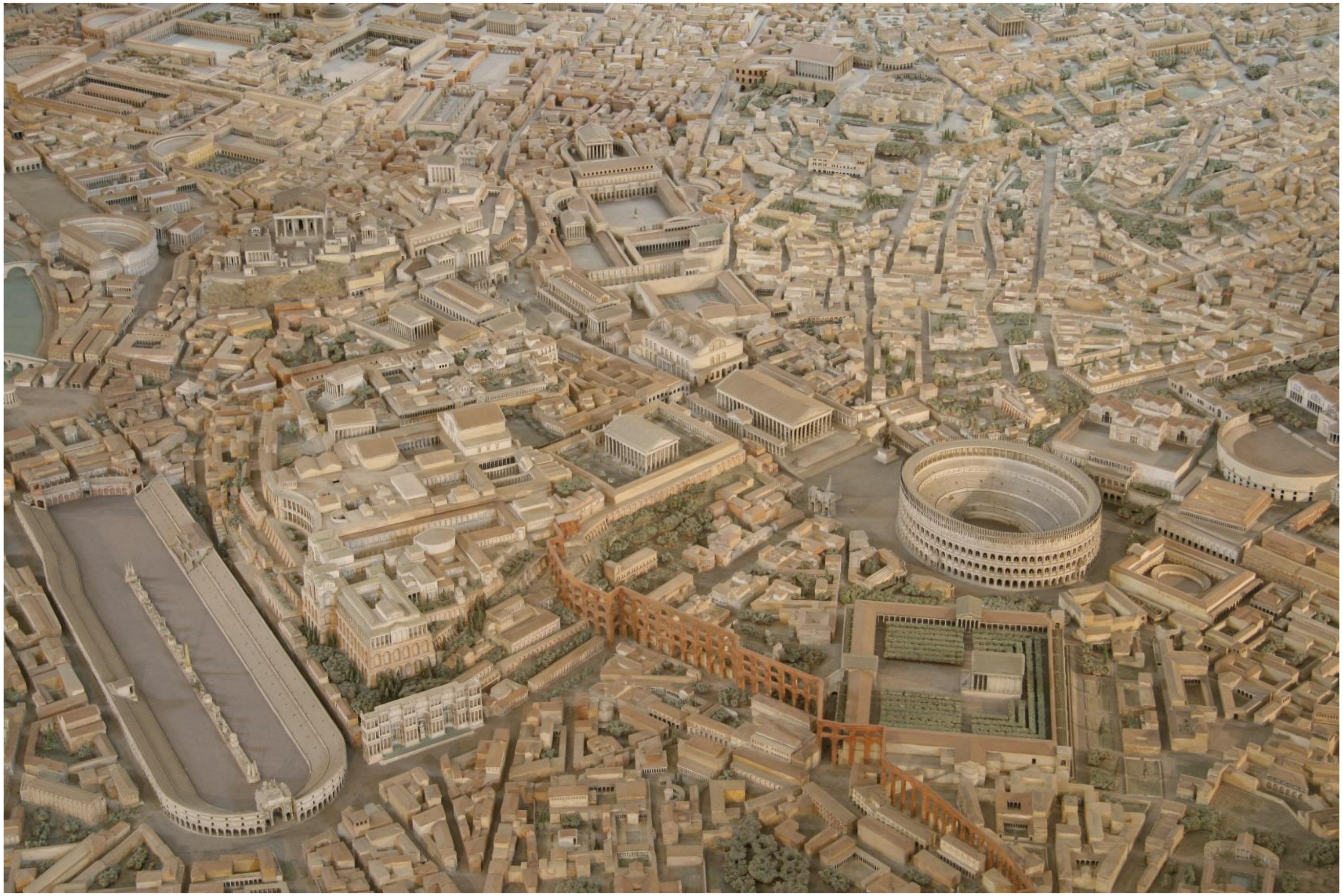
Procedural Modeling

CS334 Spring 2025

Daniel G. Aliaga
Department of Computer Science
Purdue University

How to?

How to?



How to?

How to?

Procedural Modeling

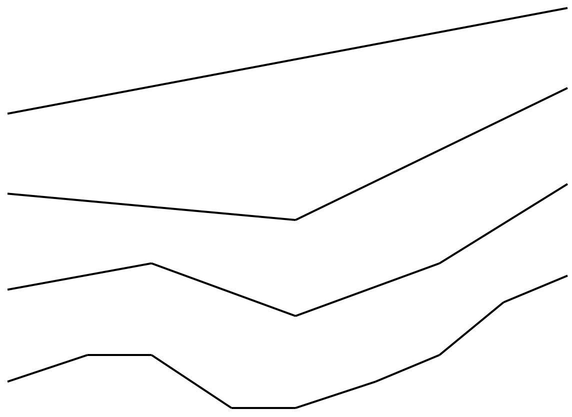
- Apply algorithms for producing objects and scenes
- The rules may either be embedded into the algorithm, configurable by parameters, or externally provided

Procedural Modeling

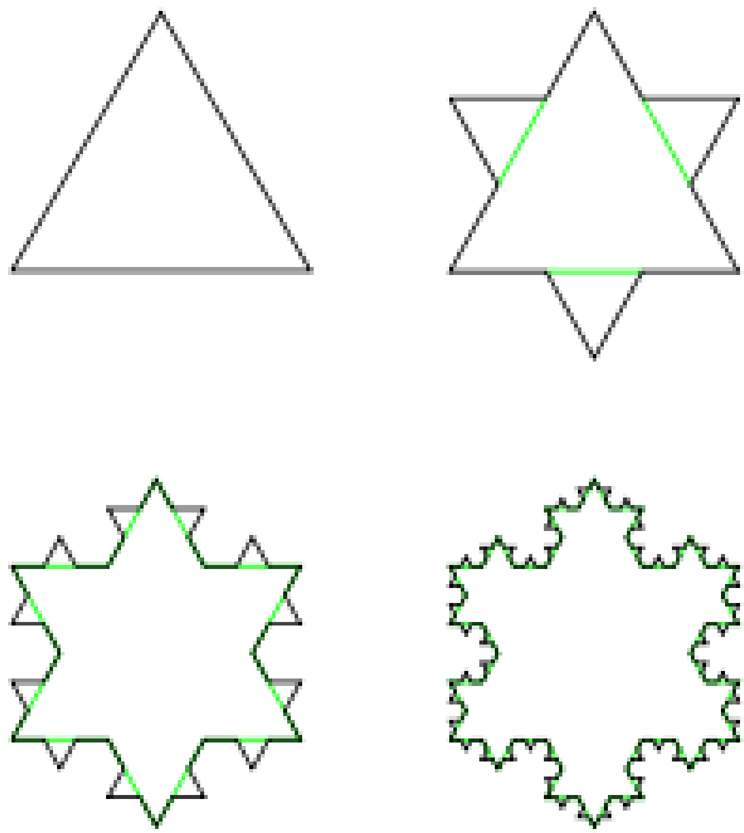
- Fractals
- Terrains
- Image-synthesis
  - Perlin Noise
  - Clouds
- Plants
- Cities
- And procedures in general…

Linear Fractals

- Definition: a shape is repeated in different orientations/scales -- a never-ending pattern.
- Consider a simple line fractal
  - Split a line segment, randomize the height of the midpoint by some number in the [-*r*,*r*] range
  - Repeat and randomize by [-*r*/2,*r*/2]
  - Continue until a desired number of steps, randomizing by half as much each step



Koch Snowflake



Demo

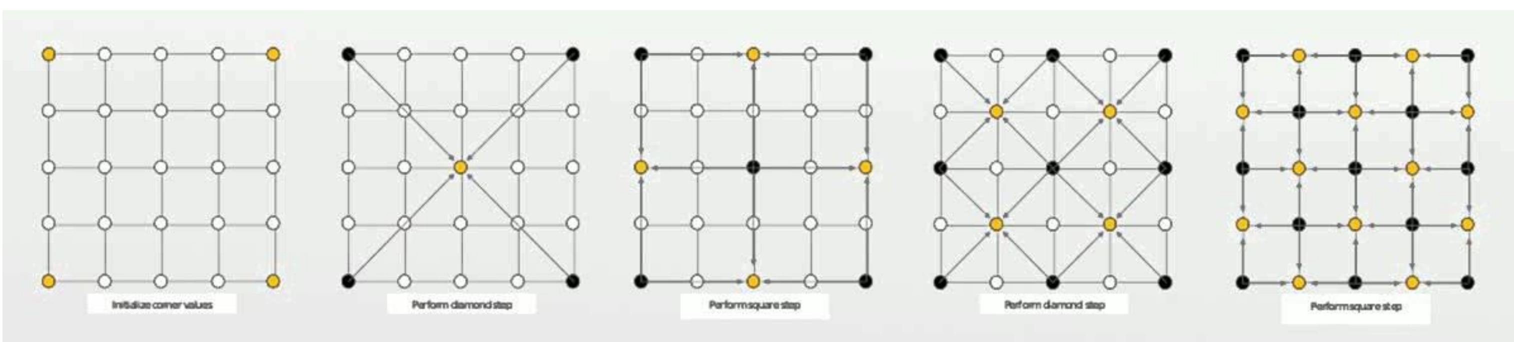
- http://nolandc.com/sandbox/fractals/

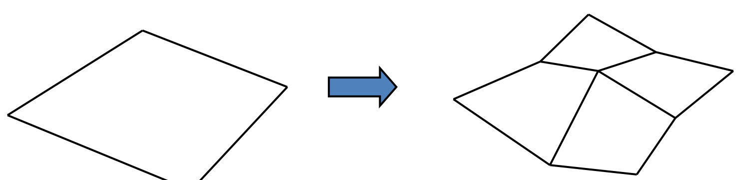
Non-linear Fractals

- Example: Mandelbrot Set
  - Iterations of "$z_{n+1} = z_n^2 + C$" starting at $z_0 = c_0$
  - https://www.youtube.com/watch?v=pCpLWbHVNhk

Fractals and Terrains

- A similar process can be applied to squares in the xz plane (Diamond-Square Algorithm):
  - At each step, an xz square is subdivided into 4 squares, and the y component of each new point is randomized
  - By repeating this process recursively, we can generate a mountain landscape





Terrains

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

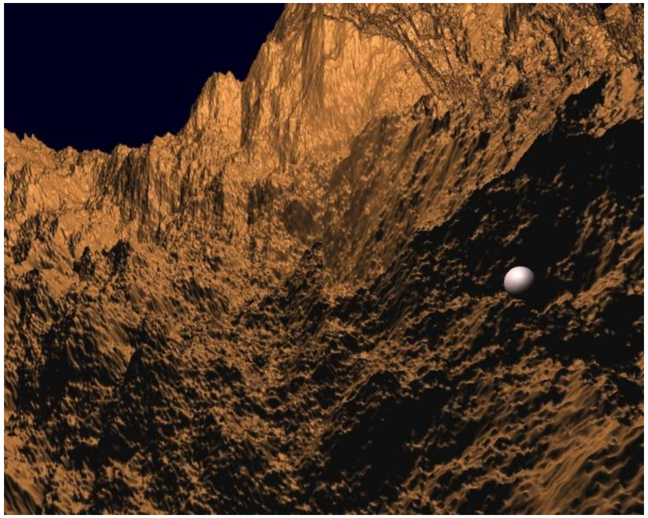
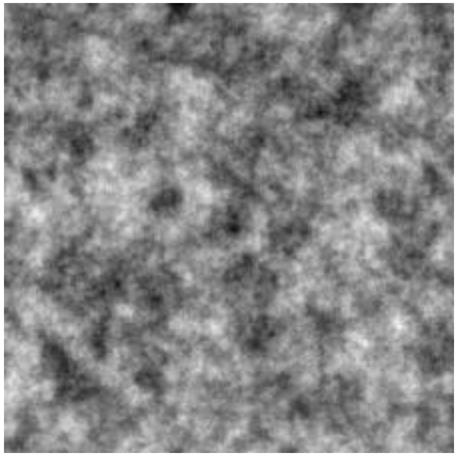


Image Synthesis

- Procedurally generate an image (pixels)

Idea: Perlin Noise

- Procedurally generate noise
  - http://js1k.com/demo/543
- See other slides



City Modeling

- Procedural Modeling of Cities

  (more on this later…)

Plant Modeling

- The Algorithmic Beauty of Plants

Background: Chomsky Hierarchy

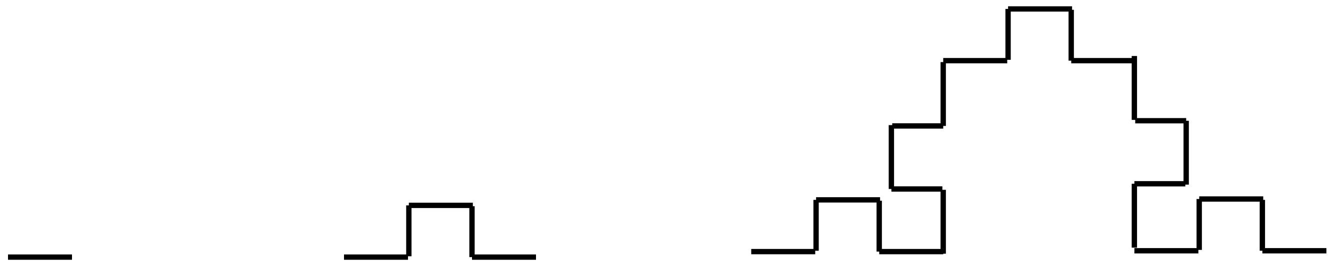
- Type 0 grammars
  - Unrestricted, recognized by Turing machine
- Type 1 grammars
  - Context-sensitive grammars
- Type 2 grammars
  - Context-free grammars
- Type 3 grammars
  - Regular grammars (e.g., regular expressions)

Lindenmayer system (or L-system)

- A context-free or context-sensitive grammar
- All rules are applied in “every iteration” before jumping to the next level/iteration
- Can be deterministic or non-deterministic

L-system

- Variables: a
- Constants: +, - (rotations of + or – 90 degrees)
- Initial string (axiom): s=a
- Rules: a → a+a-a-a+a



What does this produce?

Initial string: | (=F)

Rotation: 25-degrees

Rule: F -> F[+F]F[-F]F

Iterations: 1

What does this produce?

Initial string: | (=F)

Rotation: 25-degrees

Rule: F -> F[+F]F[-F]F

Iterations: 2

What does this produce?

Initial string: | (=F)

Rotation: 25-degrees

Rule: F -> F[+F]F[-F]F

Iterations: 3

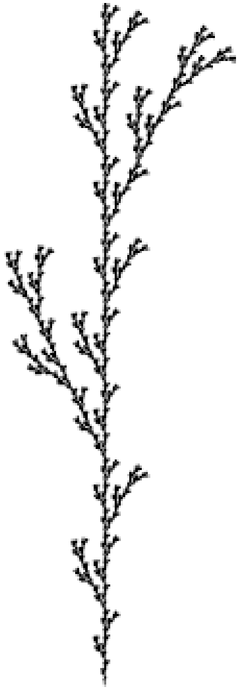
What does this produce?

Initial string: | (=F)

Rotation: 25-degrees

Rule: F -> F[+F]F[-F]F

Iterations: 5



a

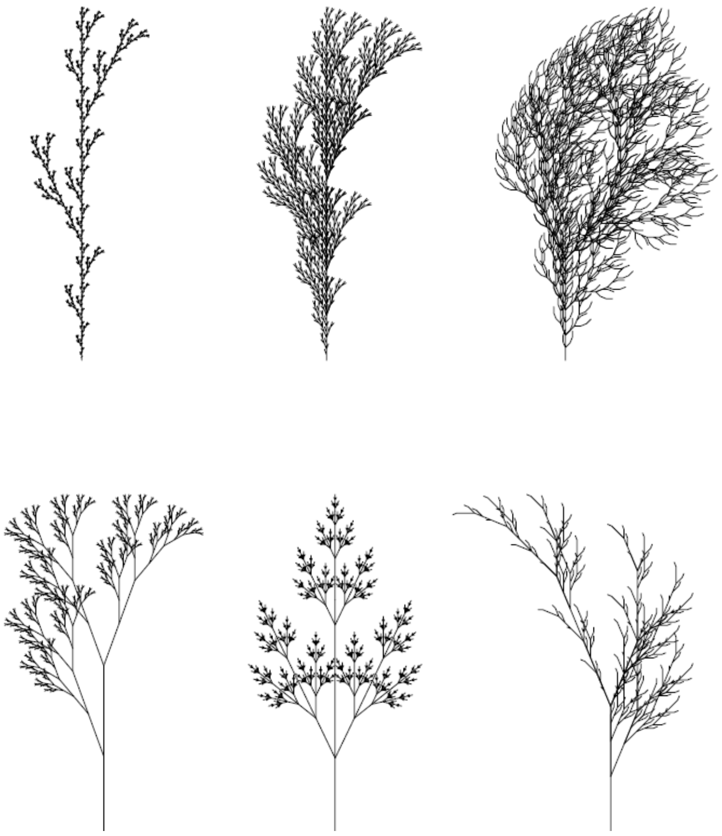
n=5, δ=25.7°

F

F → F[+F]F[-F]F

Exercise!

- **You propose a L-system**
- Starting string:
- Rotation angle:
- Rule:
- Num iterations (about)



(Context-Free) L-system for Plants

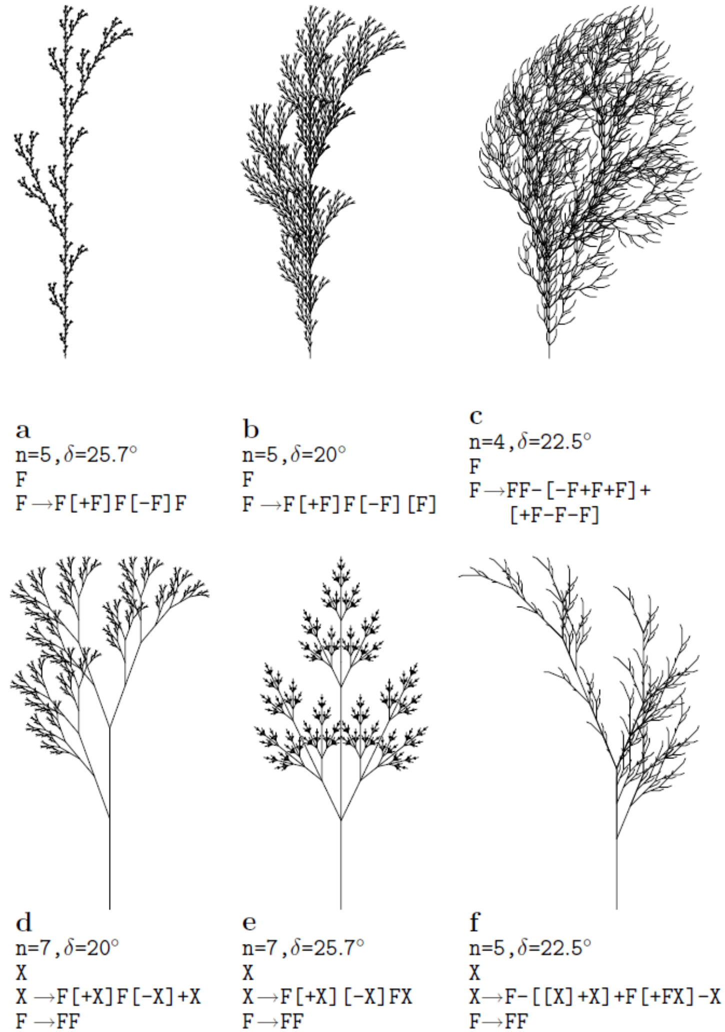
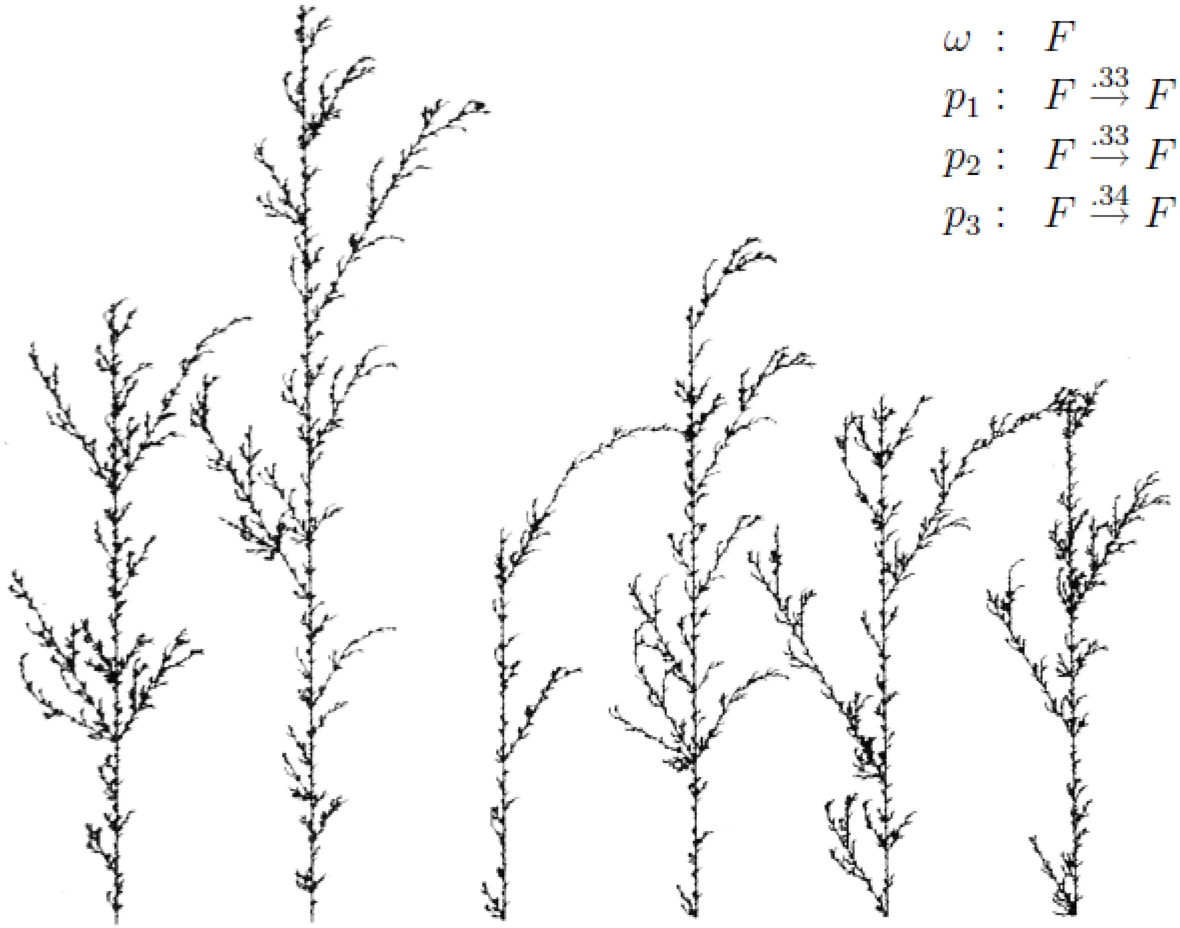


Figure 1.24: Examples of plant-like structures generated by bracketed OL-systems. L-systems (a), (b) and (c) are edge-rewriting, while (d), (e) and (f) are node-rewriting.

L-system for Plants (stochastic)



$$
\begin{aligned}
\omega &: F \\
p_1 &: F \xrightarrow{.33} F[+F]F[-F]F \\
p_2 &: F \xrightarrow{.33} F[+F]F \\
p_3 &: F \xrightarrow{.34} F[-F]F
\end{aligned}
$$

Figure 1.27: Stochastic branching structures

L-system for Plants (3D)

n=5, δ=18°

ω : plant
p_1 : plant → internode + [plant + flower] − − //
[− − leaf] internode [+ + leaf] −
[plant flower] + + plant flower
p_2 : internode → F seg [// & & leaf] [// ∧ ∧ leaf] F seg
p_3 : seg → seg F seg
p_4 : leaf → [' { +f−ff−f+ | +f−ff−f }]
p_5 : flower → [& & & pedicel ' / wedge //// wedge ////
wedge //// wedge //// wedge]
p_6 : pedicel → FF
p_7 : wedge → [' ∧ F] [{ & & & & −f+f | −f+f }]

Figure 1.26: A plant generated by an L-system

Figure 1.28: Flower field

Recent Result

- Growing Demo (Houdini)
  - https://www.youtube.com/watch?v=-e39SktwmkU
- SIGGRAPH Asia 2020
  - https://www.youtube.com/watch?v=MU9E7xJzVGs

Shape Grammar

- Is used to generate geometric models from a set of shapes and rules

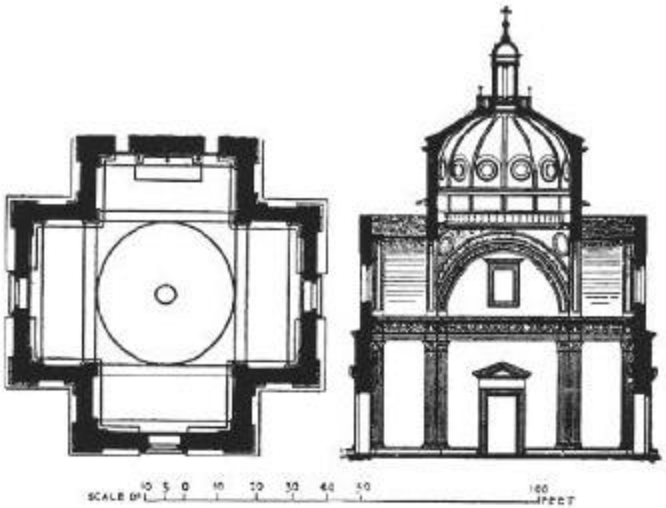
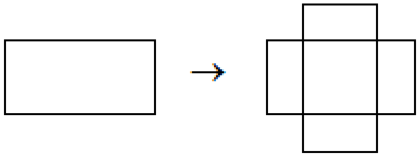


Illustration by Peter Murray, "the Artchitecture of the Italian Renaissance", Shocken Books Inc. 1963, Pp.96.

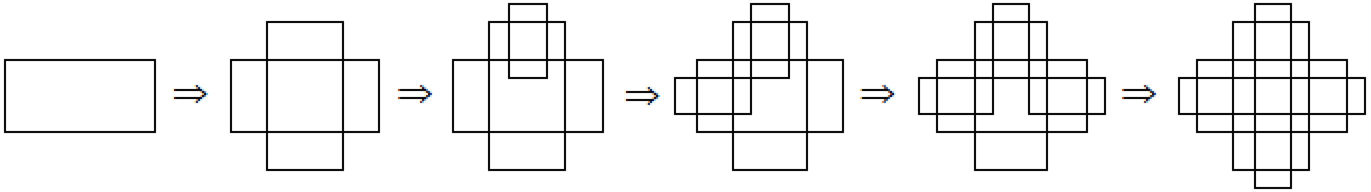
Shape Grammar

Shape Grammar

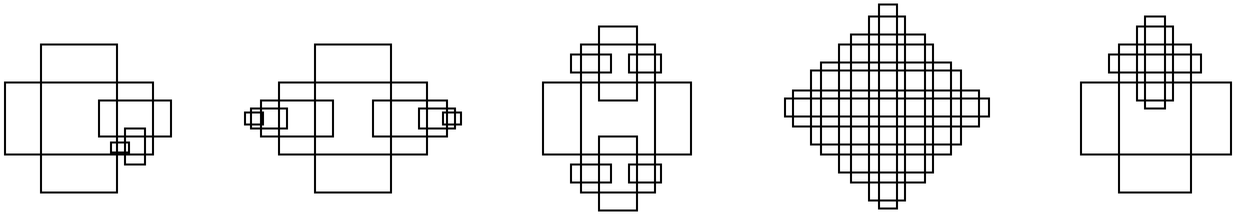


rule

DERIVATION



Shape Grammar



OTHER DESIGNS IN THE LANGUAGE

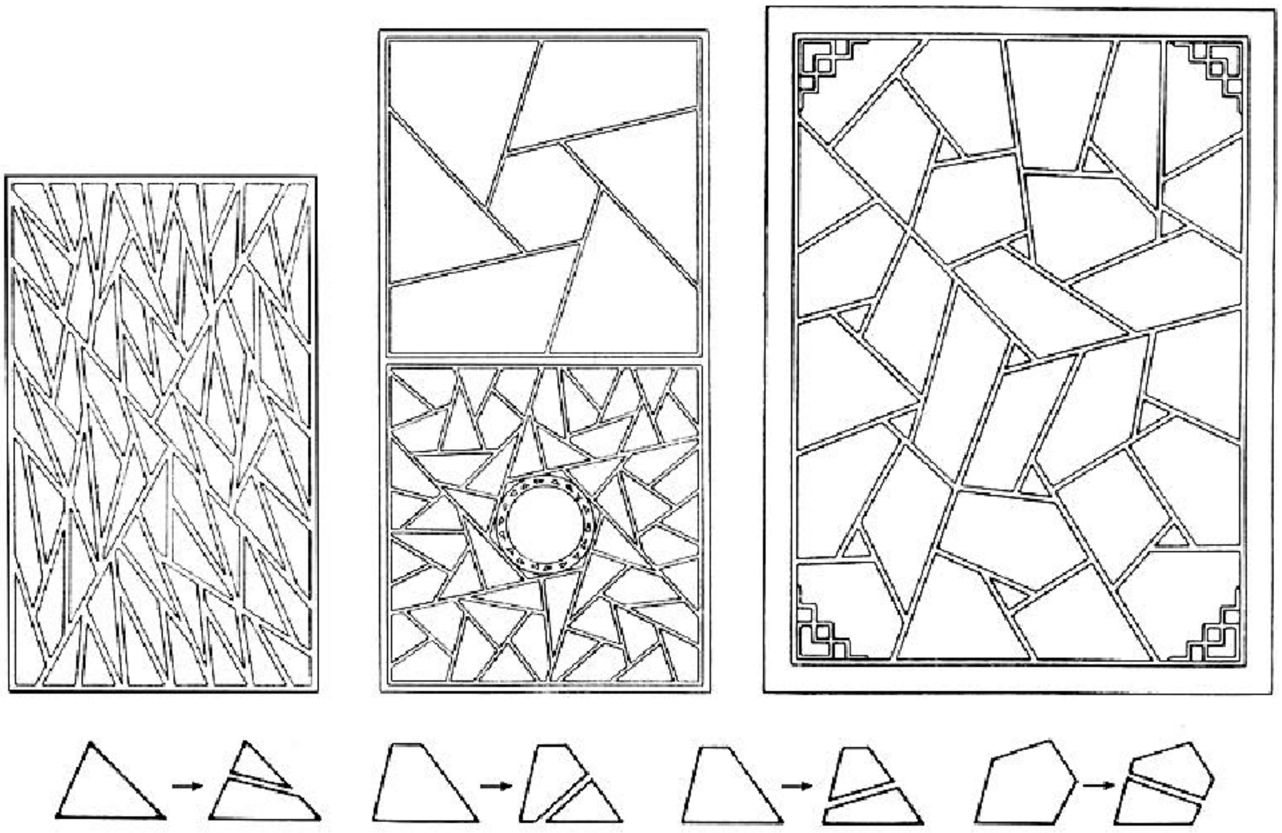
Exercise: let's make some art!

- What is a shape grammar that makes this?

Exercise: let’s make some art!

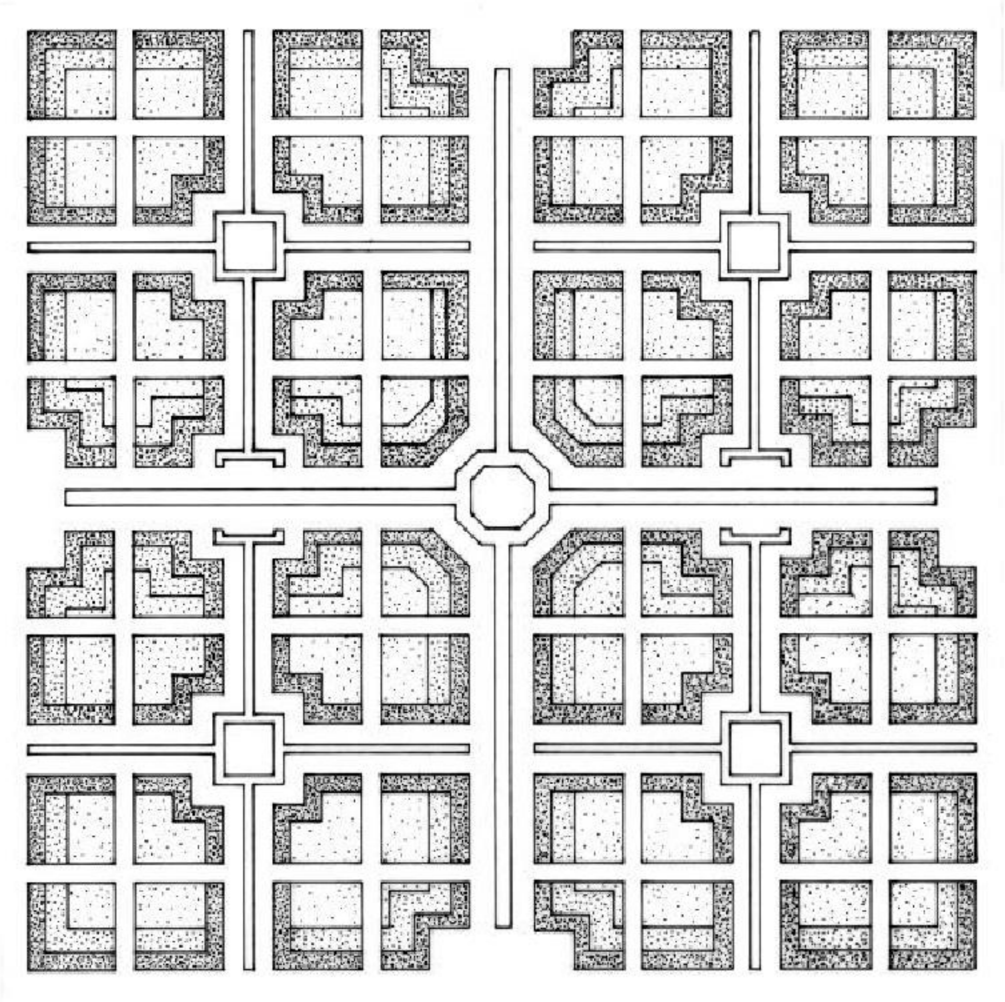
- Consult with your neighbor(s)
- What is the shape grammar that makes the art of the previous slide?
- Go!

Shape Grammar



Ice-ray grammar

Shape Grammar



Mughul garden grammar

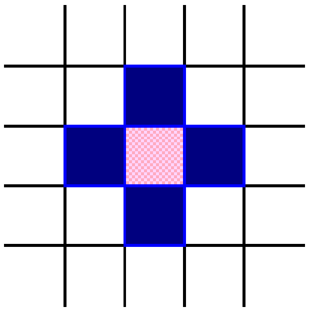
Shape Grammar

- Style: Mediterranean

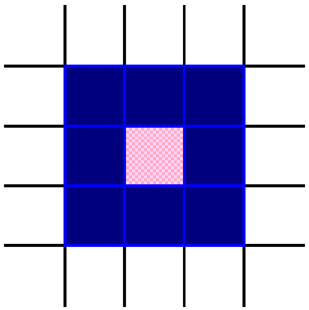
Cellular Automata

- A cellular automata (CA) is a spatial lattice of N cells, each of which is one of *k* states at time *t*.
- Each cell follows the same simple rule for updating its state.
- The cell's state *s* at time *t*+1 depends on its own state and the states of some number of neighbouring cells at *t*.
- For one-dimensional CAs, the neighbourhood of a cell consists of the cell itself and *r* neighbours on either side. Hence, *k* and *r* are the parameters of the CA.
- CAs are often described as discrete dynamical systems with the capability to model various kinds of natural discrete or continuous dynamical systems

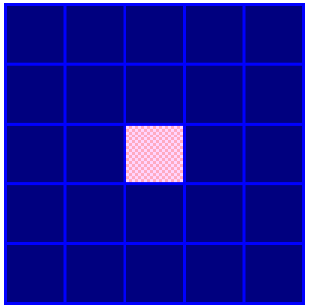
Types of Neighborhoods



von Neumann
neighbourhood



Moore
Neighbourhood



Extended Moore
Neighbourhood

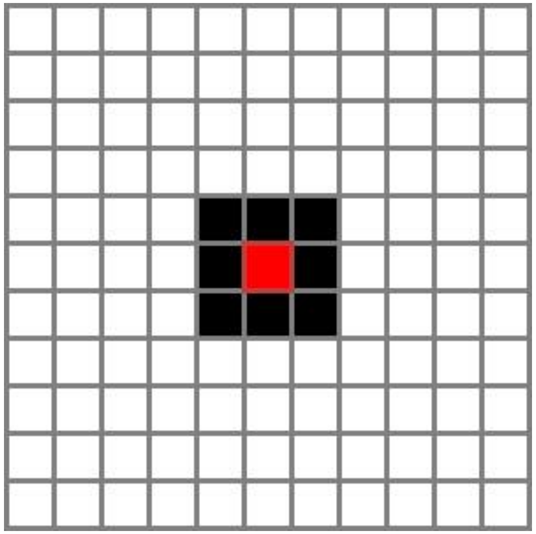
Many more neighborhood techniques exist!

Classes of cellular automata (Wolfram)

- Class 1: after a finite number of time steps, the CA tends to achieve a unique state from nearly all possible starting conditions (limit points)
- Class 2: the CA creates patterns that repeat periodically or are stable (limit cycles) – probably equivalent to a regular grammar/finite state automaton
- Class 3: from nearly all starting conditions, the CA leads to aperiodic-chaotic patterns, where the statistical properties of these patterns are almost identical (after a sufficient period of time) to the starting patterns (self-similar fractal curves) – computes 'irregular problems'
- Class 4: after a finite number of steps, the CA usually dies, but there are a few stable (periodic) patterns possible (e.g. Game of Life) - Class 4 CA are believed to be capable of universal computation

John Conway's Game of Life

- 2D cellular automata system.
- Each cell has 8 neighbors - 4 adjacent orthogonally, 4 adjacent diagonally.
- This is the Moore Neighborhood.

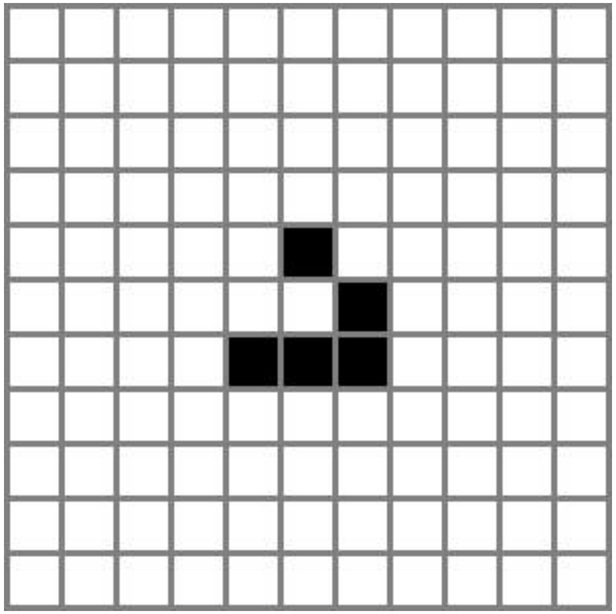


John Conway's Game of Life

- A live cell with 2 or 3 live neighbors survives to the next round.
- A live cell with 4 or more neighbors dies of overpopulation.
- A live cell with 1 or 0 neighbors dies of isolation.
- An empty cell with exactly 3 neighbors becomes a live cell in the next round.

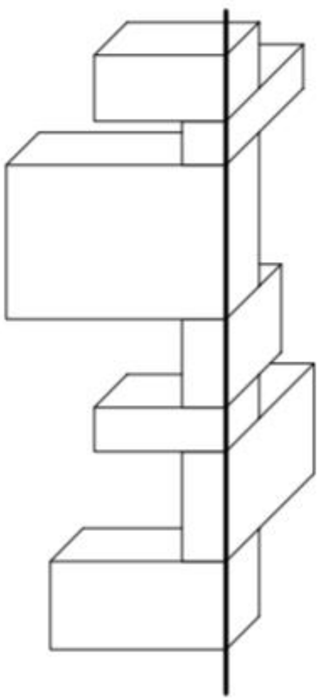
Is it alive?

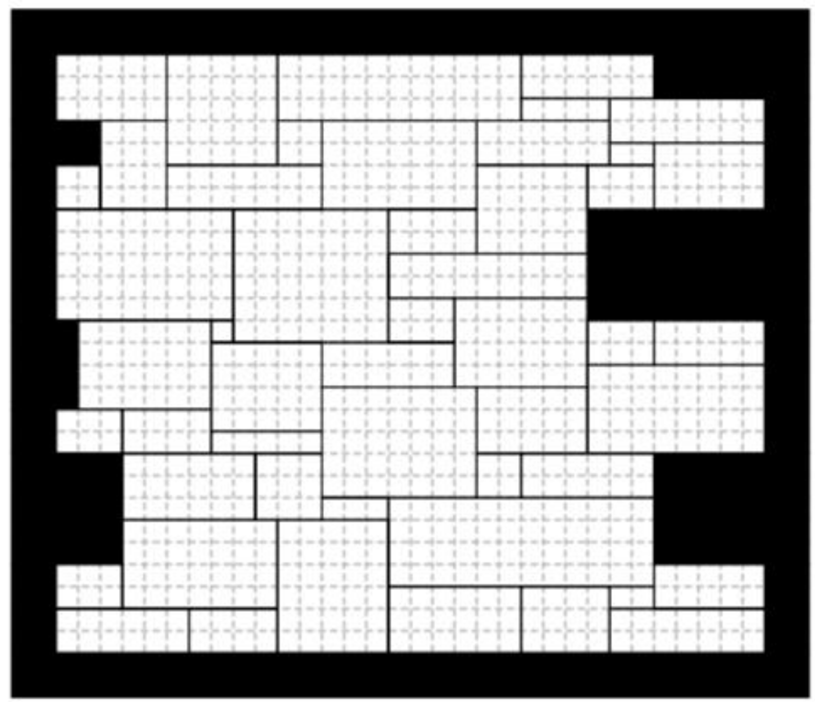
- http://www.bitstorm.org/gameoflife/
- Compare it to the definitions…

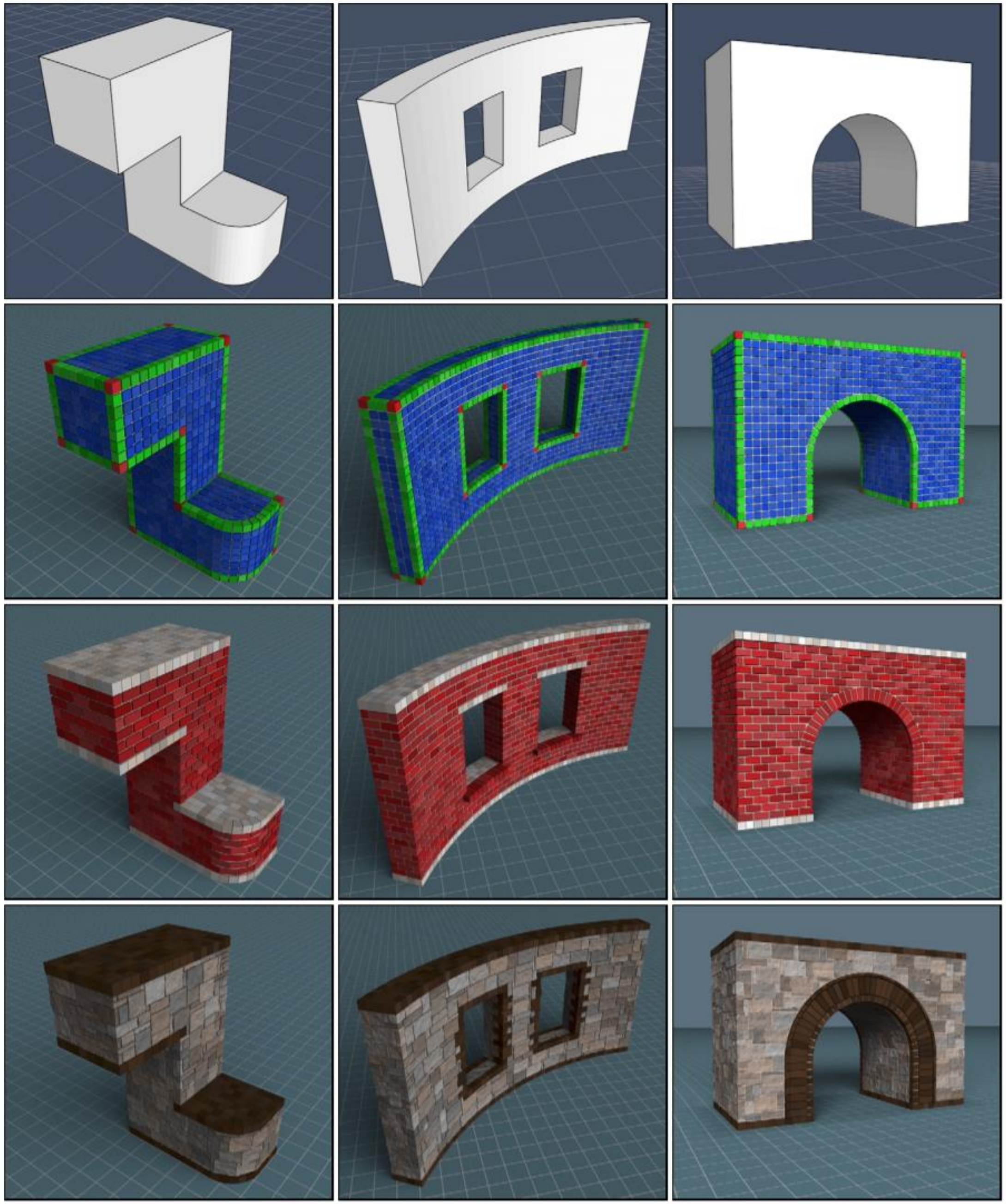


Cellular Automata

- Used in computer graphics:
  - Cellular Texturing







Urban Procedural Modeling

- Seminal paper:
  - "Procedural Modeling of Cities", Parish and Mueller, SIGGRAPH 2001

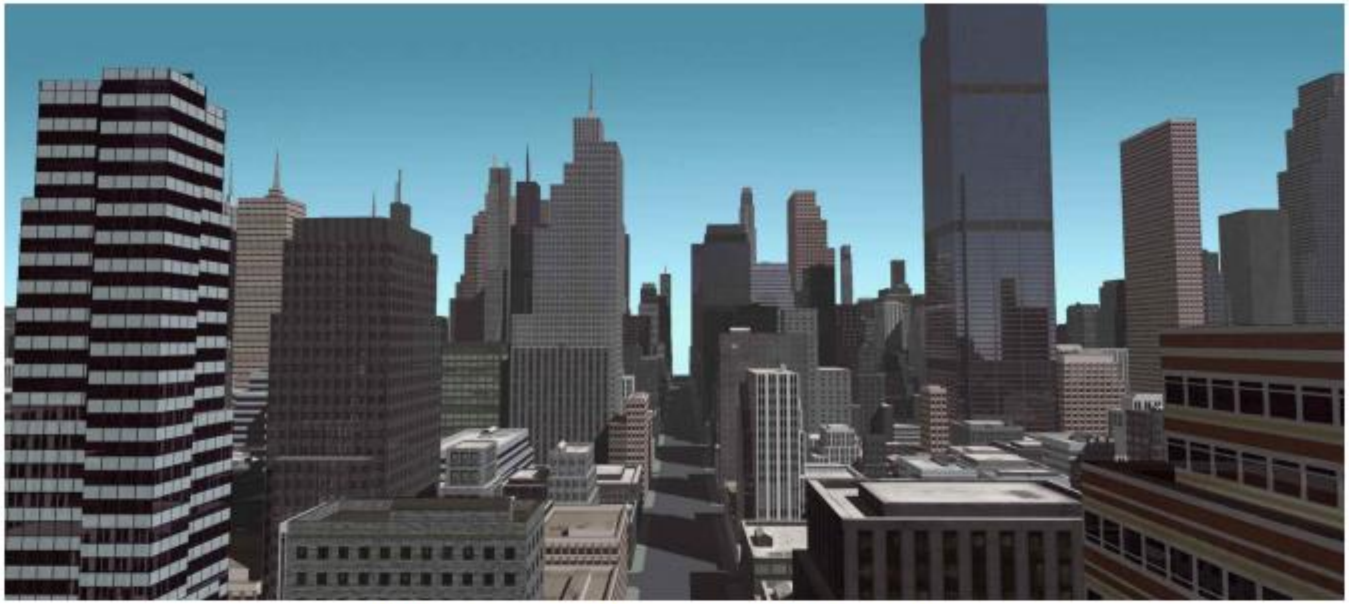


Figure 18. Somewhere in a virtual Manhattan.

Split Grammars

Instant Architecture

Peter Wonka[*,†] Michael Wimmer[†] François Sillion[‡] William Ribarsky[*]

[*]Georgia Institute of Technology [†]Vienna University of Technology [‡]INRIA

Figure 1: Left: This image shows several buildings generated with split grammars, a modeling tool introduced in this paper. Right: The terminal shapes of the grammar are rendered as little boxes. A scene of this complexity can be automatically generated within a few seconds.

Split Grammars

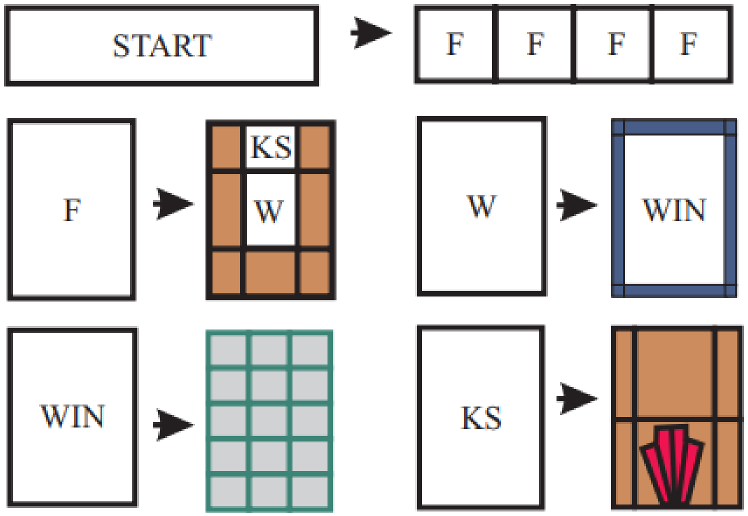


Figure 5: The rules for a simple example split grammar. The white areas (which contain symbols) represent the non-terminal shapes, colored elements are the terminal shapes of the split grammar. The start symbol is split into 4 façade elements, which are further split into a window element, a keystone element and some wall elements etc.

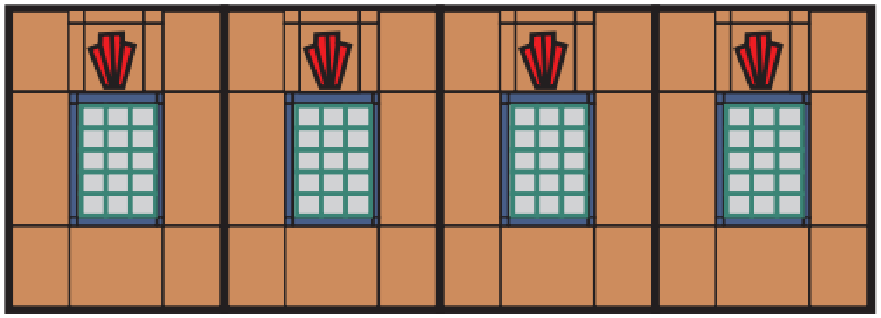
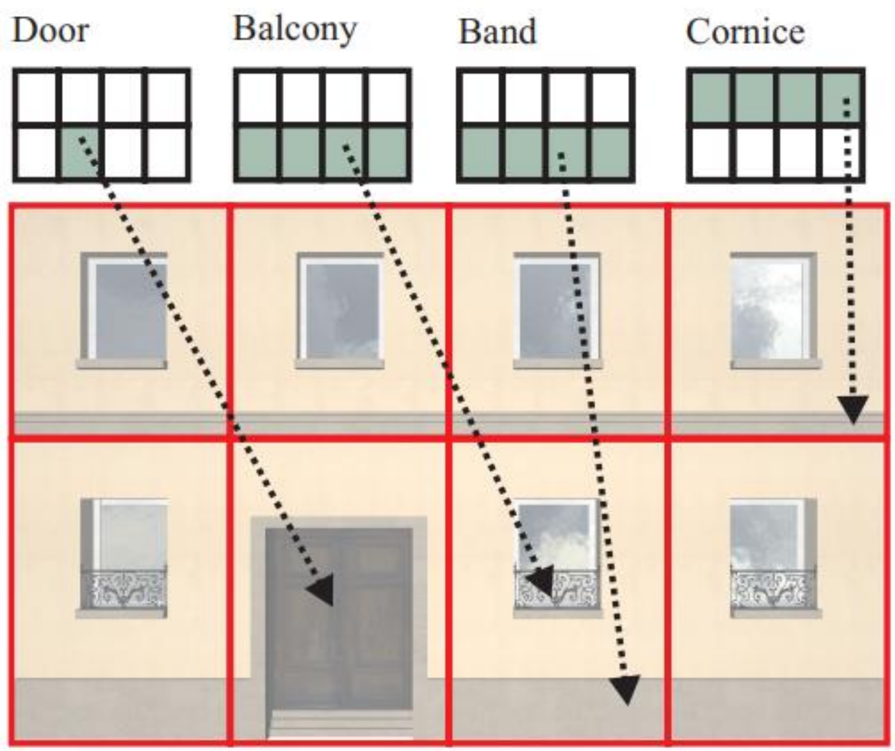


Figure 6: This figure shows the result of the derivation of the grammar in Figure 5.

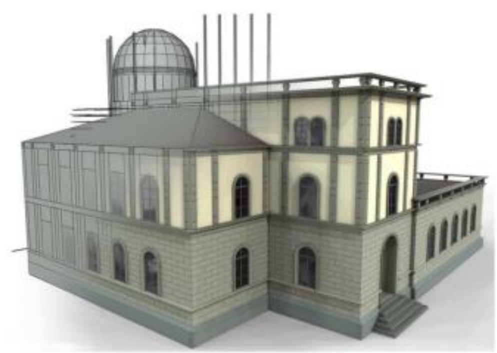
Split Grammars



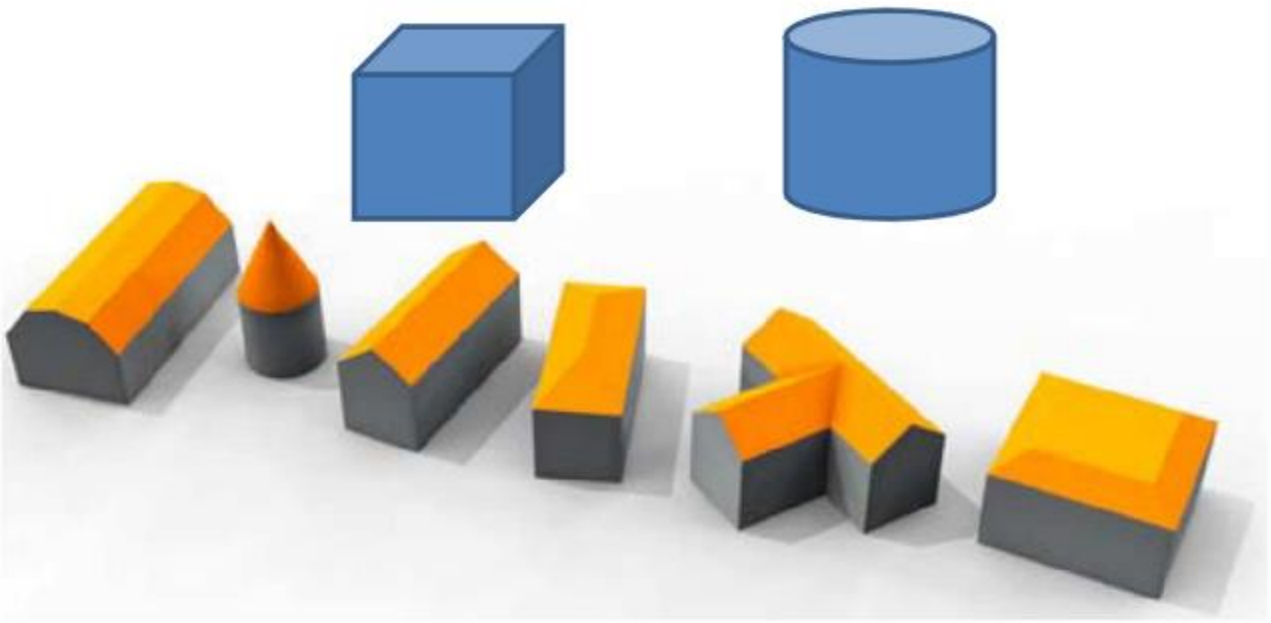

```
FACADE_CONTROL ::- DOOR_PATTERN, RANDOM_PATTERN,
                   RANDOM_PATTERN, RANDOM_PATTERN
RANDOM_PATTERN ::- CORNICE | BAND | BALCONY
                   | QUOIN | PILASTER
DOOR_PATTERN ::- DOOR | GARAGE
DOOR ::- <[0,1], door, 1> | <[0,2], door, 1>
         | <[0,MAX], door, 1>
```

CGA Shape Grammar

- P. Müller, P. Wonka, S. Haegler, A. Ulmer, L. Van Gool: Procedural modeling of buildings SIGGRAPH 2006

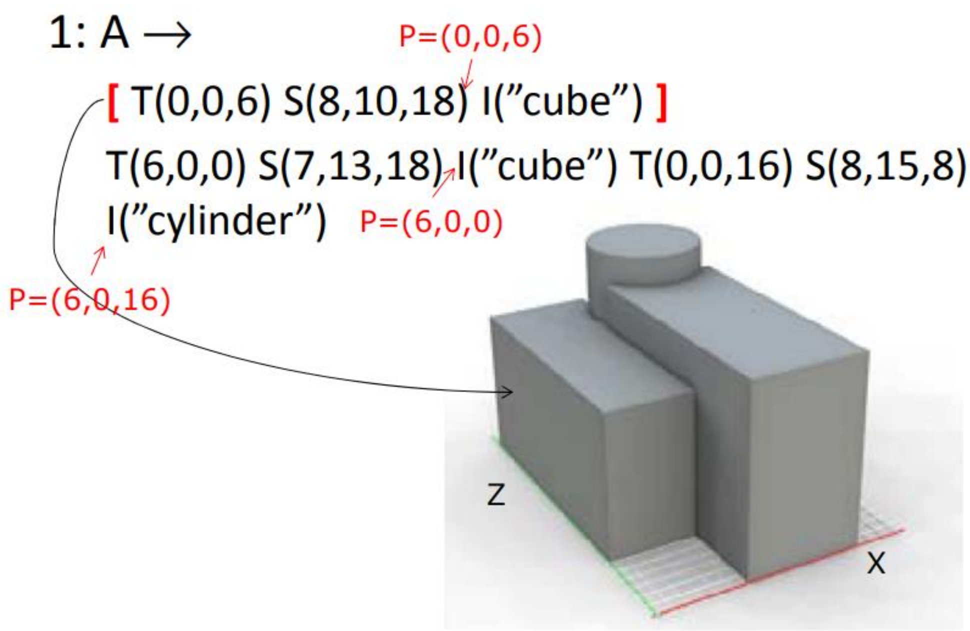


Basic Shapes



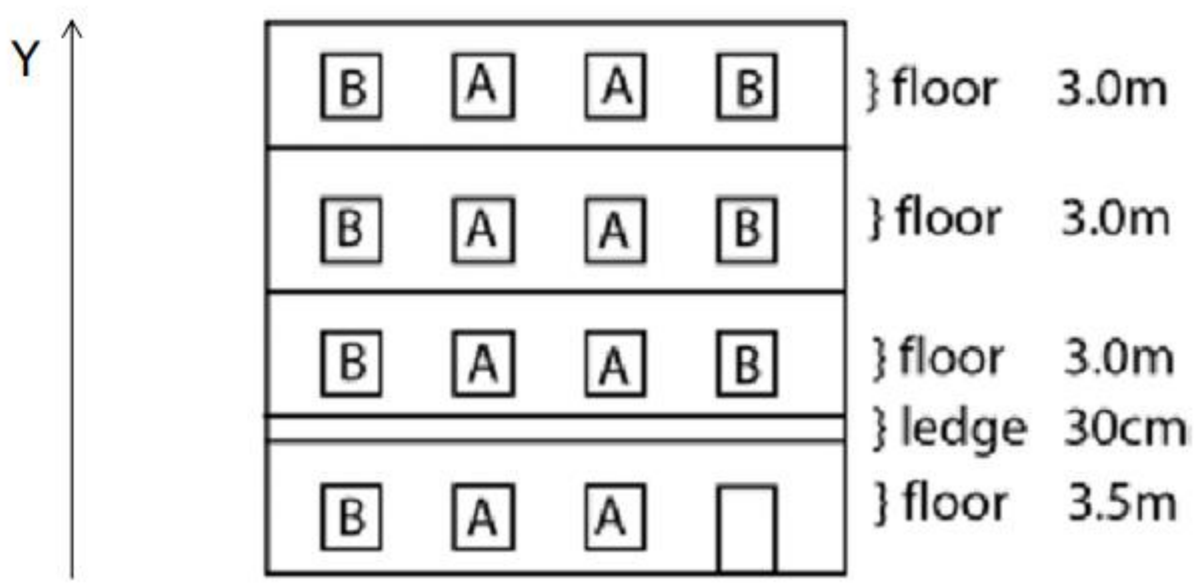
Rules and Operations

- T(x,y,z) = translate by [x y z]
- S(a,b,c) = scale by [a b c]
- Context (like [] in L-systems) =



Subdivision

1: fac $\rightsquigarrow$ Subdiv("Y",3.5,0.3,3,3,3){ floor | ledge | floor | floor | floor }



Examples

PRIORITY 1:

1: lot ⇝ S(1r,*building_height*,1r)
Subdiv("Z",Scope.sz*rand(0.3,0.5),1r){ facades | sidewings }

2: sidewings ⇝
Subdiv("X",Scope.sx*rand(0.2,0.6),1r){ sidewing | ε }
Subdiv("X",1r,Scope.sx*rand(0.2,0.6)){ ε | sidewing }

3: sidewing
⇝ S(1r,1r,Scope.sz*rand(0.4,1.0)) facades : 0.5
⇝ S(1r,Scope.sy*rand(0.2,0.9),Scope.sz*rand(0.4,1.0)) facades : 0.3
⇝ ε : 0.2

4: facades ⇝ Comp("sidefaces"){ facade }

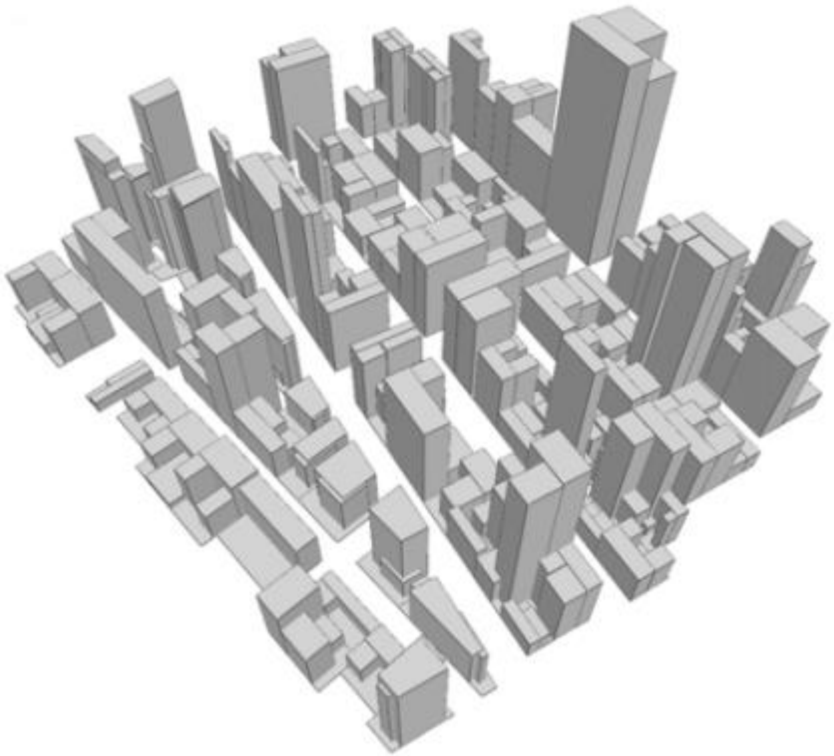


Figure 14: Stochastic variations of building mass models generated with only four rules (starting with the building lot as axiom).

Examples

```
PRIORITY 2:
5:  facade : Shape.visible("Street") == 0 ⤳
      Subdiv("Y",groundfloor_height,1r,topfloor_height)
        { groundfloor | floors | topfloors } firescape
6:  groundfloor ⤳ Subdiv("X",1r,entrance_width,1r){ groundtiles |
      entrance SnapLines("Y","entrancesnap") | groundtiles }
PRIORITY 3:
7:  facade ⤳ floors
8:  floors ⤳ Repeat("YS",floor_height){ floor Snap("XZ") }
9:  floor ⤳ Repeat("XS",tile_width){ tile Snap("Y","tilesnap") }
...
15: wall : Shape.visible("Street") ⤳ I("frontwall.obj")
PRIORITY 4:
16: fireescape ⤳ Subdiv("XS",1r,2*tile_width,7r,"tilesnap")
      { epsilon | escapestairs | ε }
17: escapestairs ⤳ S(1r,1r,firescape_depth)
      T(0,0,-firescape_depth) Subdiv("YS",groundfloor_height,1r)
        { ε | Repeat("YS",floor_height){ I("firescape.obj") } }
```

Figure 15: A procedurally generated building modeled with snap lines. Note the alignment of important lines and planes in the construction.

Urban Procedural Modeling

- Cities
- Buildings
- CityEngine
  - CityEngine
  - https://www.youtube.com/watch?v=xJCIIE9pulk
  - (for Unreal: https://www.youtube.com/watch?v=faOdiVcxRG4)

Videos and more

- Procedural Modeling of Cities
  - http://www.youtube.com/watch?v=khrWonALQiE
- Procedural Modeling of Buildings
  - http://www.youtube.com/watch?v=iDsSrMkW1uc
- Procedural Modeling of Structurally Sound Masonry Buildings
  - http://www.youtube.com/watch?v=zXBAthLSxSQ
- Image-based Procedural Modeling of Facades
  - http://www.youtube.com/watch?v=SncibzYy0b4
- Image-based Modeling
  - Facades: http://www.youtube.com/watch?v=amD6_i3MVZM