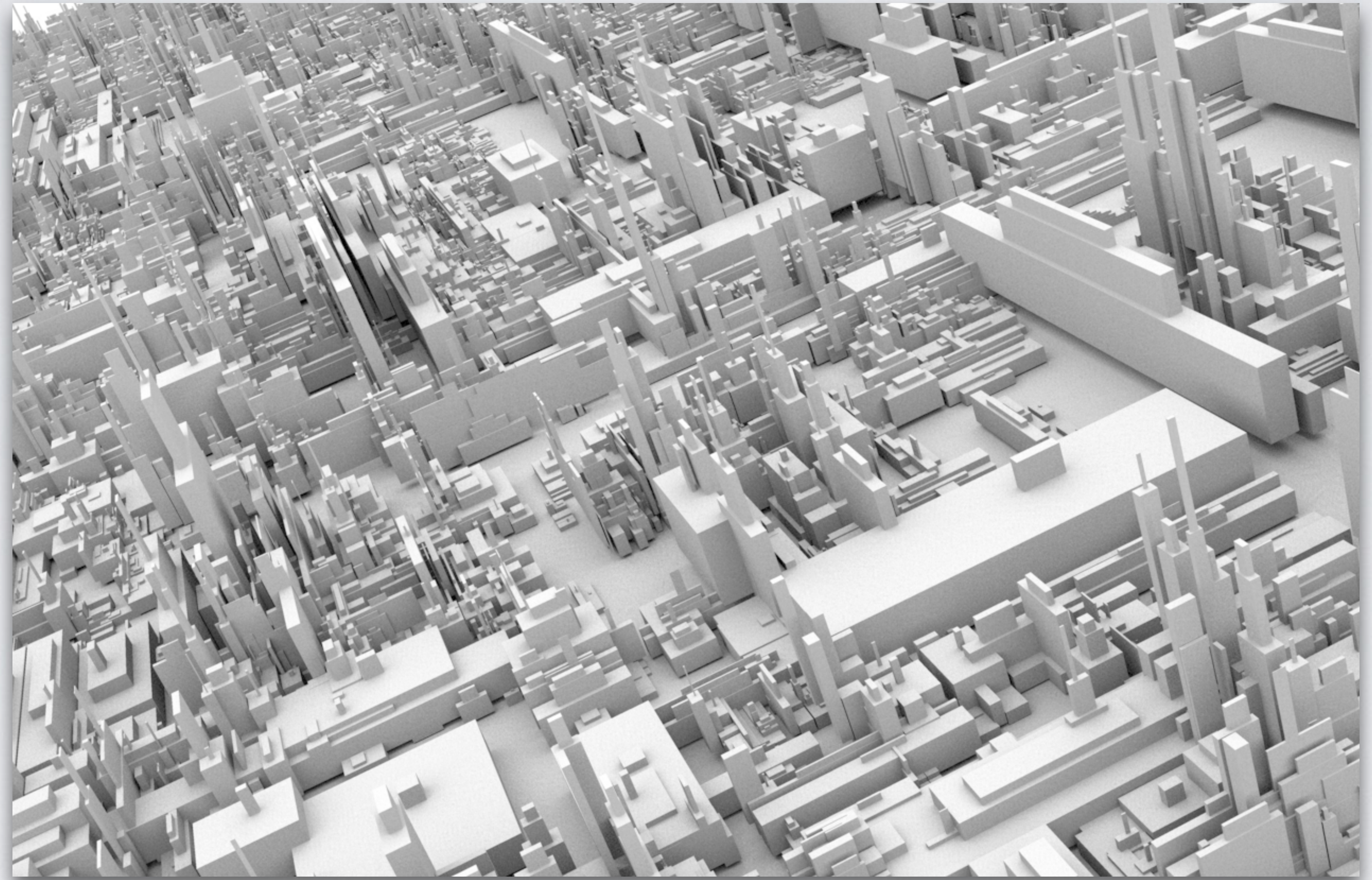
SHAPE GRAMMARS

procedural generation techniques for virtual cities



Sebastien Parodi (source)

URBAN ENVIRONMENTS

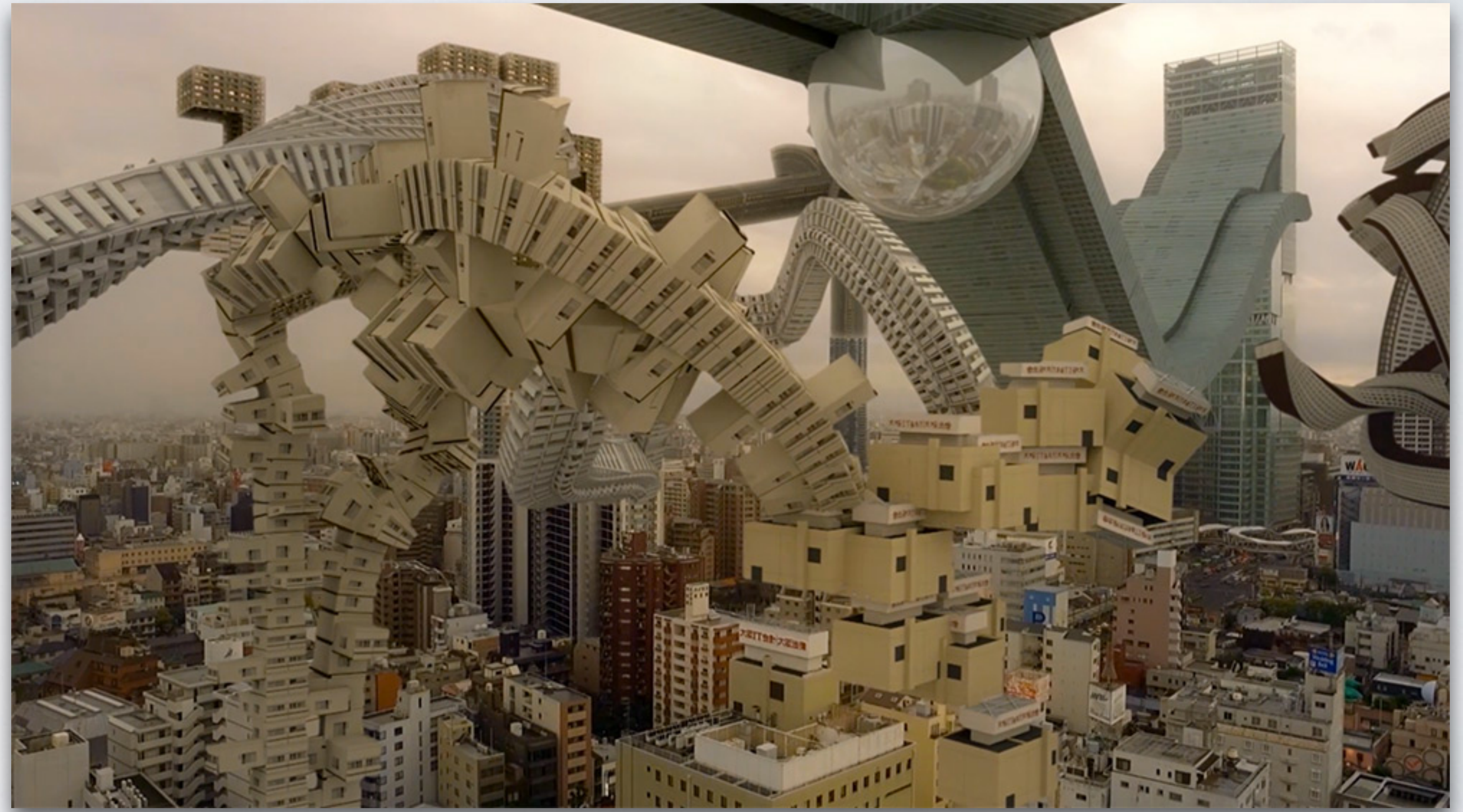
Shinjuku Piano (source)

- High demand for city or building assets in film/games
- Luckily, lots of repetition, both in a single building and in architectural styles
  - Basic geometry / footprint
  - Structural elements. eg windows
  - Decorative elements, eg ledges
  - Textures / materials
- Repetition? Sounds like a procedural generation problem!

PROCEDURAL CITIES!

- With just a bit of code, we can have infinite cites, living architecture, realistic OR fantastic environments

  - whoa
  - whoa
  - whoa
  - whoa



AUJIK (source)

BUILDING BUILDINGS

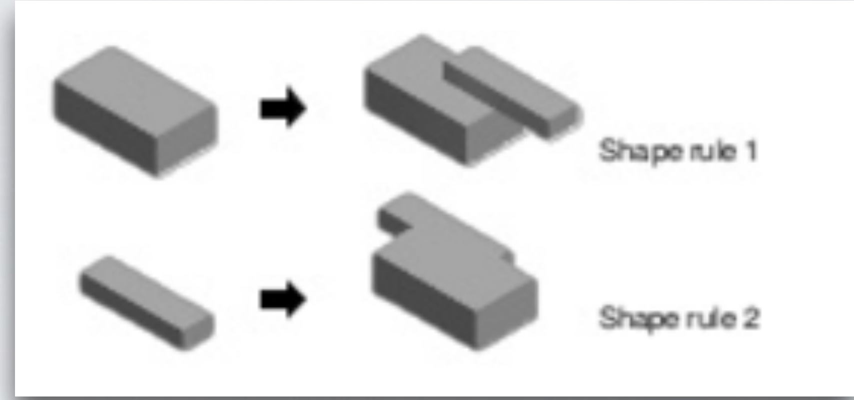
THE APPROACH

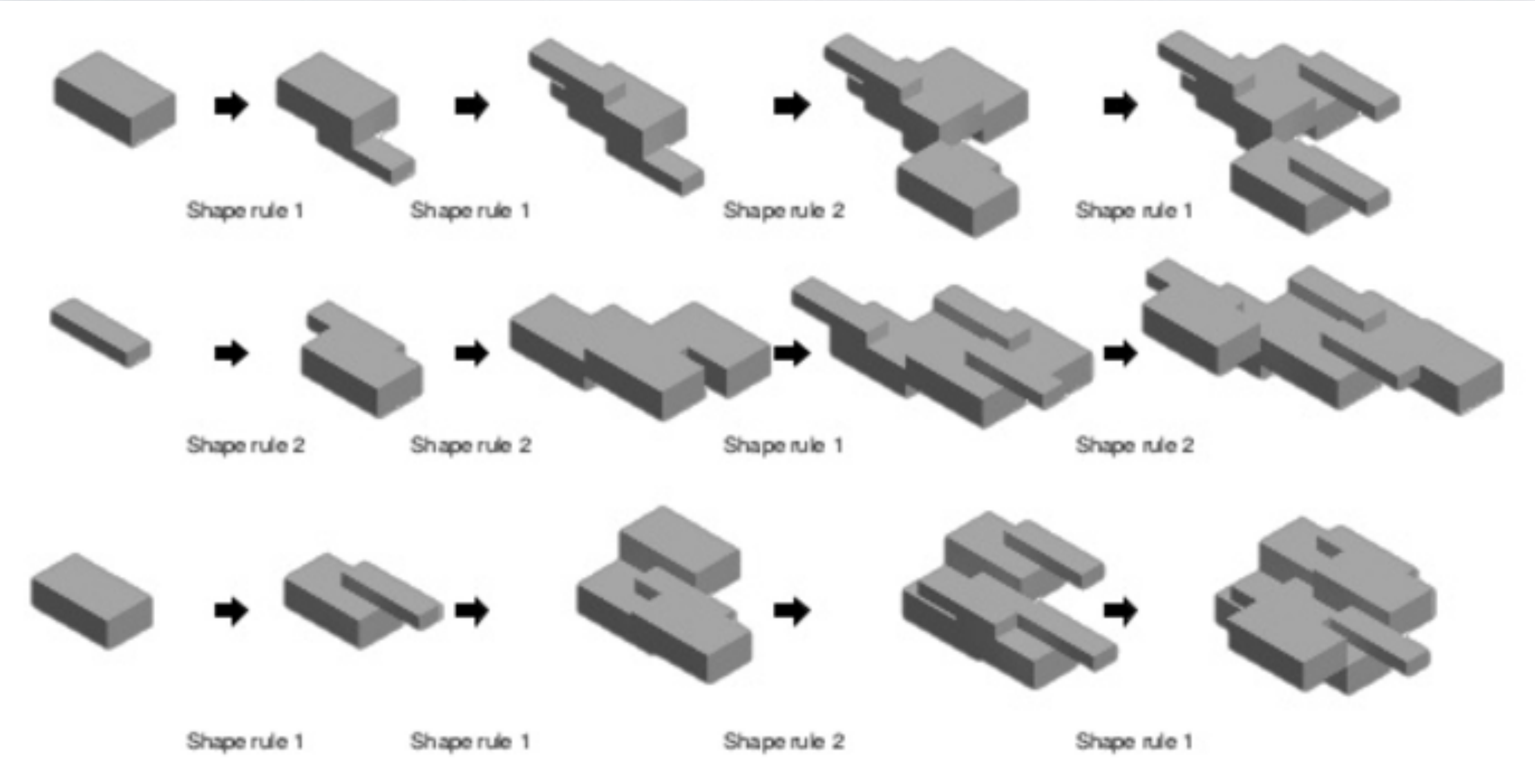
- Repeated elements in specific contexts with some structured variation… sounds very familiar.
- Just like l-systems!
- Start by identifying the basic building blocks

Procedural Modeling of Buildings (source)

THE GEOMETRIC BASE

- Some example shape grammar rules





Mahd Adib Ramli (source)

SHAPE GRAMMAR

- Shape grammars are almost like classic l-system grammars.
- But the grammar production process and rendering instructions are more intertwined.
    - Symbols have numeric attributes, eg. position, scale
    - Successors are computed, based on the numeric attributes of their predecessor, not just predetermined.
    - Since transformation information is usually stored, symbol ordering is not necessarily important

SHAPE GRAMMARS

Symbol = {terminal, non-terminal}

Shape = {symbol, geometry, numeric attributes}

Rule = {predecessor, successor = f(predecessor), probability}

1. Begin with some configuration of shapes (like an l-system axiom)
2. Select an shape S from set.
3. Choose a production rule with S as predecessor, compute successor(s) SNEW, add to set.
4. Remove S from the set.
5. Repeat until all shapes in the set are terminal.

EXAMPLE RULE

Replace a floor with a row of wall/window units. Demo.

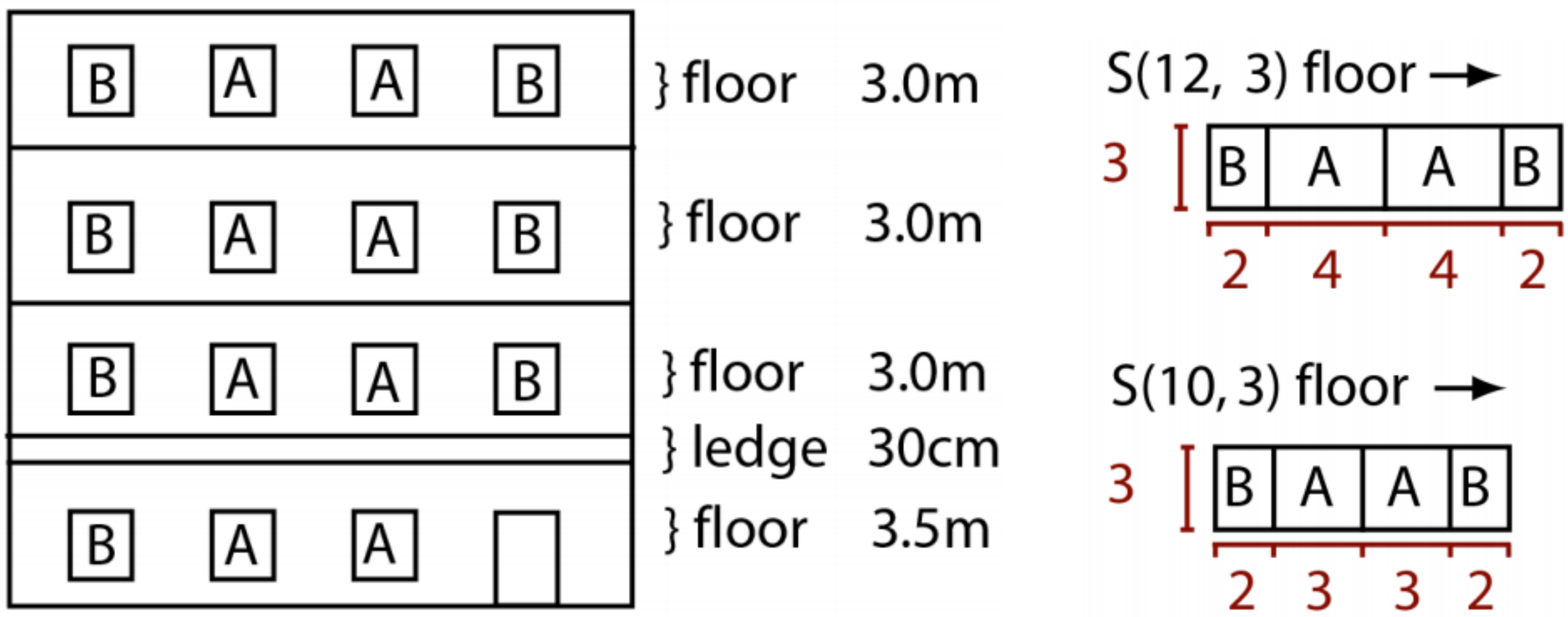


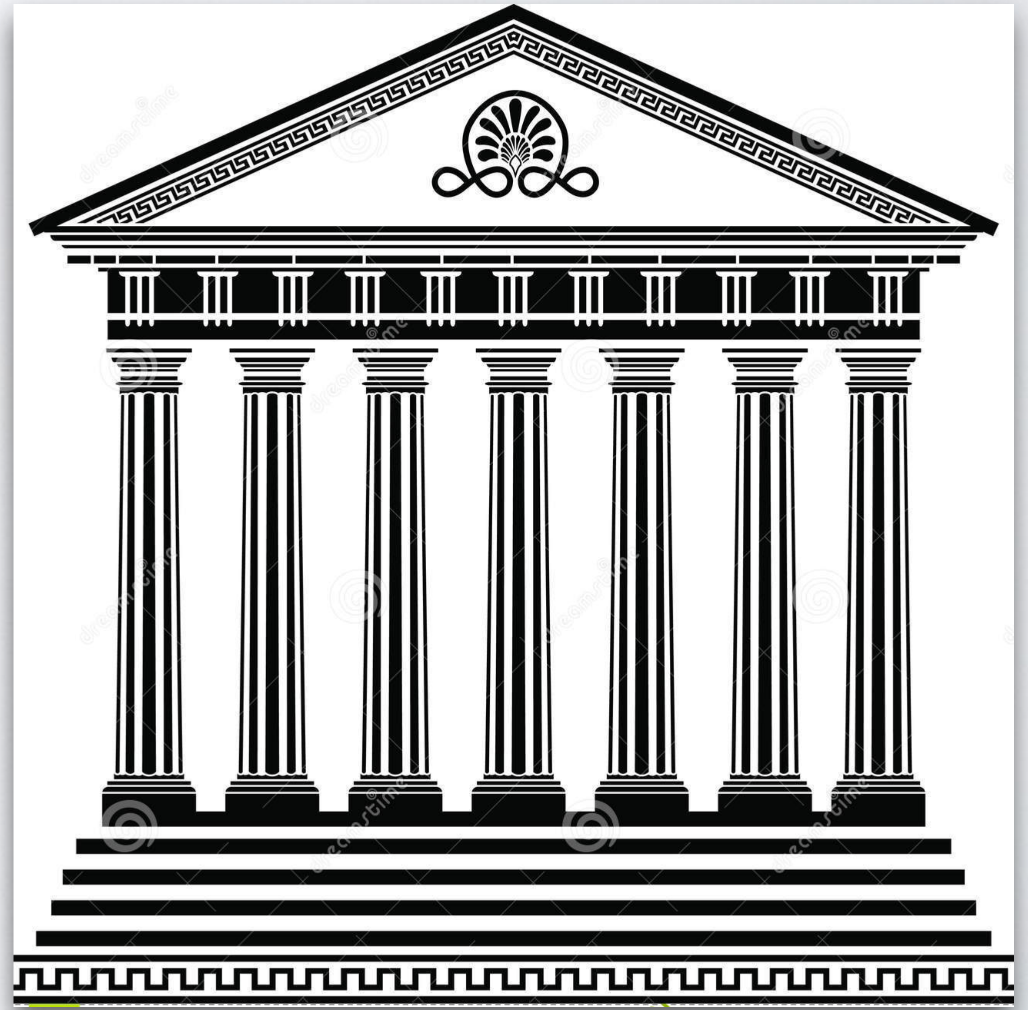
Figure 4: Left: A basic façade design. Right: A simple split that could be used for the top three floors.

Procedural Modeling of Buildings (source)

EXAMPLE SYSTEM

Describing a simple building with some basic rules.

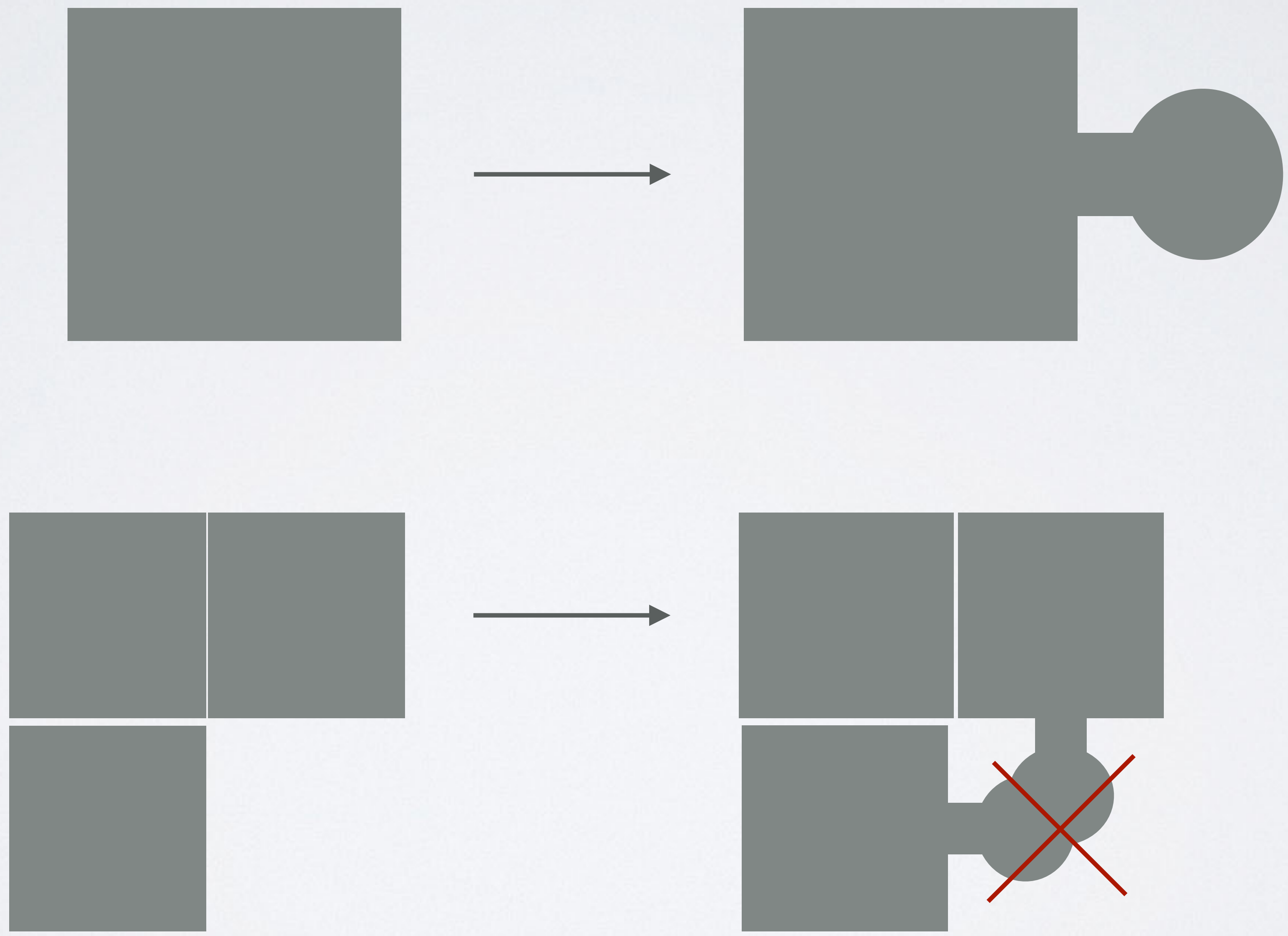
- temple -> Subdiv("Y", …, … } { podium | columns | roof }
- column -> Subdiv("Y", …){ base | shaft | capital }
- columns -> Repeat("X", …){ column }
- base -> (corinthian_base)
- shaft -> (corinthian_shaft)
- capital -> (corinthian_capital)
- podium -> (podium)
- roof -> (roof)



Dreamstime (source)

INTERSECTION ISSUES

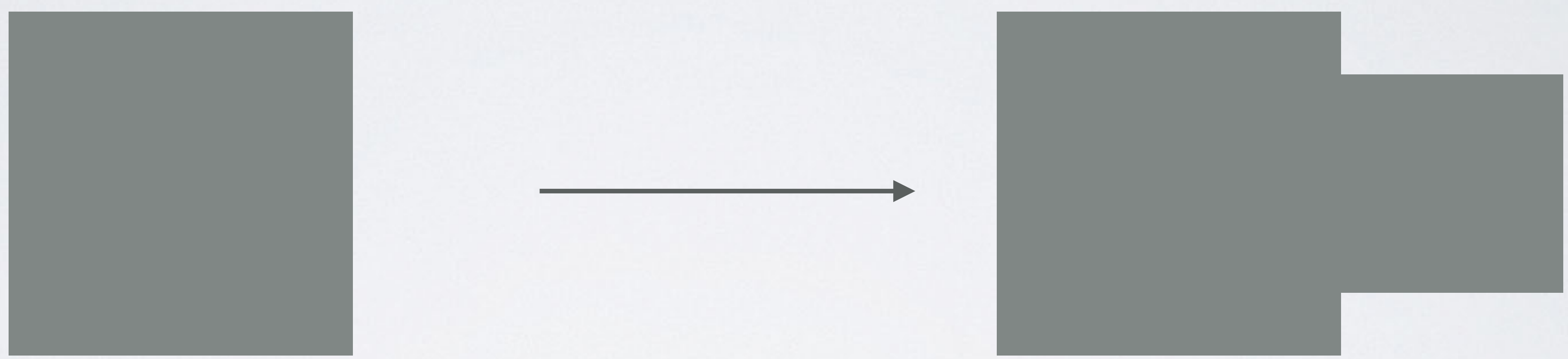
- Adding a lot of geometry may create undesirable intersections. Eg. rule:



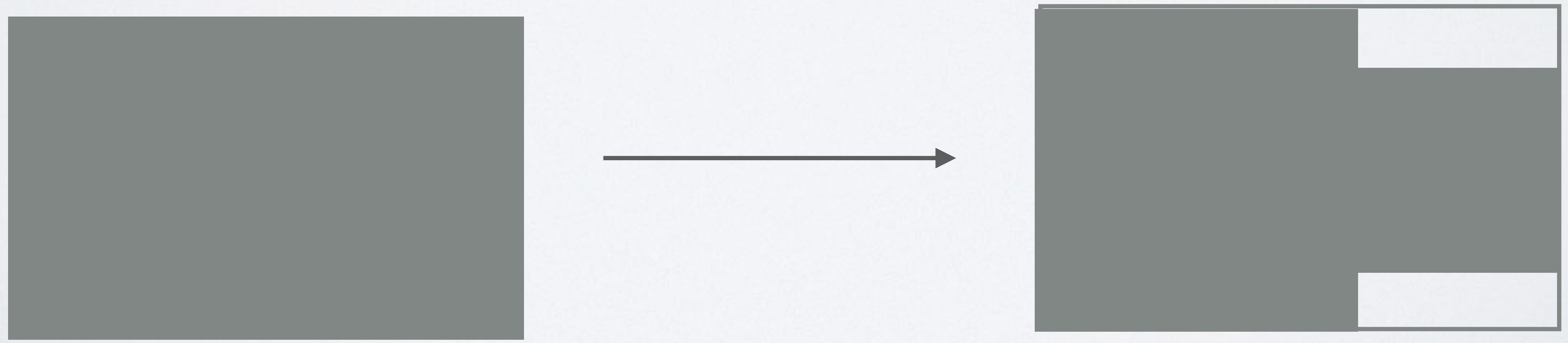
INTERSECTION PROBLEMS

Two basic grammar strategies

Additive - use an oct-tree to keep track of occupied space

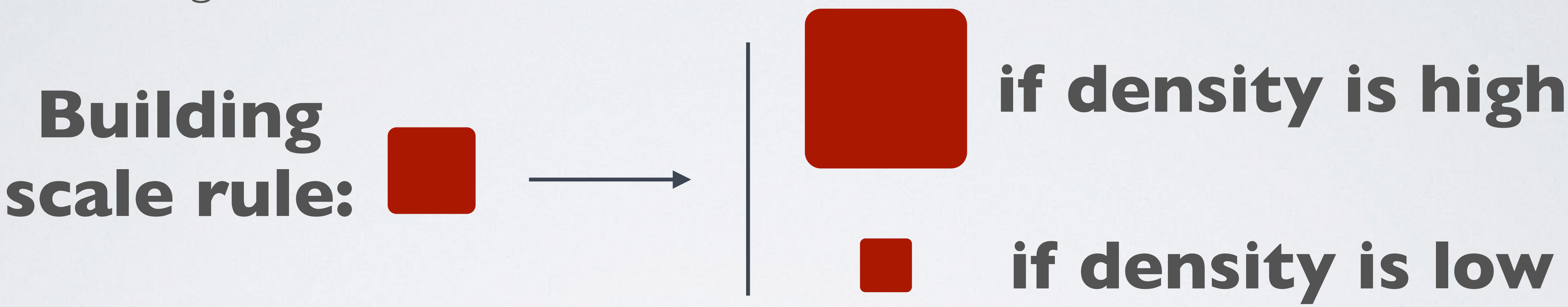


Subtractive - Geometry only shrinks. No intersections!

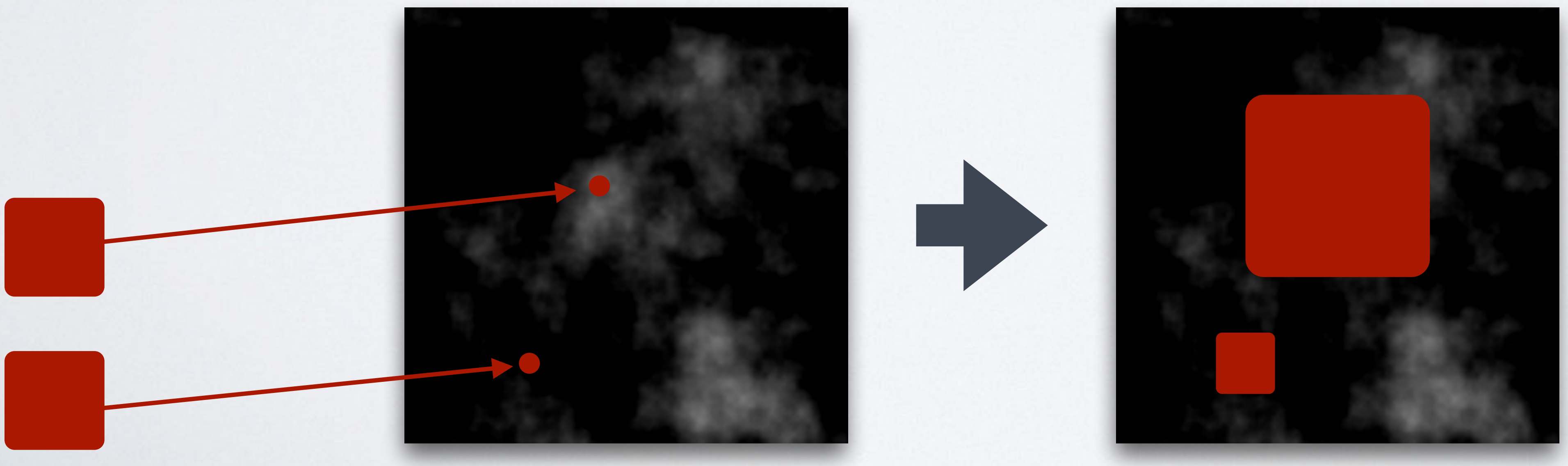


CONDITIONAL RULES

- We can add conditional attributes and/or use grammar external data.
- For example, let's say noise value on terrain corresponds to population density. White = high, black = low



In application:



ENCODING DESIGN

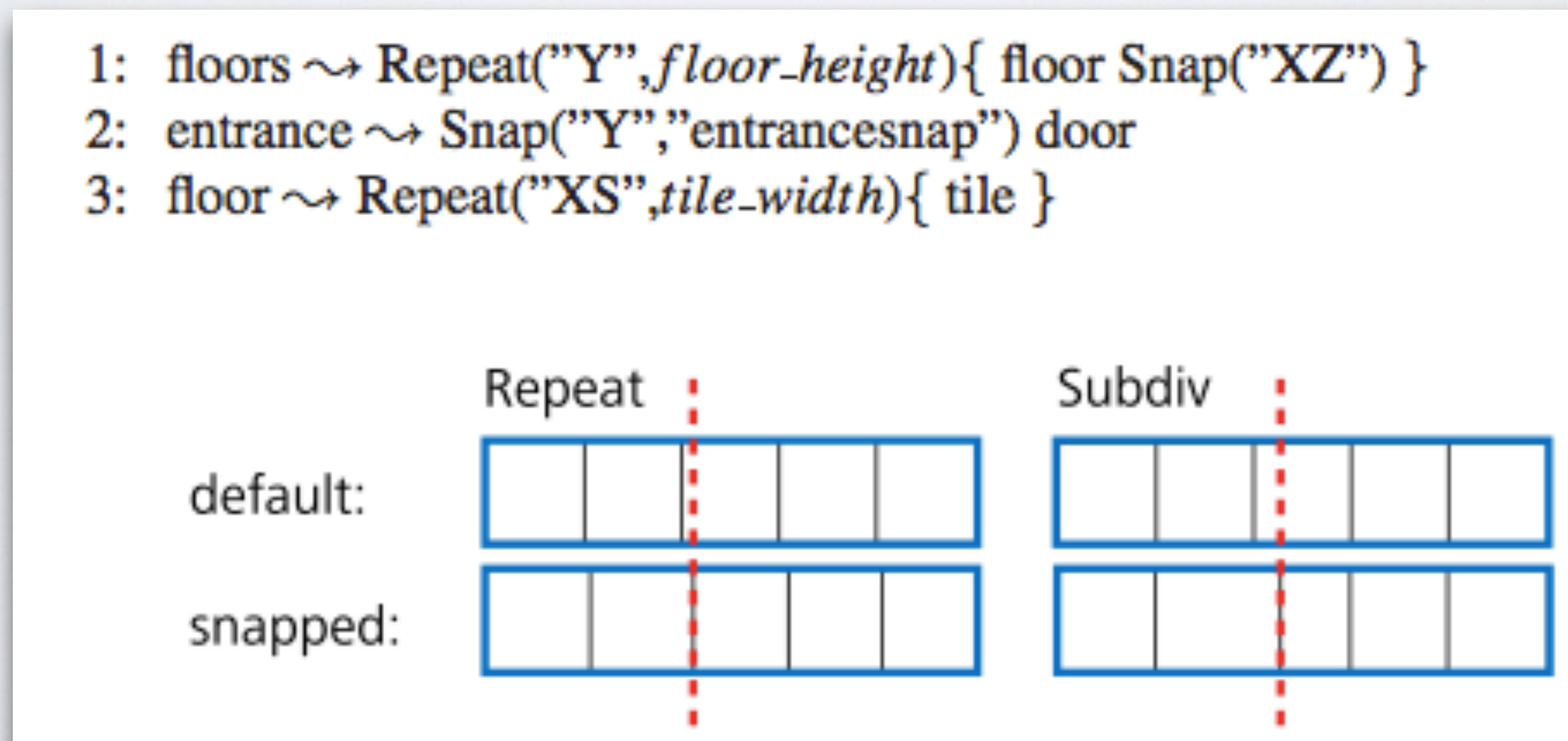
*(No disrespect to this hotel.) Frank Gehry (source)

- Many rules applied haphazardly create chaos!
- Mimicking artificial structures can be trickier than organic structures, since artificial structures must look designed. Consider:
    - Symmetry, guide-lines, whether your structure looks functional

SNAP LINES

- The creators of CityEngine observed the chaos issue, and introduced the concept of snap lines.

- Idea: enforce order by "snapping" divisions/splits to specific lines.



Procedural Modeling of Buildings (source)

SIMPLE HOUDINI EXAMPLE

Painfully-created by Austin Eng and Rachel in Houdini Python

CITY LAYOUTS

HOW TO GROW A CITY?

- Complex layers of related detail!
  - Layout
  - Building distribution
  - Streets vs Highways
- How do we start?
  - Well, buildings usually depend on function.
  - Which depends on neighborhood
  - Which depends on street map
  - Which depends on layout
  - Which depends on geography

Subversion (source)

GENERAL APPROACH

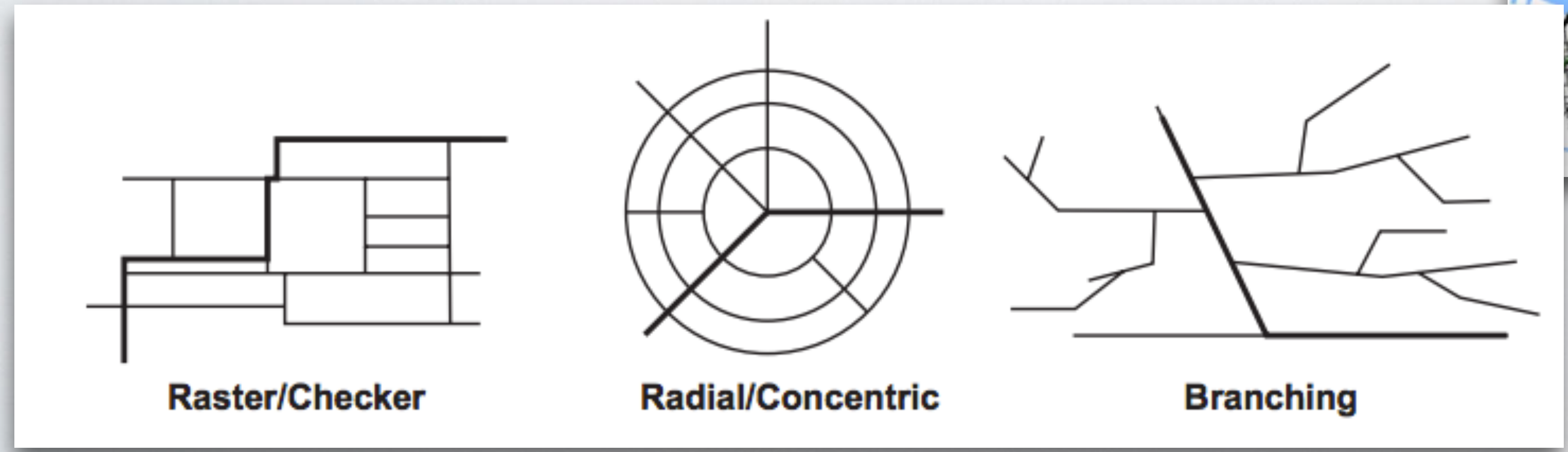
one of many possible!

Procedural Modeling of Cities (source)

1. Generate terrain
2. Generate grammar-based roads, potentially terrain-sensitive
3. Use roads to divide the area into blocks, then blocks into individual building lots
4. For each building lot, generate an appropriately-sized building.

CITY LAYOUTS

- We can use a modified version of l-systems to generate road maps
- Many viable strategies, eg
    - draw rings around dense areas
    - connect dense areas
    - create square blocks or various dimensions



Procedural Modeling of Cities

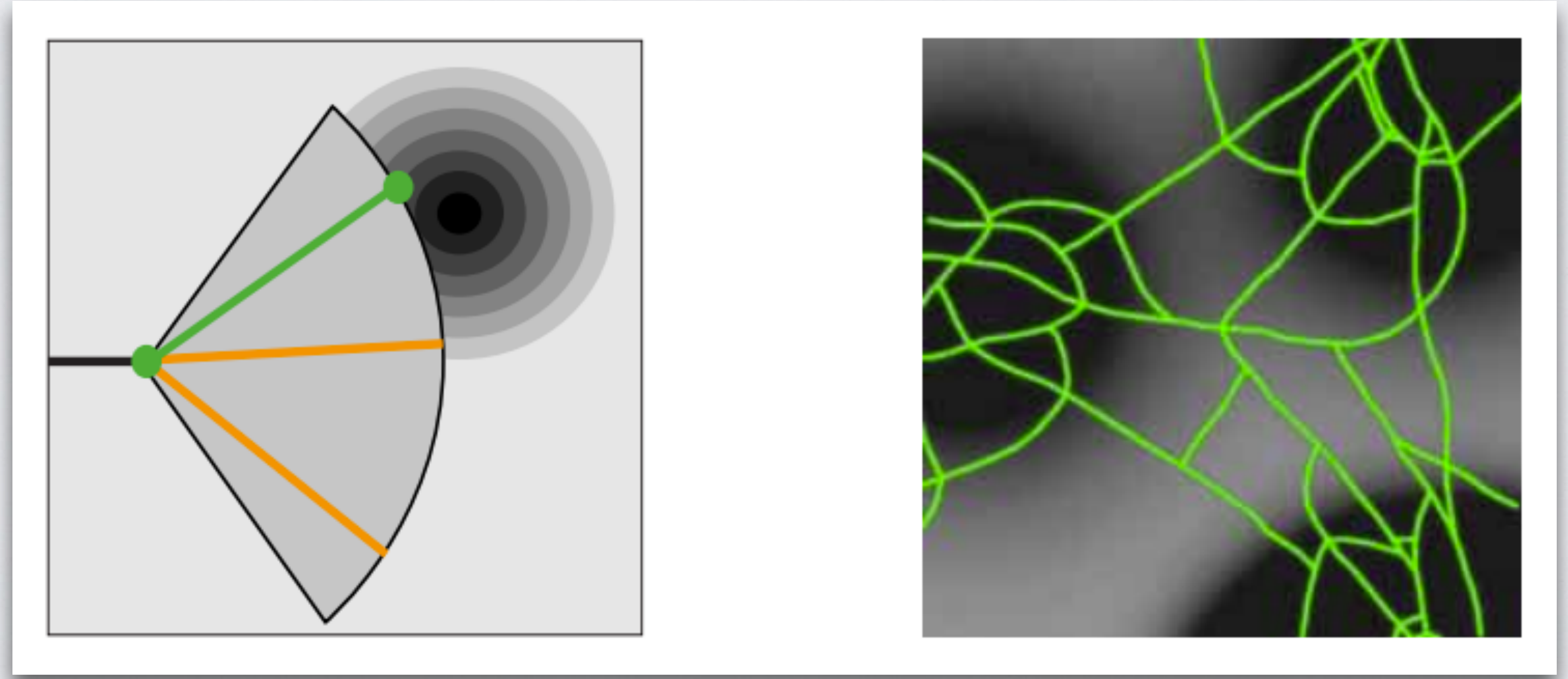
(source)

L-SYSTEMS EXTENDED

- To make our roads geography-conscious, as in shape grammars, we can make our l-systems context-sensitive.

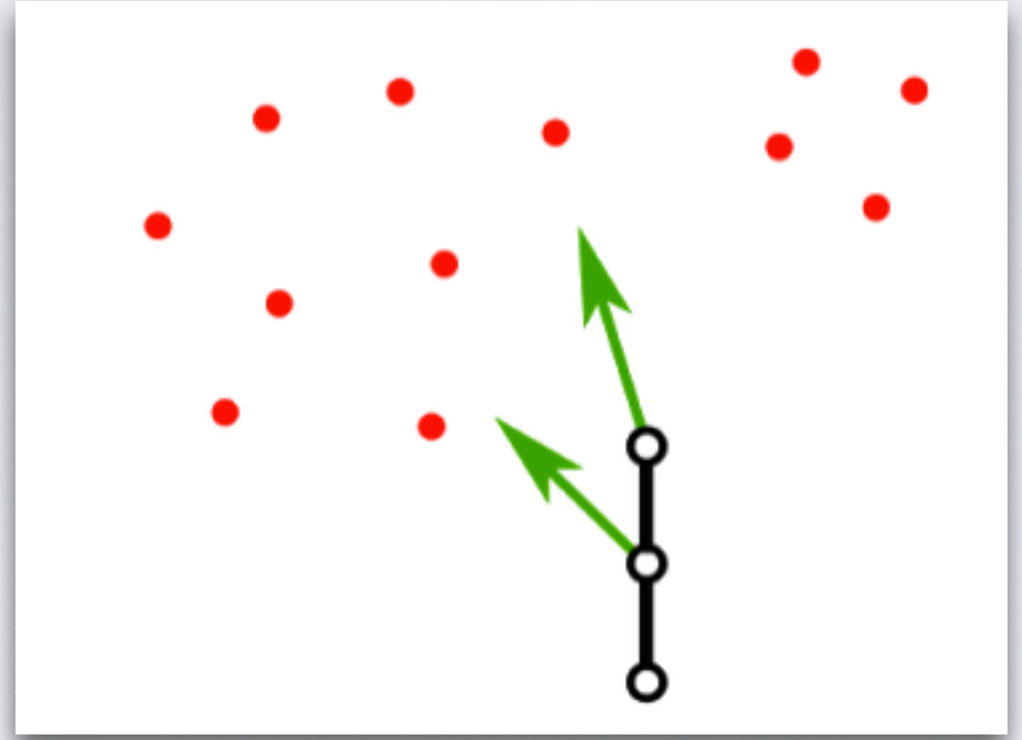
- Rather than just branching off, we can *propose* a branch (or a set of possible branching within a range of values), then modify it based on context.

- For example, roads should always preferentially point towards high-density areas.



Procedural Modeling of Cities (source)

ANOTHER APPROACH

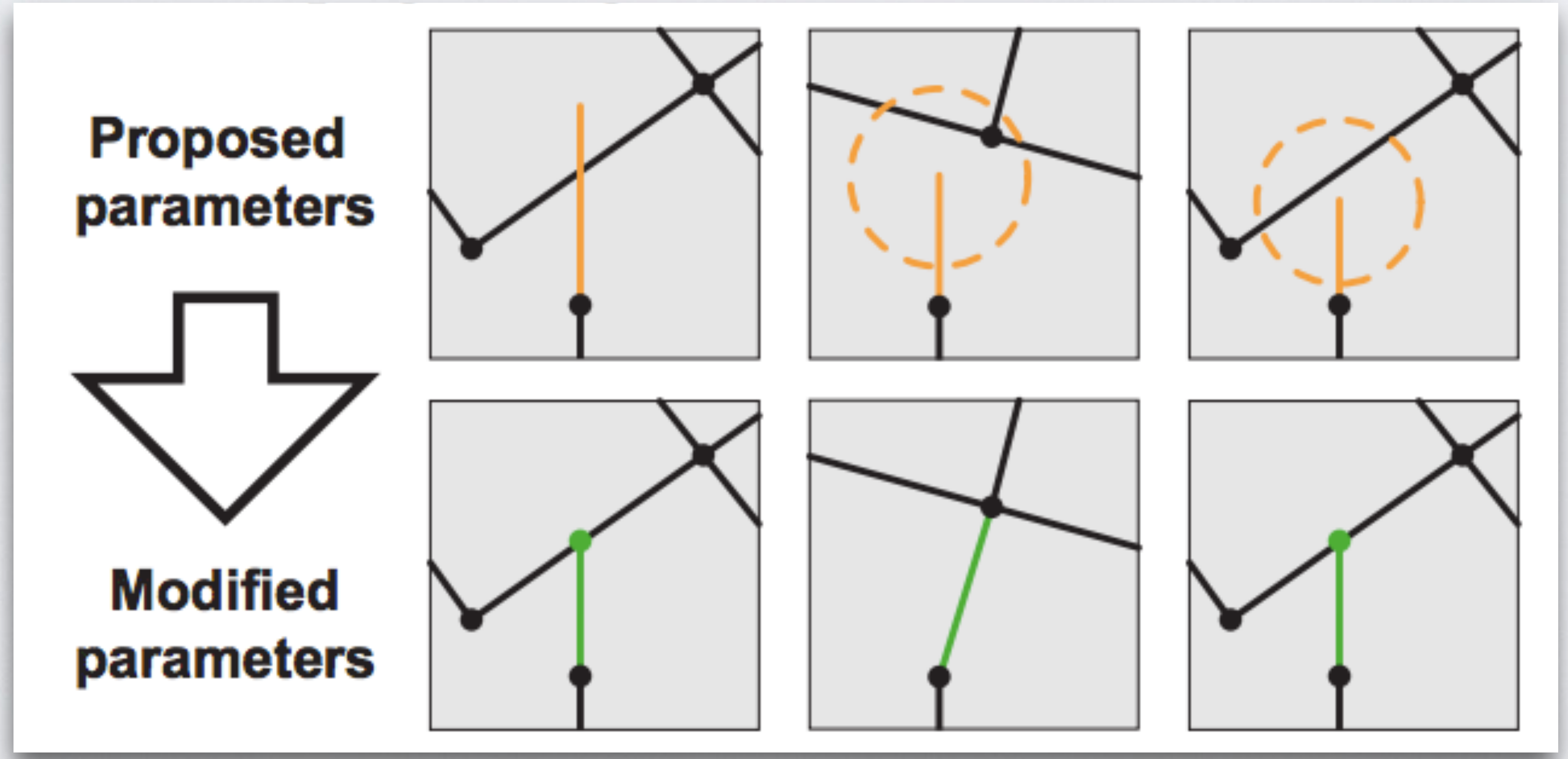


Procworld (source)

- Rather than bothering with extended l-systems, we can use a space colonization approach
- Scatter points using some algorithm, then try to connect them.

SELF-SENSITIVITY

- As with building generation, we want to create the illusion of deliberate design
- With roads, we can track previously generated paths and modify subsequent road additions to enforce order.



Procedural Modeling of Cities (source)

AND IT WORKS!

- Several powerful commercial tools available.
- Such as CityEngine

Esri (source)

IMPLEMENTATION

SHAPE SYMBOLS

- Suggested implementation pseudo-code:
  - Shapes have to store geometric and transformation data, since it's computed based on its predecessor
  - Order no longer matters though, so a set is fine

```
class Shape {
  char symbol;
  geometry_type g_type;
  vec3 position;
  vec3 rotation;
  vec3 scale;
  bool terminal;
};
```

THE PARSER

- Suggested implementation pseudo-code:
  - Basically just like l-systems, with a simple render step afterwards
  - The render step basically just adds the specified geometry. No turtle!

```
// Apply rules to all shapes in our shape set for n interations
ShapeSet parseShapeGrammar(ShapeSet shapes, RuleList grammar, int iterations) {
  for (int n=0; n < iterations; ++n) {
    for (shape s : shapes) {
      // s is not a terminal symbol
      if (!s.terminal) {
        // Apply a rule to get successor of s
        ShapeSet successors = applyRandomRule(s, grammar);
        // Remove old shape
        shapes.remove(s);
        // Add new shapes
        shapes.add(successors);
      }
    }
  }
  return shapes;
}

render(shapes);
```

IN SUMMARY

- Shape grammars (similar to l-systems)
    - Symbols have numeric attributes used in rules
    - Rules compute successors instead of just replacing symbols deterministically.
    - Rules can use data to further parameterize generated successors.
- Modeling artificial structures is harder than organic because output needs to look designed
- Cities are complicated. We can model this complexity by modeling layers of influential features.
    - We can carve the city into pieces using a road map
    - Then generate building in the lots between the roads

REFERENCES

- Papers
    - Procedural Modeling of Buildings
    - Procedural Modeling of Cities
    - Subversion building generation
    - Citygen
- Helpful articles
    - Demo of street generation
    - Interesting critique of the CityEngine road approach
    - Good discussion on street network generation

ASSIGNMENT

Pixel City (source)

- Generate a procedural city (or town, or village) populated by procedural buildings
  - Buildings must vary in structure and decor
  - Buildings must be placed along procedural roads in a meaningful way
  - Buildings/roads must be "context-sensitive" in some way. eg. neighborhoods
  - Simple example (NOT a good completion).