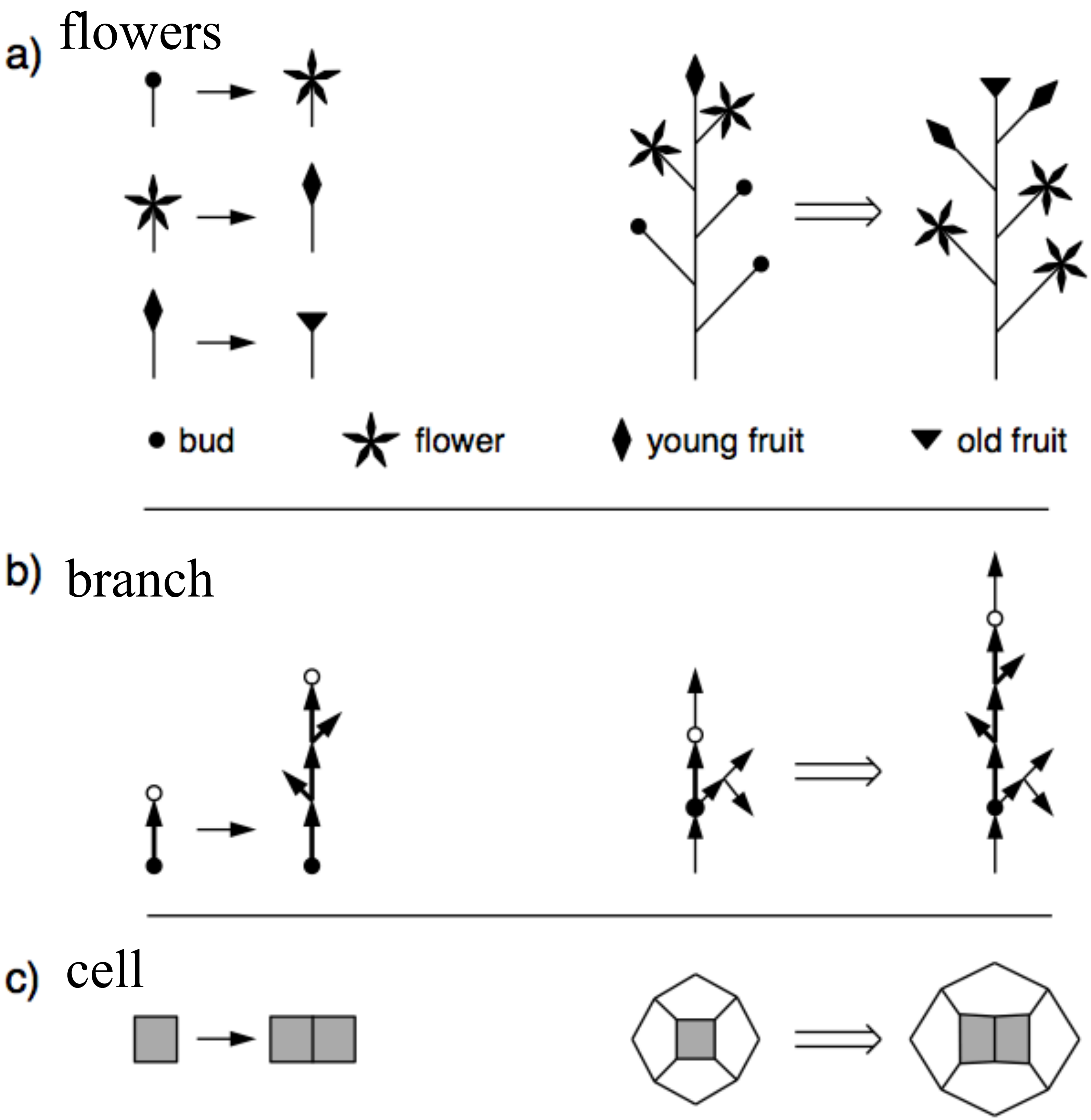
n=7, δ=20°
X
X $\rightarrow$ F[+X]F[-X]+X
F $\rightarrow$ FF



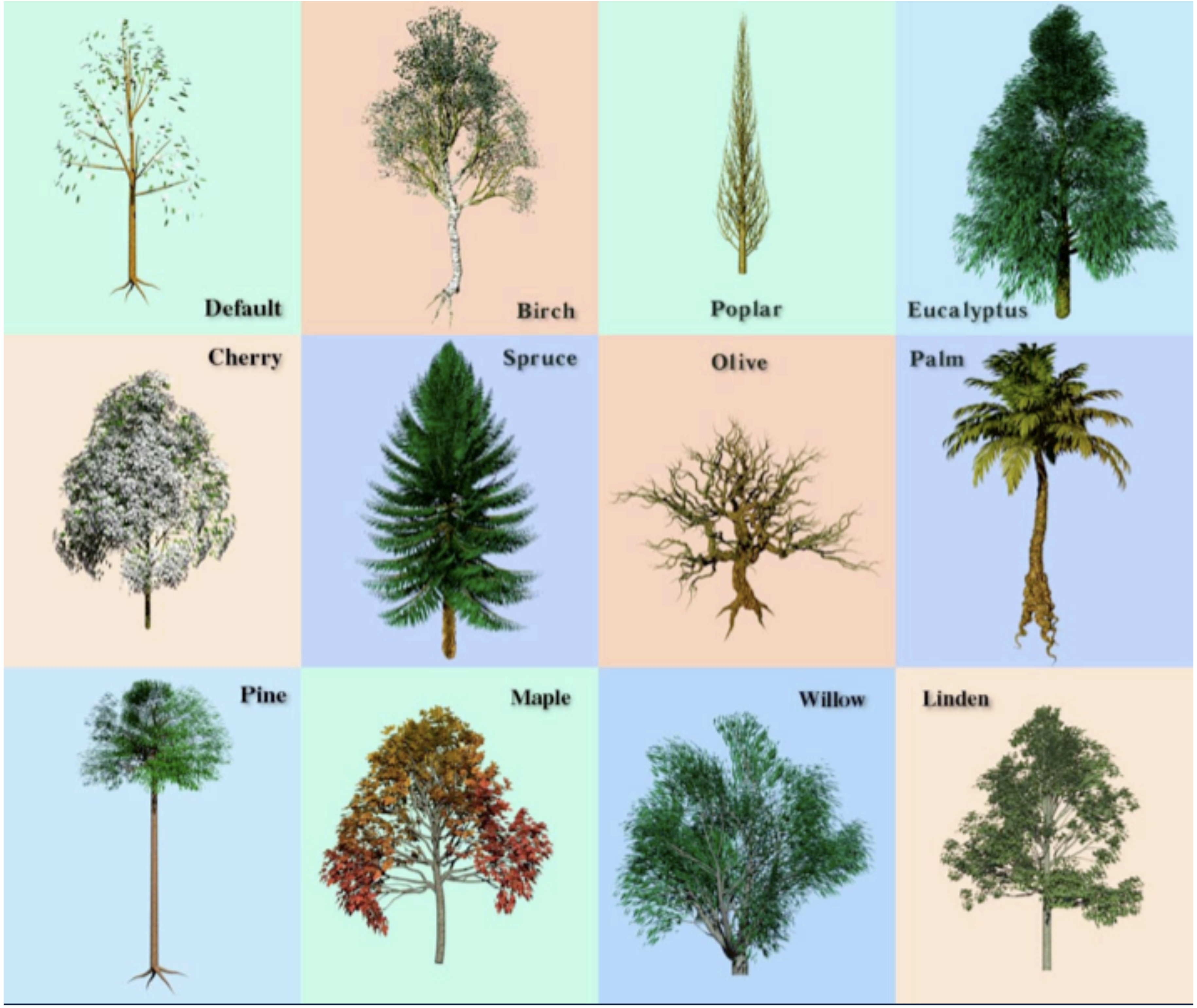
e

n=7, δ=25.7°
X
X $\rightarrow$ F[+X][-X]FX
F $\rightarrow$ FF

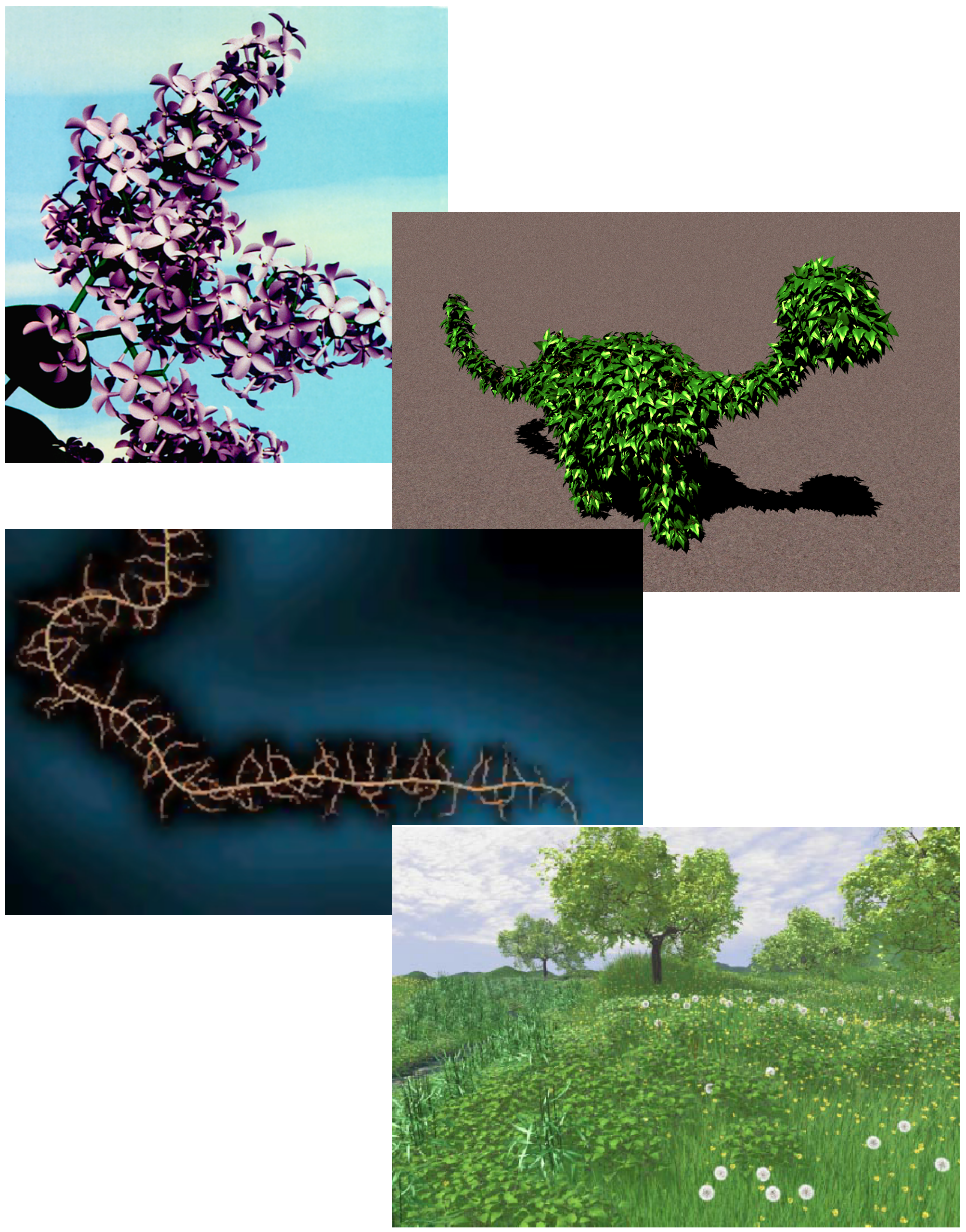
L-system



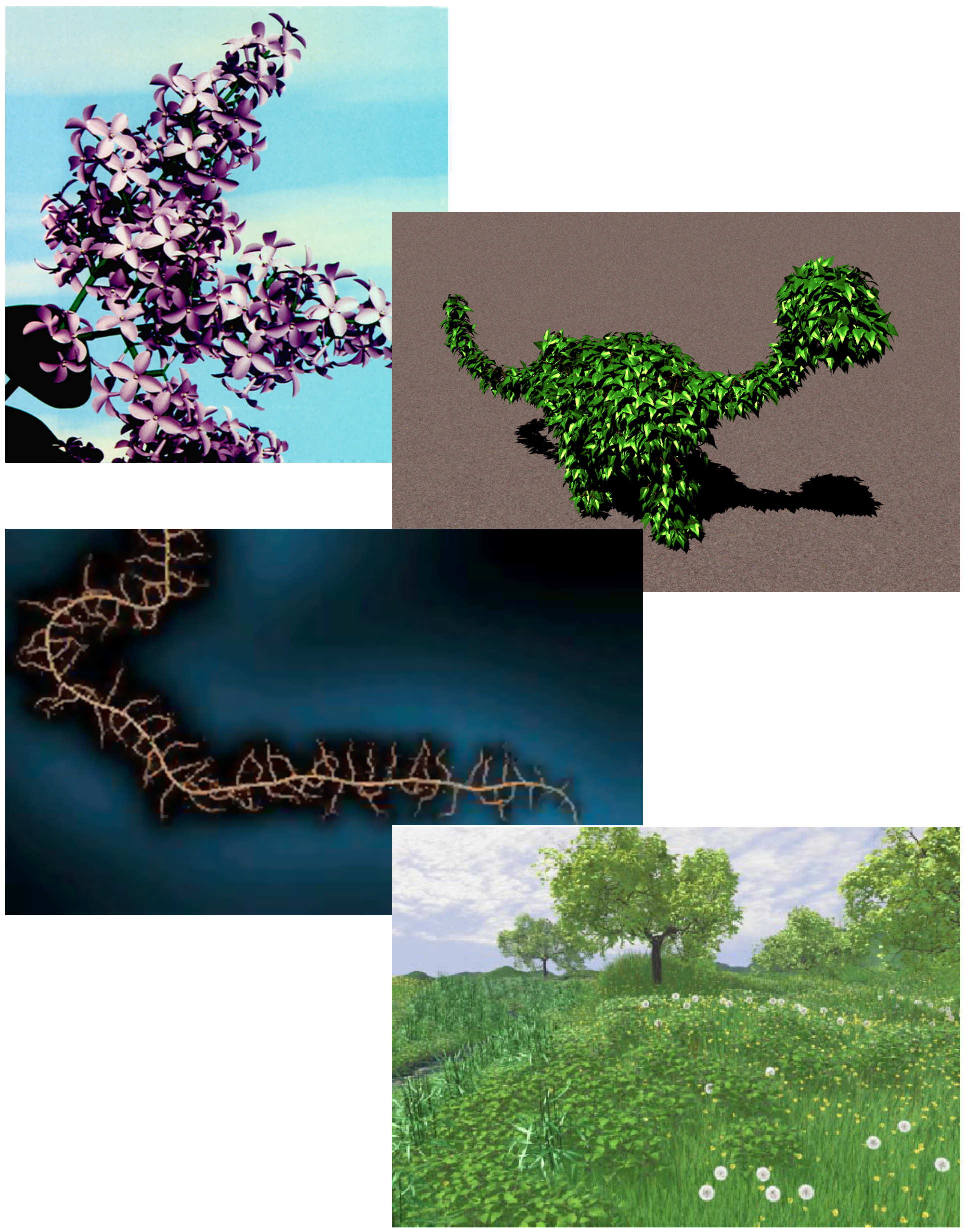
PovTree


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

L-Systems



L-Systems



- Generation of plants
  Prusinkiewicz, Lindenmayer; 1990
- Environment-sensitive
  Prusinkiewicz, James, Mech; 1994
- Interaction (Open L-System)
  Mech, Prusinkiewicz; 1996
- Ecosystems
  Deussen, et al.; 1998

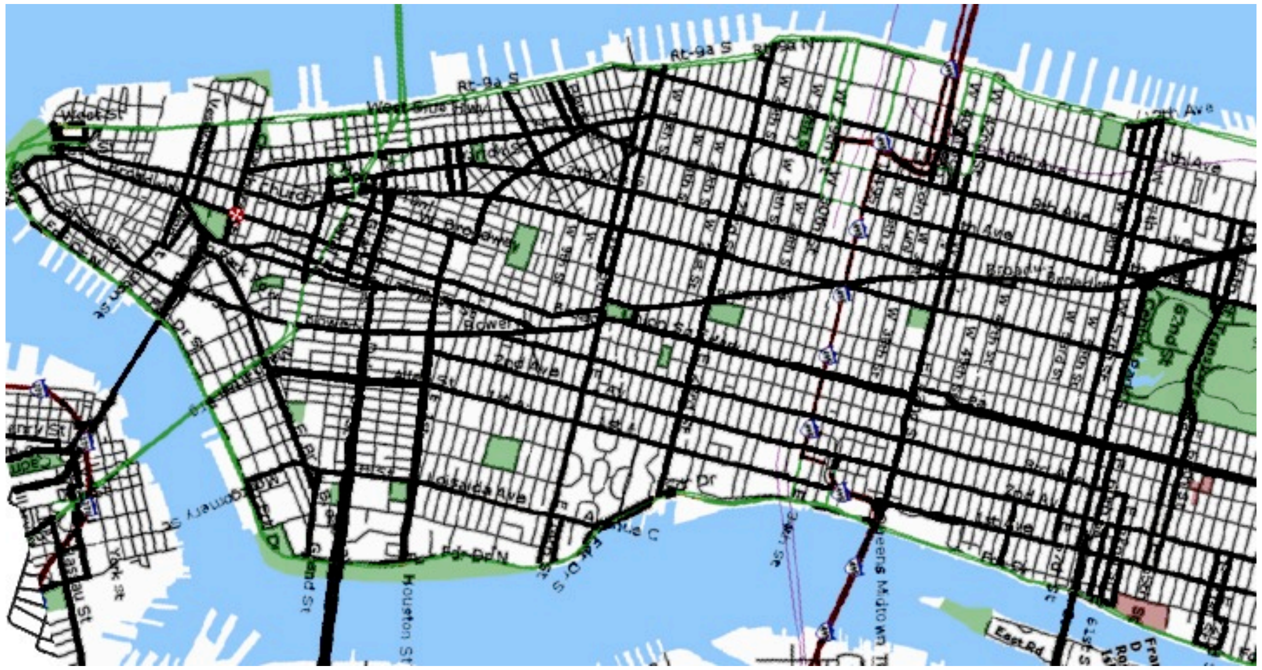
L-Systems Grammar: Extensions

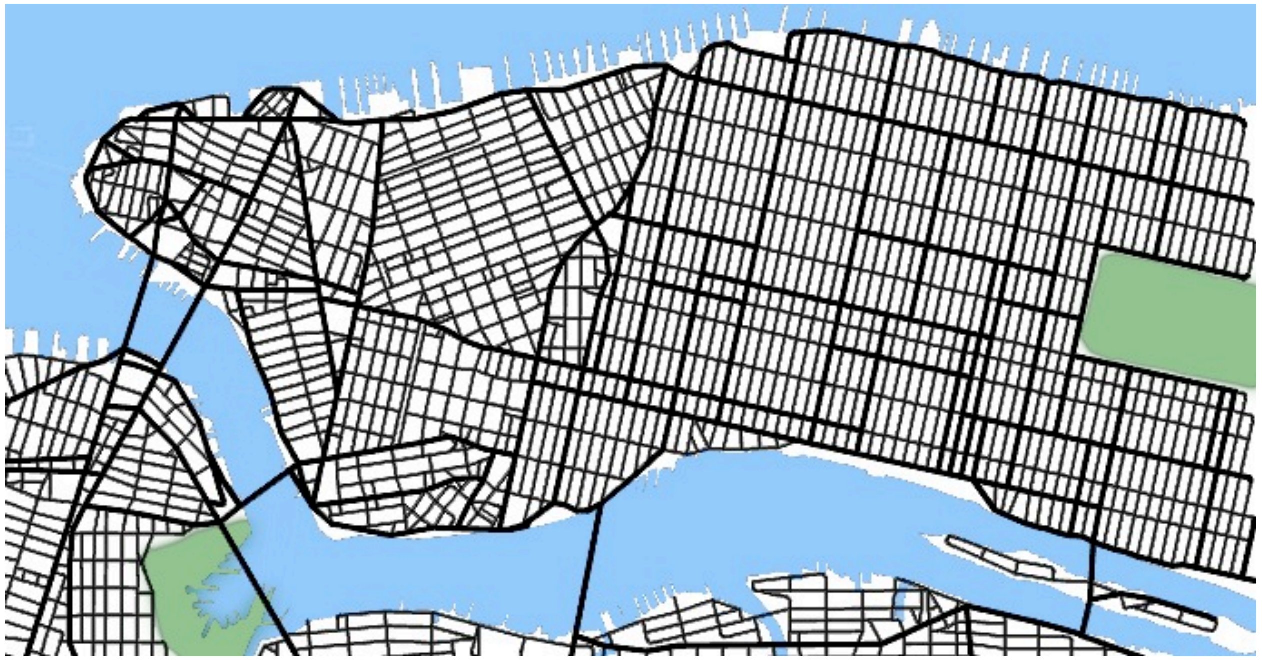
- Basic L-Systems have inspired a large number of variations
  - Context sensitive: productions look at neighboring symbols
  - Bracketed: save/restore state (for branches)
  - Stochastic: choose one of *n* matching productions randomly
  - Parametric: variables can be passed between productions

L-Systems: Further Readings

- Algorithmic Botany
  - Covers many variants of L-Systems, formal derivations, and exhaustive coverage of different plant types.
  - http://algorithmicbotany.org/papers
- PovTree
  - http://propro.ru/go/Wshop/povtree/povtree.html
  - http://arbaro.sourceforge.net/

L-Systems for Cities/Game Levels



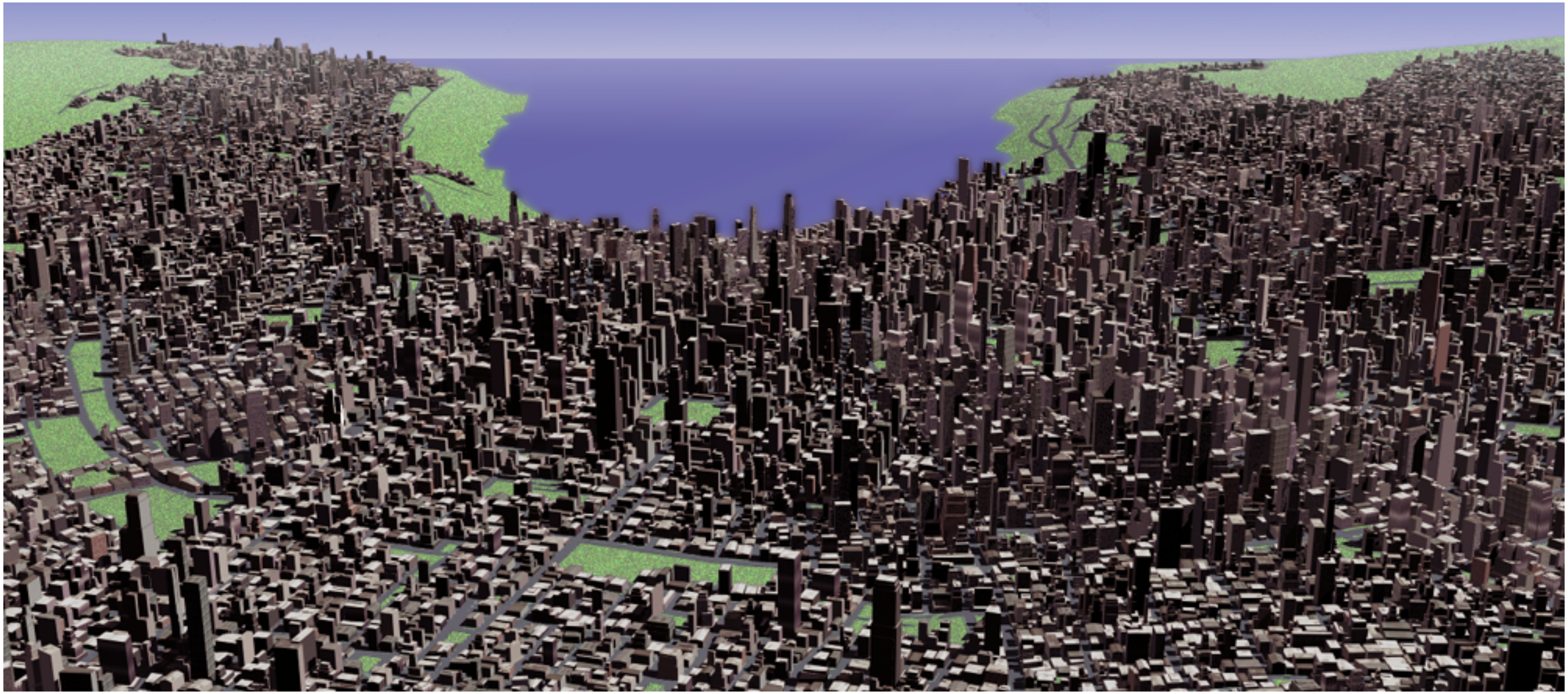


- Start with a single street
- Branch & extend w/ parametric L-System
- Parameters of the string are tweaked by goals/constraints
- Goals control street direction, spacing
- Constraints allow for parks, bridges, road loops
- Once we have streets, we can form buildings with another L-System
- Building shapes are represented as CSG operations on simple shapes

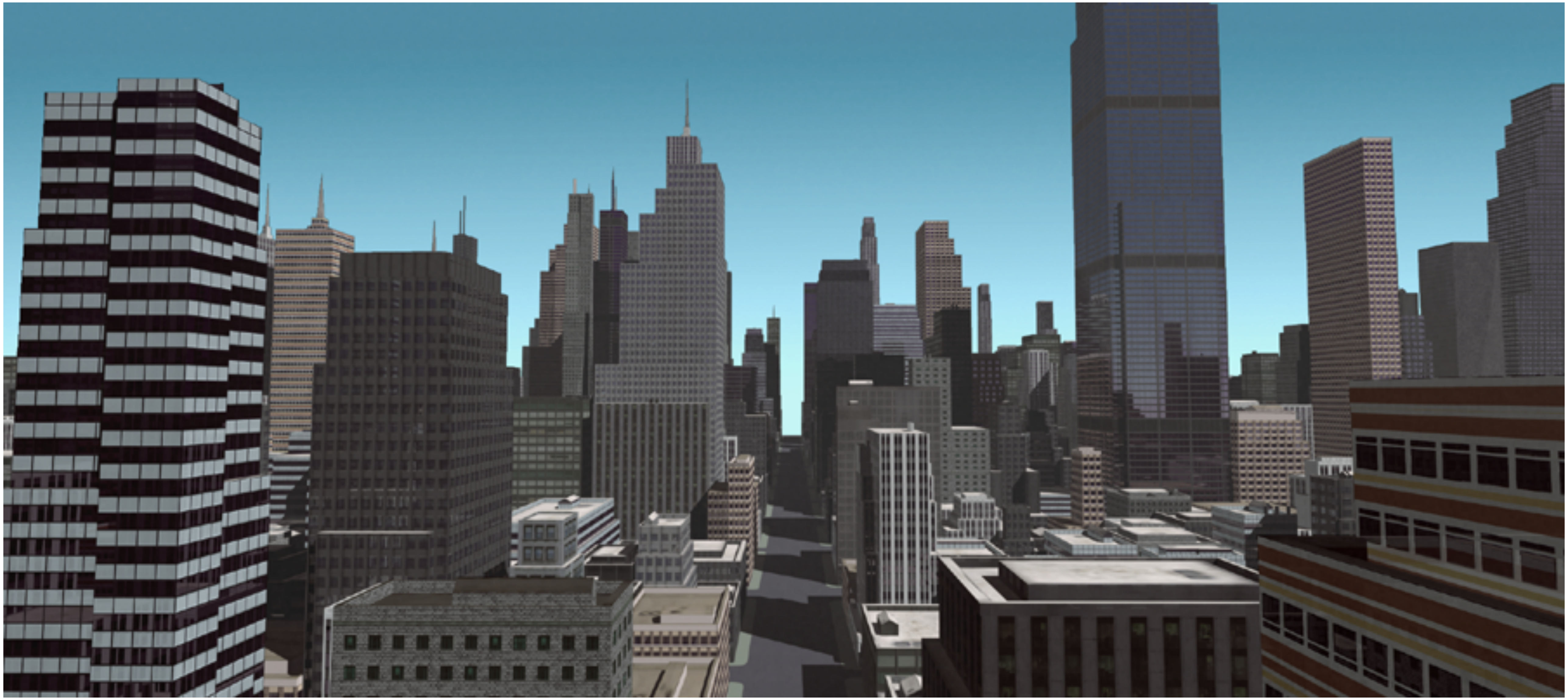
The City Engine System

- Procedurally creates complex city models.
- Cities consist of:
  - Street maps
  - Buildings
  - Facade textures

Example Zurich-London-Paris

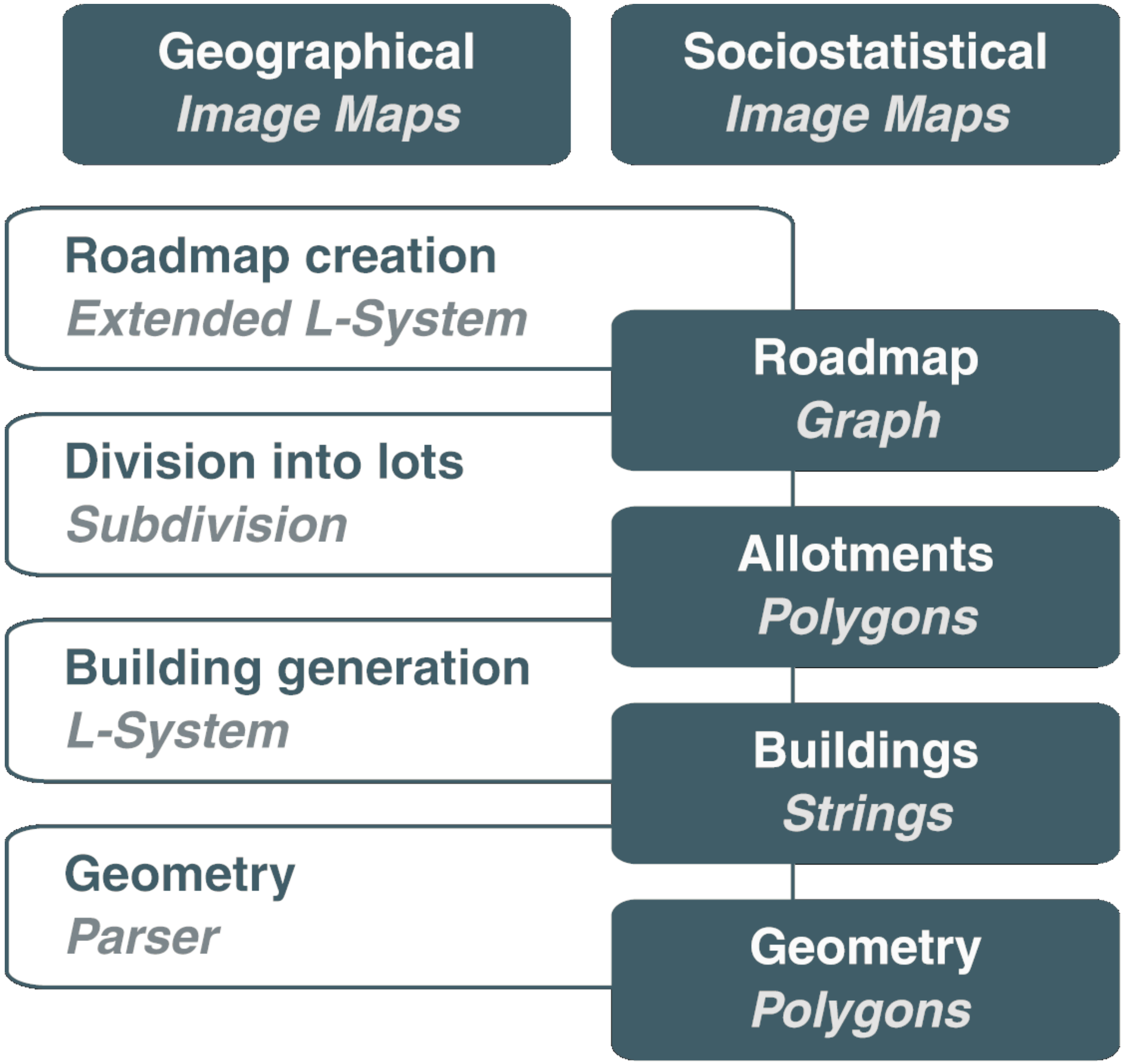


Example Manhattan

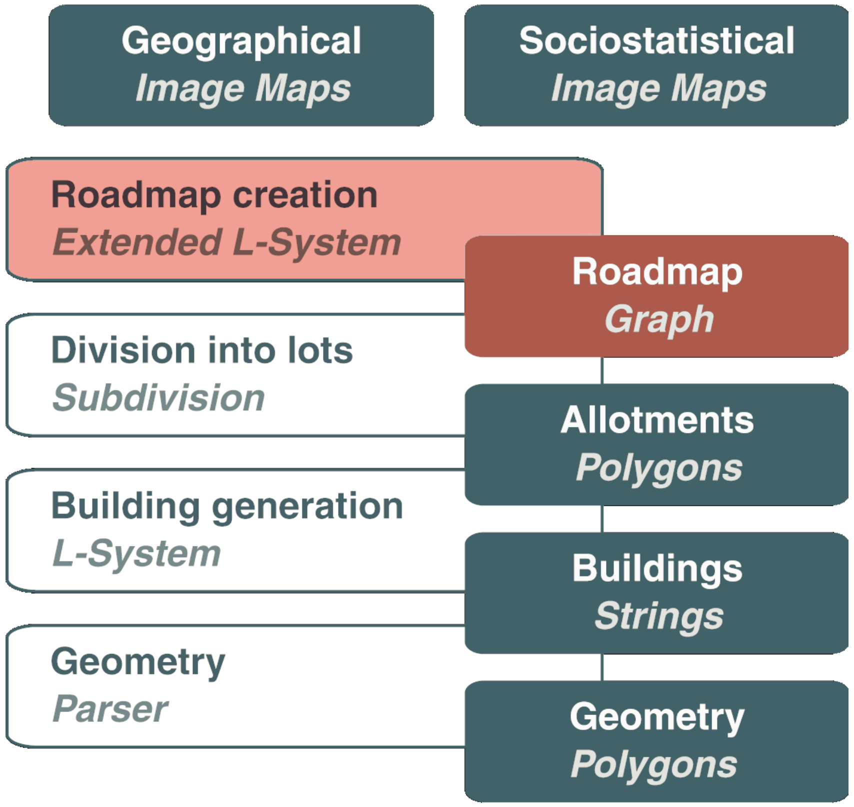


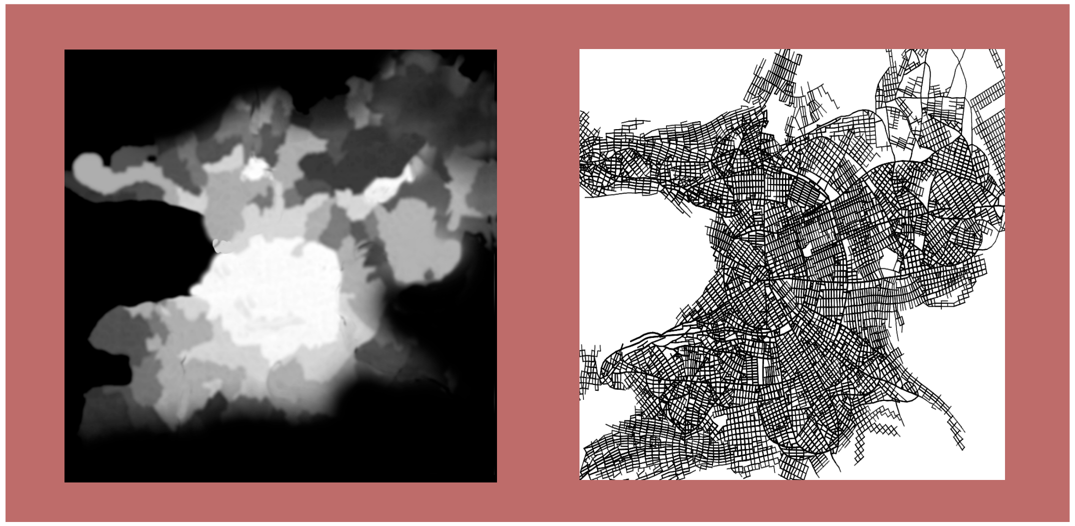
Example Manhattan 2259

System Pipeline



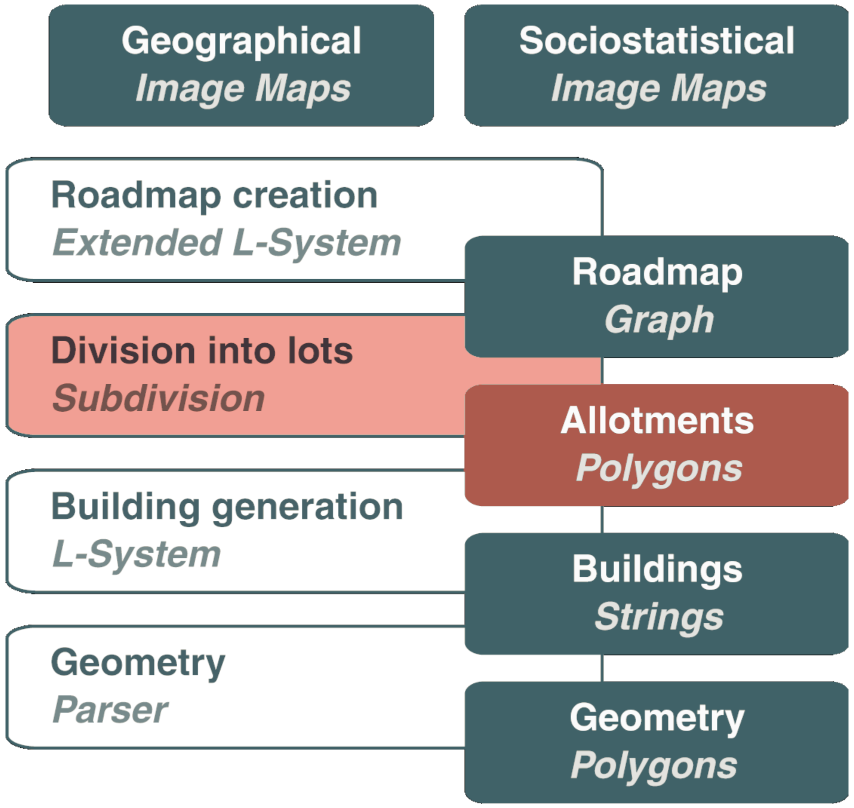
Module 1: Streetmap Creation

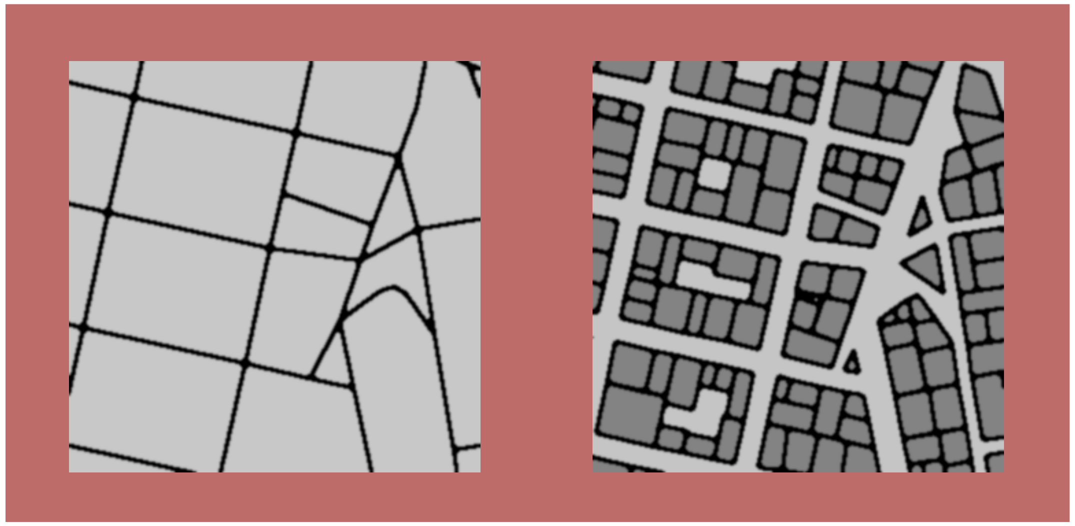




- Input:
  Image maps, parameters for rules
- Output:
  A street graph for interactive editing

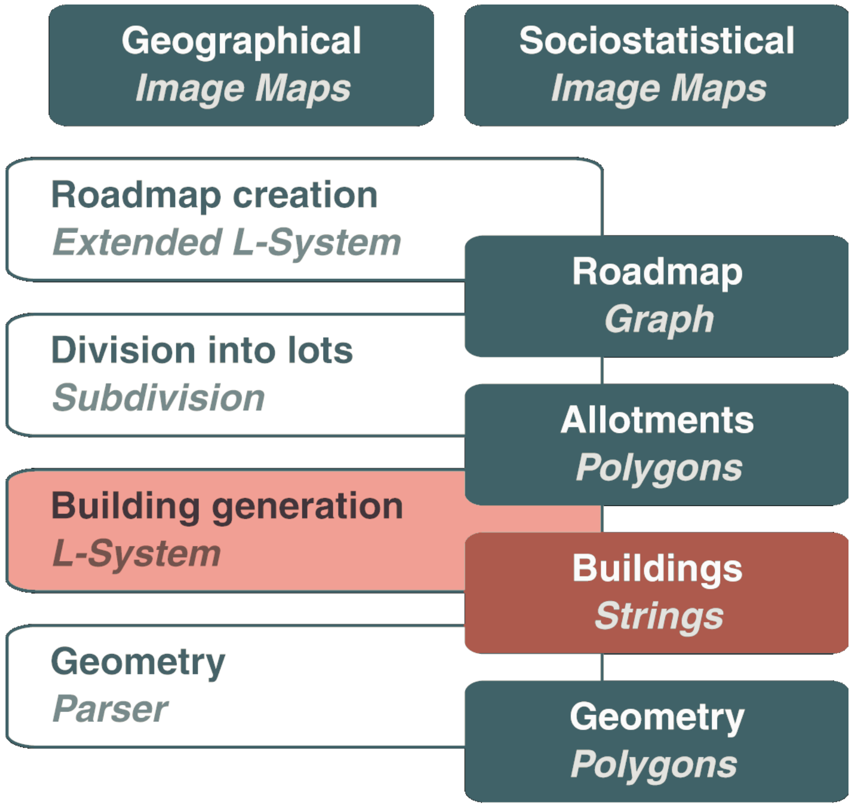
Module 2: Division into Lots

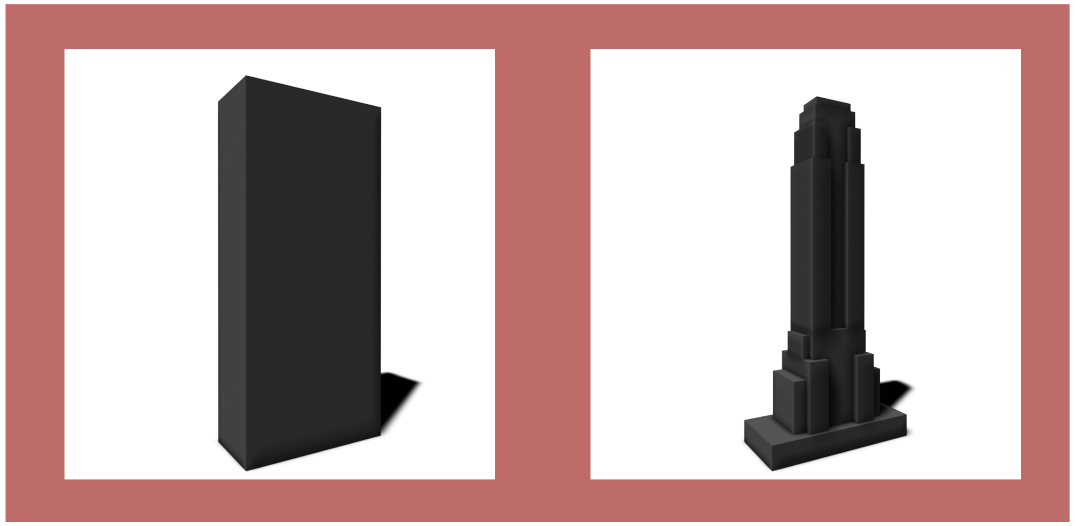




- Input:
  Street graph, area usage map
- Output:
  Polygon set of allotments for buildings

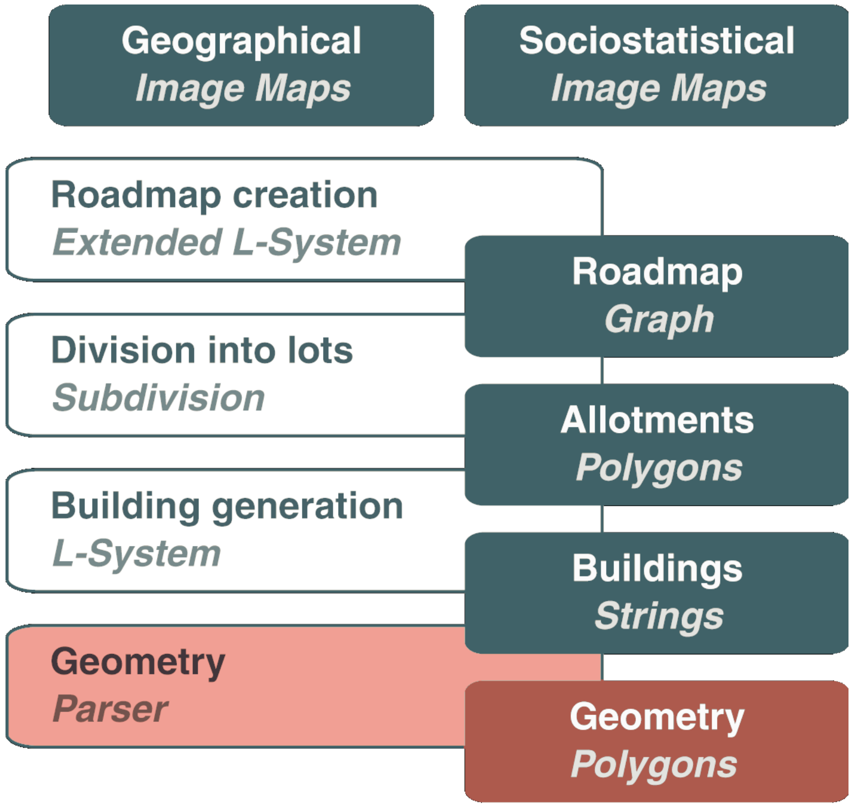
Module 3: Building Generation

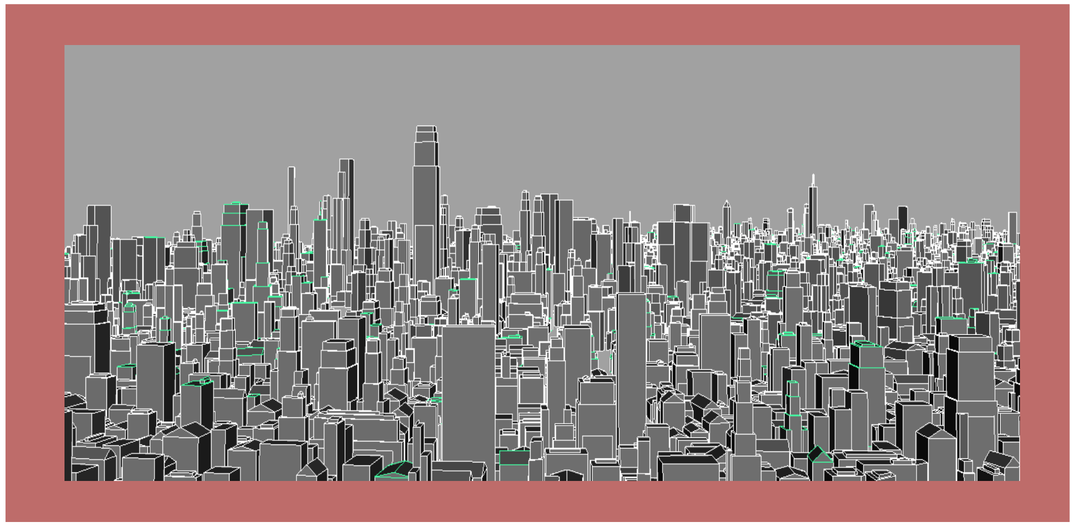




- Input:
  Lot polygons, age map and zone plan
- Output:
  Building strings with additional info

Module 4: Geometry and Facades





- Input:
  Strings and building type
- Output:
  City geometry and facade texture (procedural shader)

L-Systems for Streets

- Grouping parameters of different street patterns
- Hierarchical influences: global goals and local constraints

Extended L-Systems

- Template successor defines 3 branches
- Parameters fields are unassigned

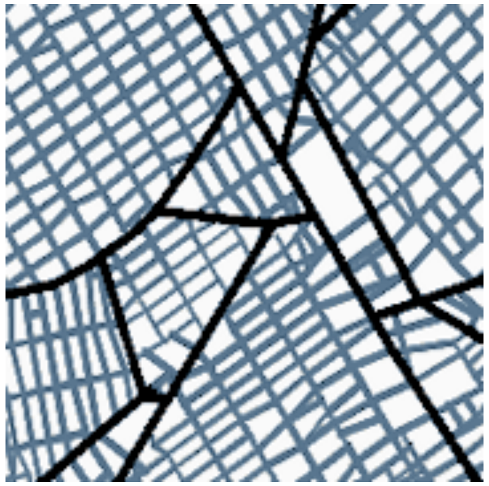
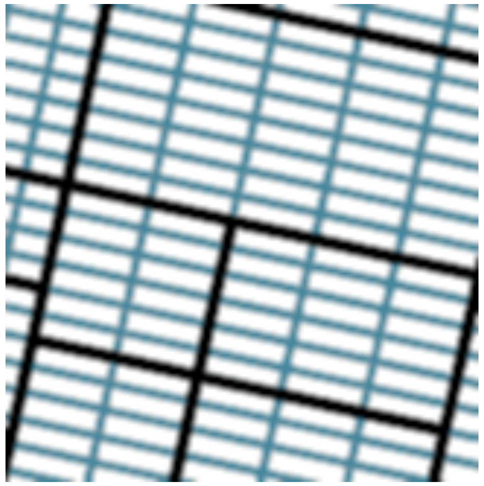
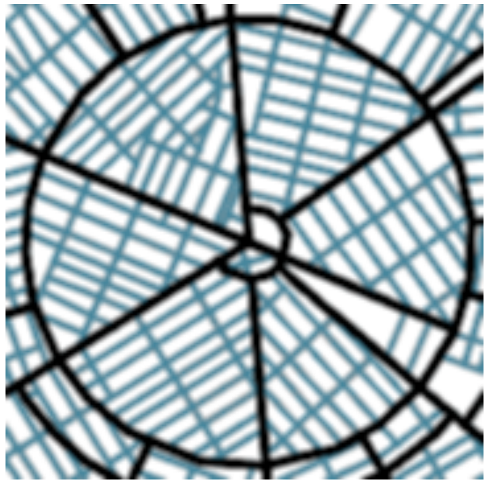
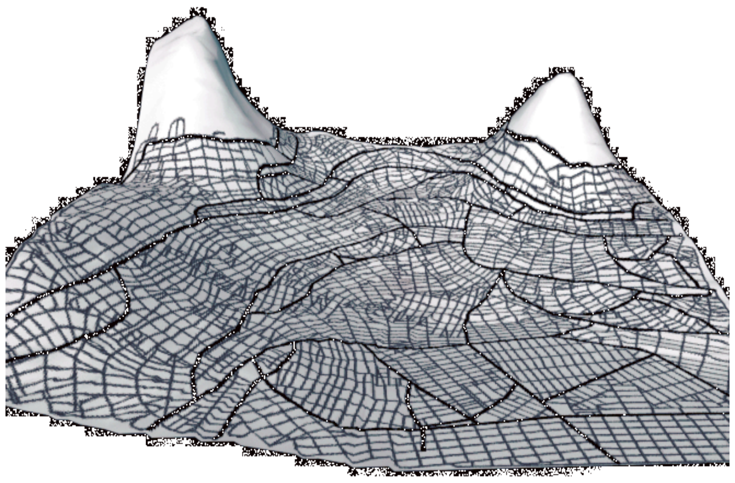
Extended L-Systems

- Initial parameter settings
- Design goal

Extended L-Systems

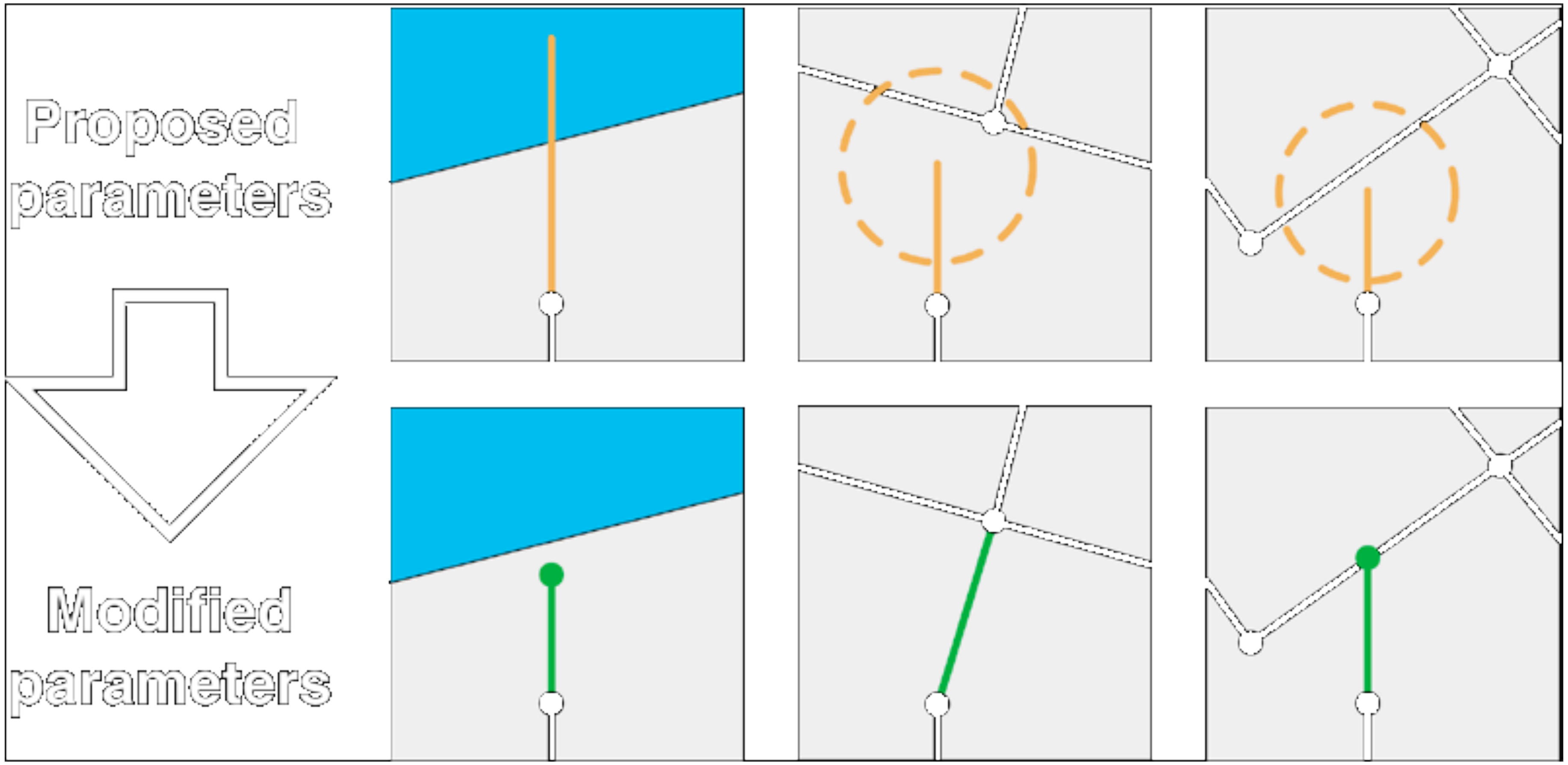
- Parameter value correction
- Influenced by local environment

Global Goals

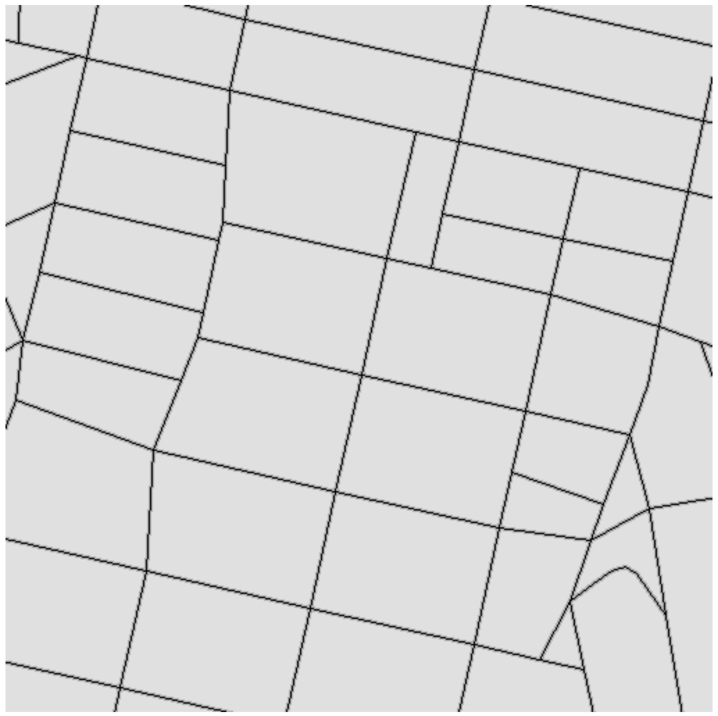
- Could be a planned urban design
- Different goals in the same city
- Controlled by image map (user input)

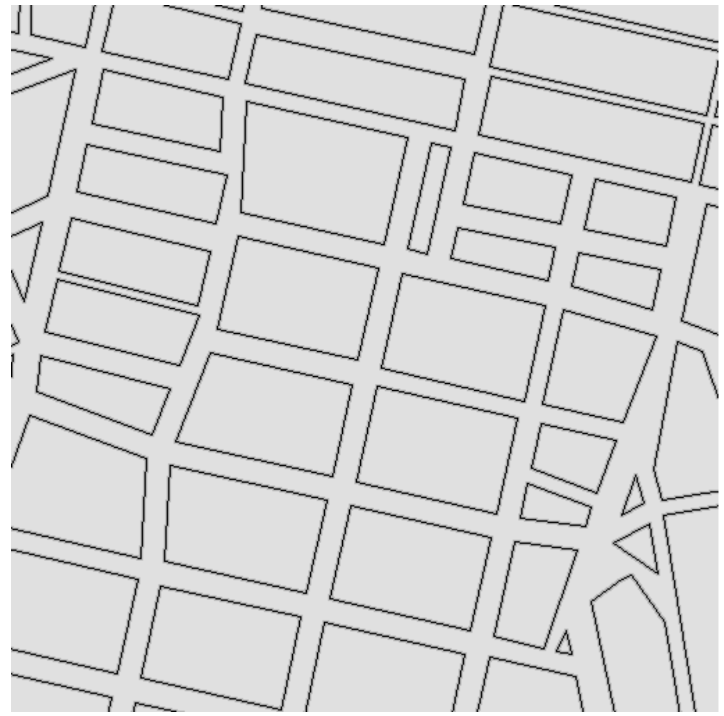
Local Constraints



- Environment-sensitivity for legal streets
- Self-sensitivity for closed loops

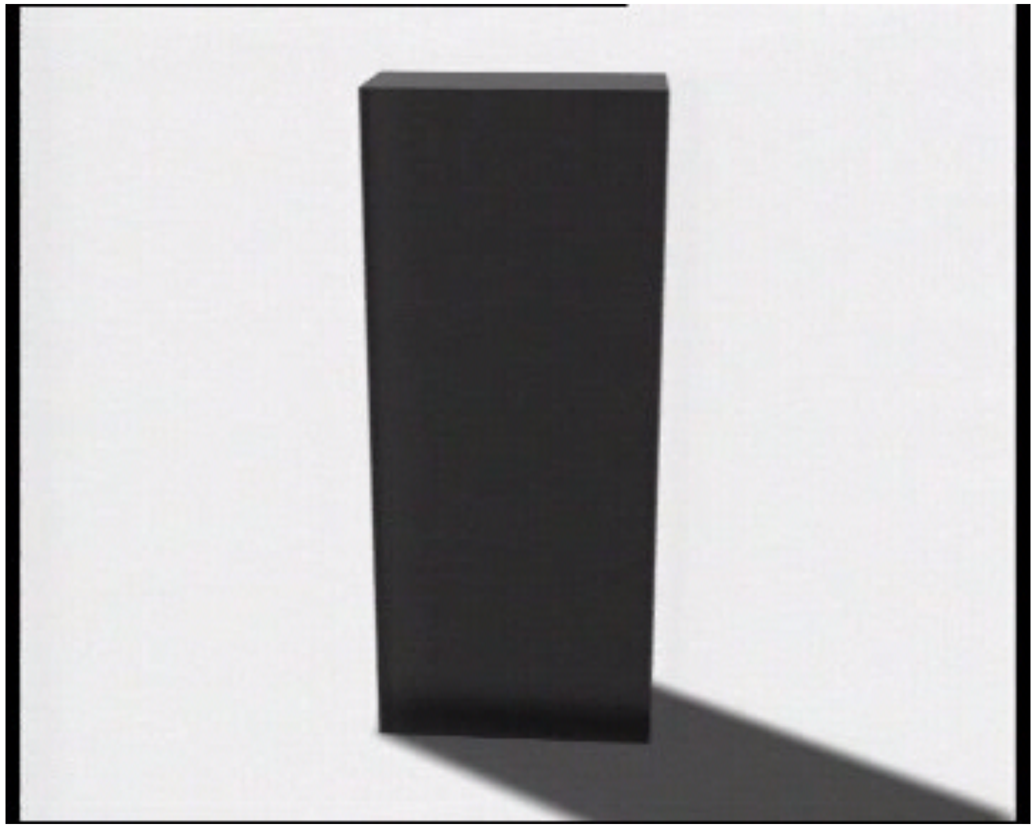
Division into Lots





- Lot area depends on:
  - Land Use map
  - Population density
  - **Building height**
  - **Access to street**

Procedural Buildings

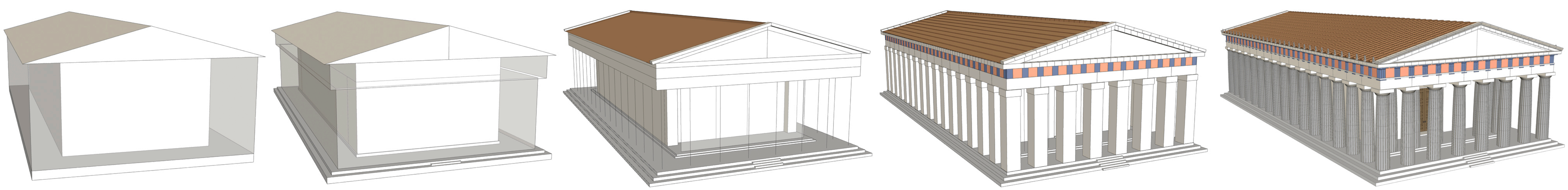


- Modeled with a common L-System
- L-System modules consist of geometric operations like extrusion

CGA Shape

Production process:

- Rule-driven modification & replacement of shapes
- Iteratively evolve a design by creating more and more details
- Sequential application (like Chomsky grammars)

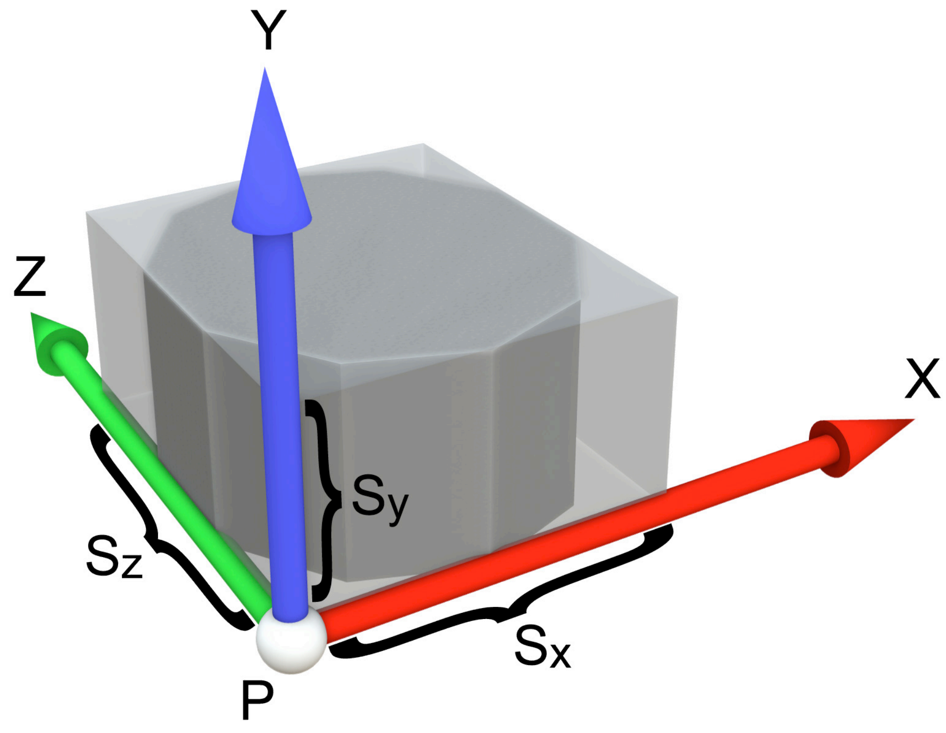


Shape Rules

- Notation:

id: predecessor : condition ⇢ successor : prob

- A shape consists of:
  - Symbol (string)
  - Geometry (geometric attributes)
  - Oriented bounding box called **scope** (numeric attributes)

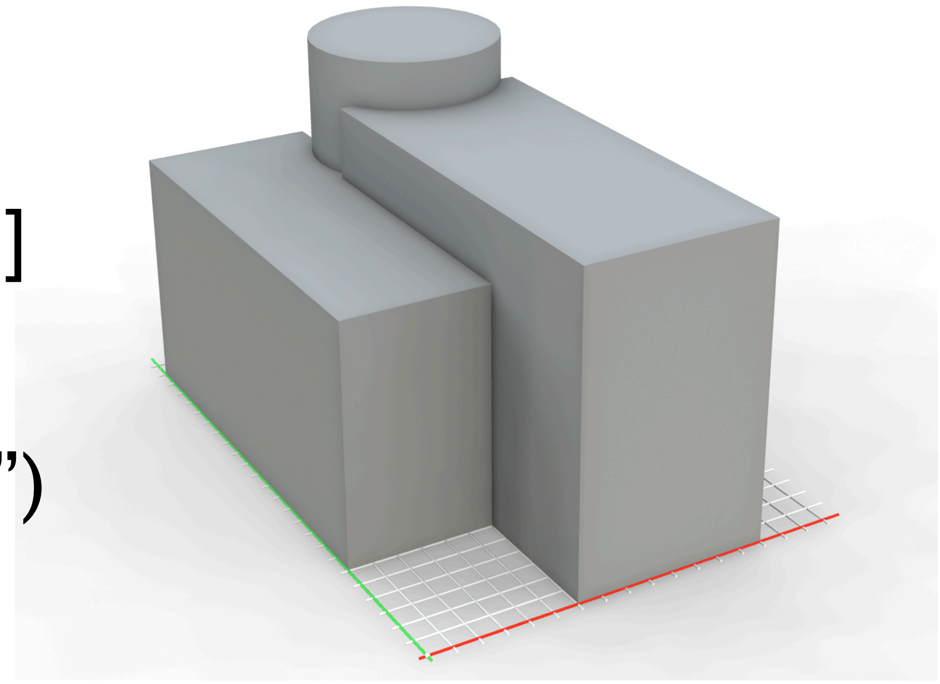


Basic Shape Operations

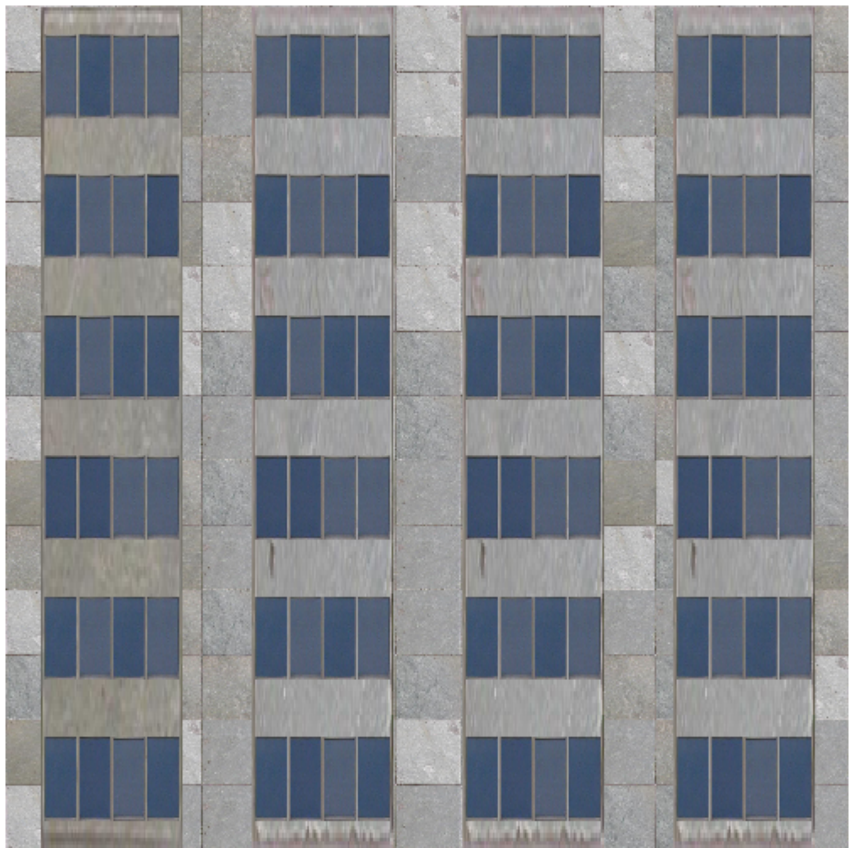
- Insertion: I(obj_Id)
- Transformations: $T(t_x,t_y,t_z)$, $S(s_x,s_y,s_z)$, $Rx(\alpha)$..
- Branching: [...]

Simple example:

1: A ⇢ [T(0,0,6) S(8,10,18) I("cube")]
T(6,0,0) S(7,13,18) I("cube")
T(0,0,16) S(8,15,8) I("cylinder")

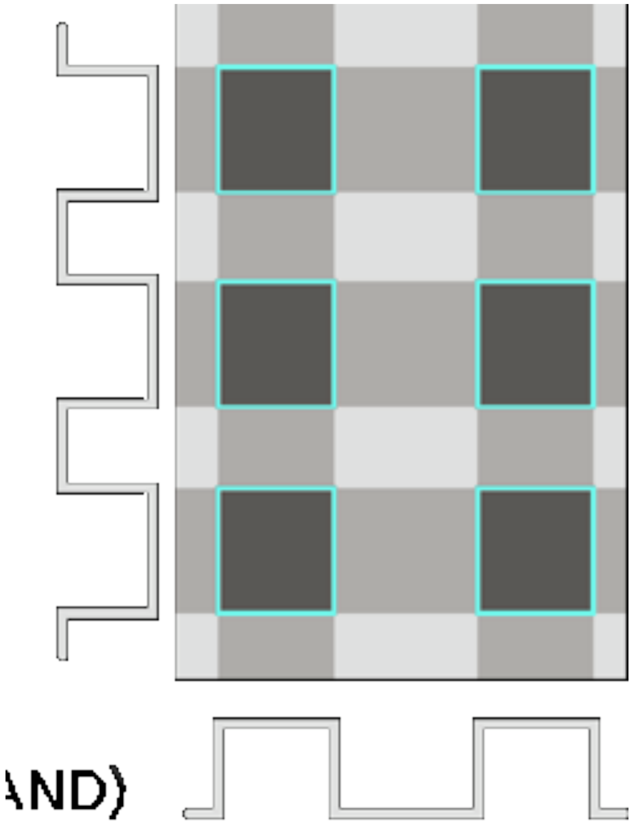


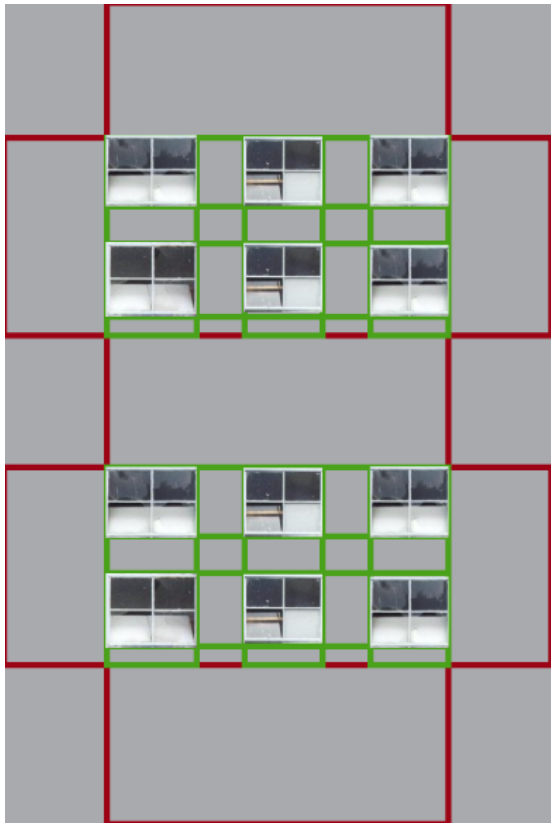
Facade Textures



- Division into simple grid-like structures
- Structures can be layered

Layered Textures





- Two base functions form a layer
- Every layer defines a facade element

Procedural Modeling of Buildings

Example-Based Approach