Making Stuff: Synthesis, Generative Modeling, and Procedural Modeling

CS535

Daniel G. Aliaga
Department of Computer Science
Purdue University

Making Stuff

- Synthesis/Generation/Procedural-Modeling/Content-Creation/Model-Creation
  - Manual Modeling (most prevalent approach):
    - Softimage (Microsoft+SoftImage3D, 1989-2002)
    - Maya (SGI+Alias+Wavefront, 1998-today)
    - Blender (open source, 1995-today)
    - Houdini (SideFX, 1996-today)
    - Rhino/Grasshopper (TLM, 1978? – today)
    - And more…
- Problem: TIME CONSUMING!
  - E.g., 3-20 days for a photorealistic character

Making Stuff

- **Texture Synthesis**
  - **Make/replicate small fragments**
- Image Synthesis, Generative 2D Images, Procedural Modeling
  - Make 2D images
- Model Synthesis, Generative 3D Modeling, Procedural Modeling
  - Make 3D models

Synthesis by Tiling

- Simple tiling (next slides)
- Wang tiles (~1960s)

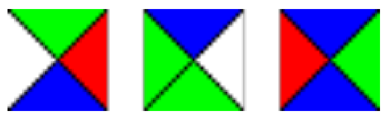
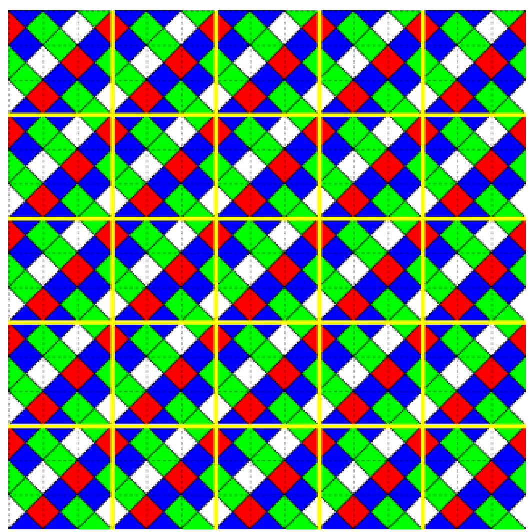


Figure 2: A set of Wang tiles.



Periodic

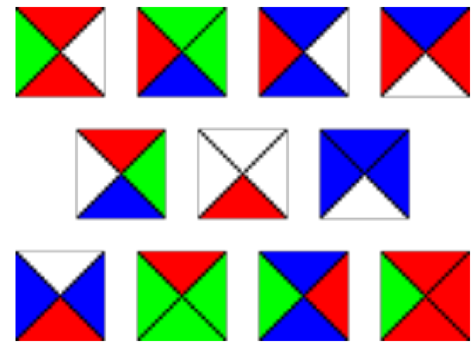
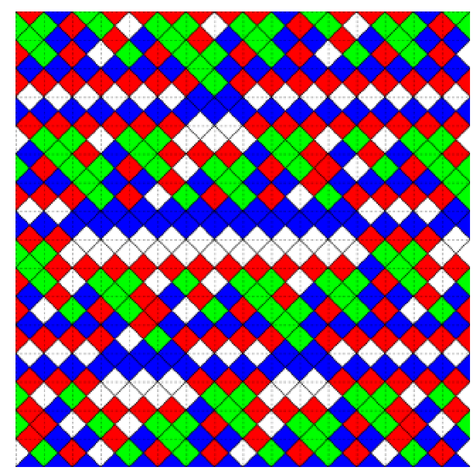


Figure 4: The smallest aperiodic set of Wang tiles.



Aperiodic

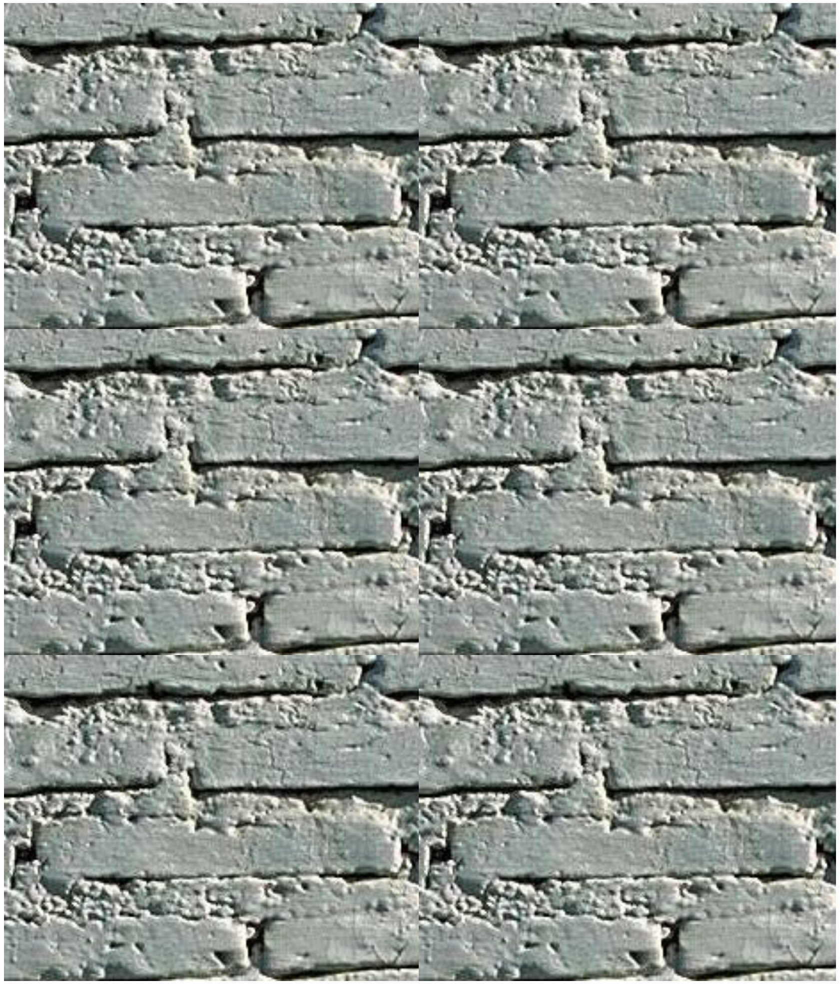
Synthesis by Tiling

- Repeat pattern

Synthesis by Tiling

- Repeat pattern

Synthesis by Tiling



- Repeat pattern
- How can we improve?

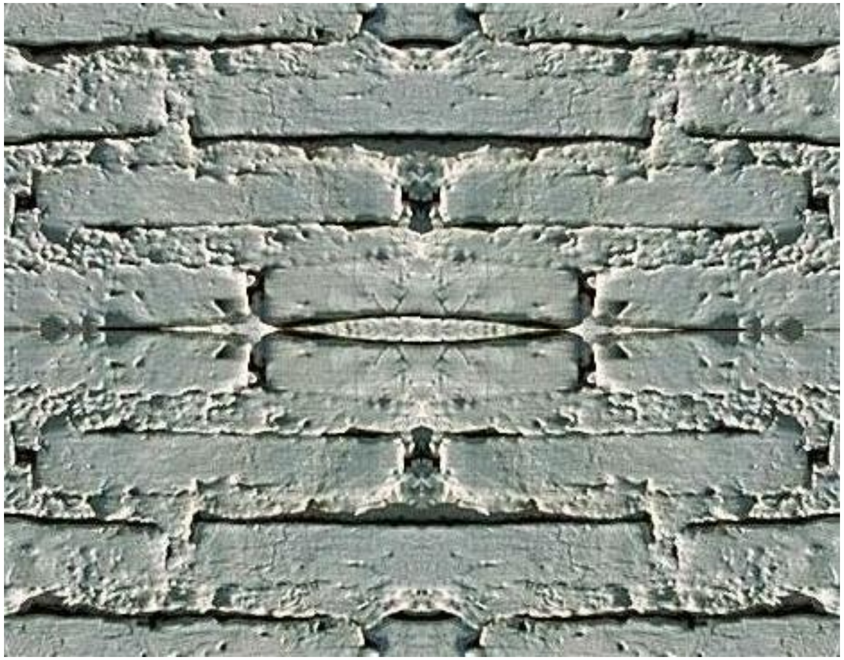
Synthesis by Tiling

- Repeat pattern
  - reduce seems by mirroring

Synthesis by Tiling

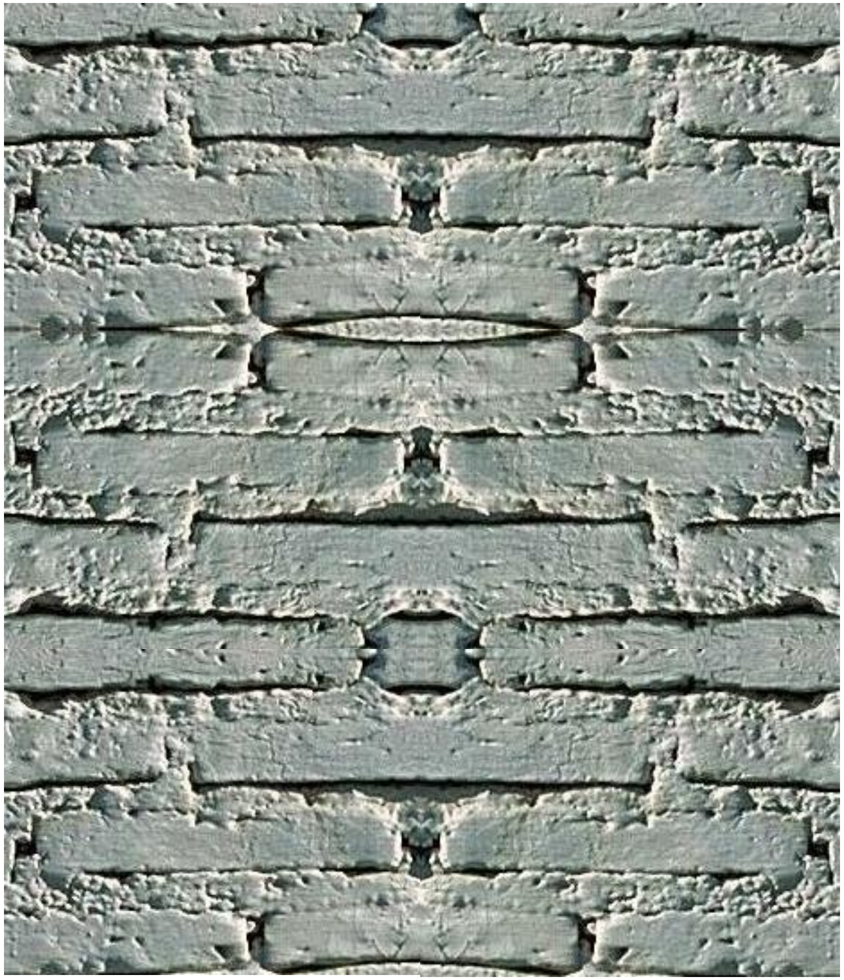
- Repeat pattern
  - reduce seems by mirroring

Synthesis by Tiling



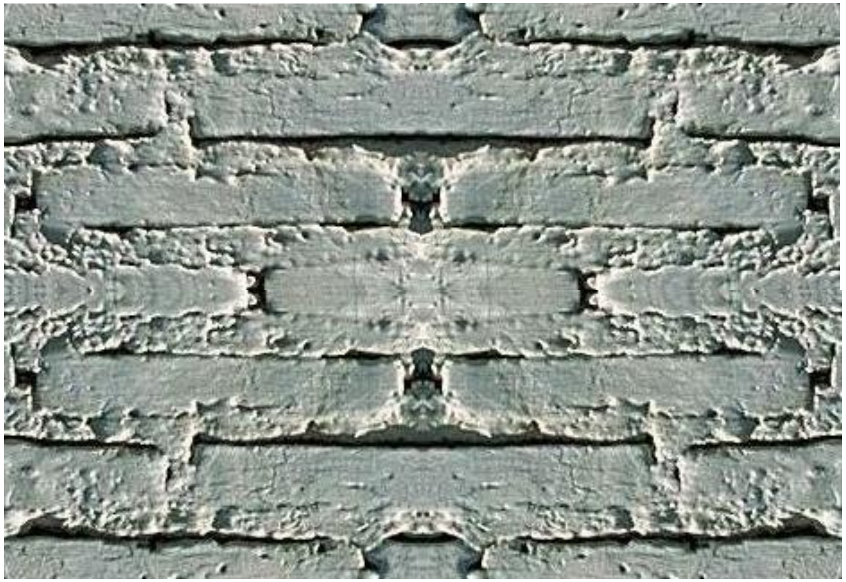
- Repeat pattern
  - reduce seems by mirroring

Synthesis by Tiling



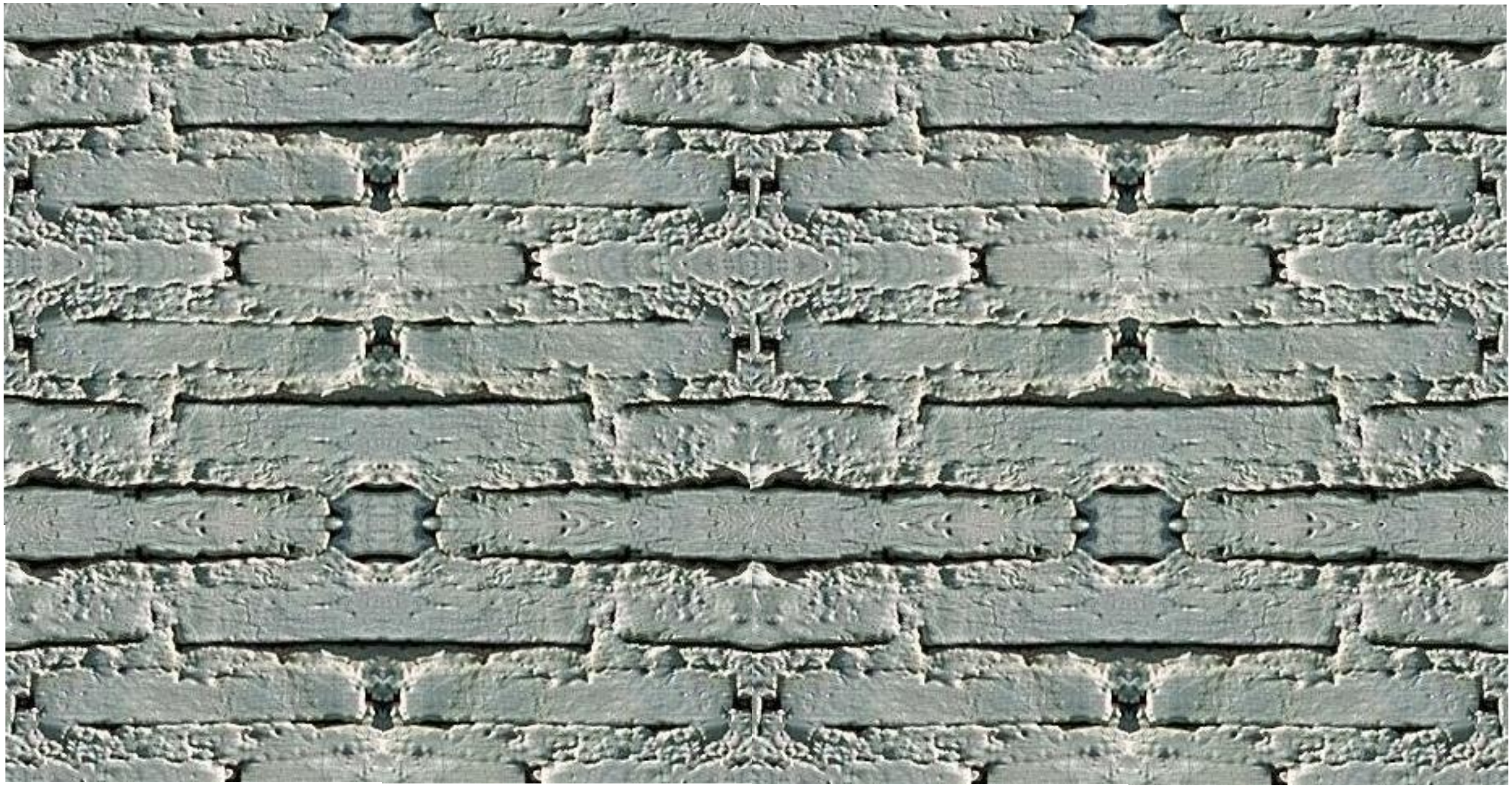
- Repeat pattern
  - reduce seems by mirroring
  - How we can further improve?

Synthesis by Tiling



- Repeat pattern
  - reduce seems by mirroring
  - reduce seems by choosing tile that covers one period of repeated texture

Synthesis by Tiling

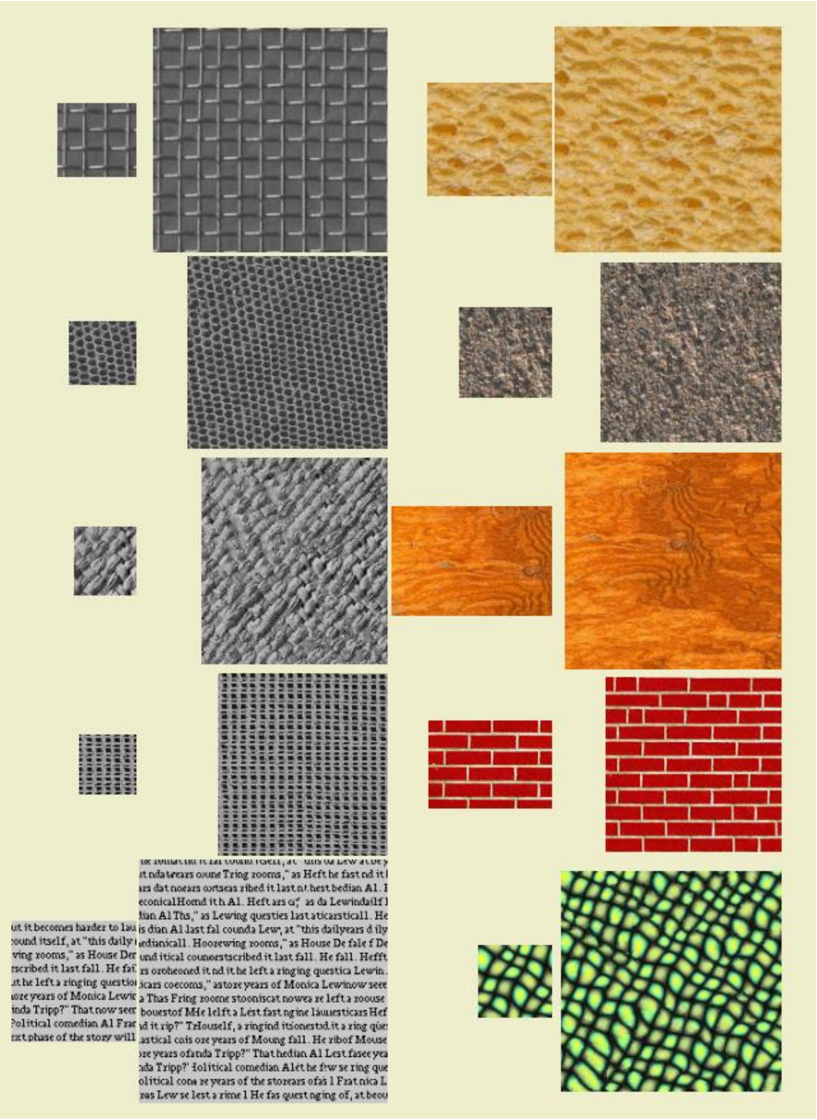


Bricks are similar but not identical

Solution?

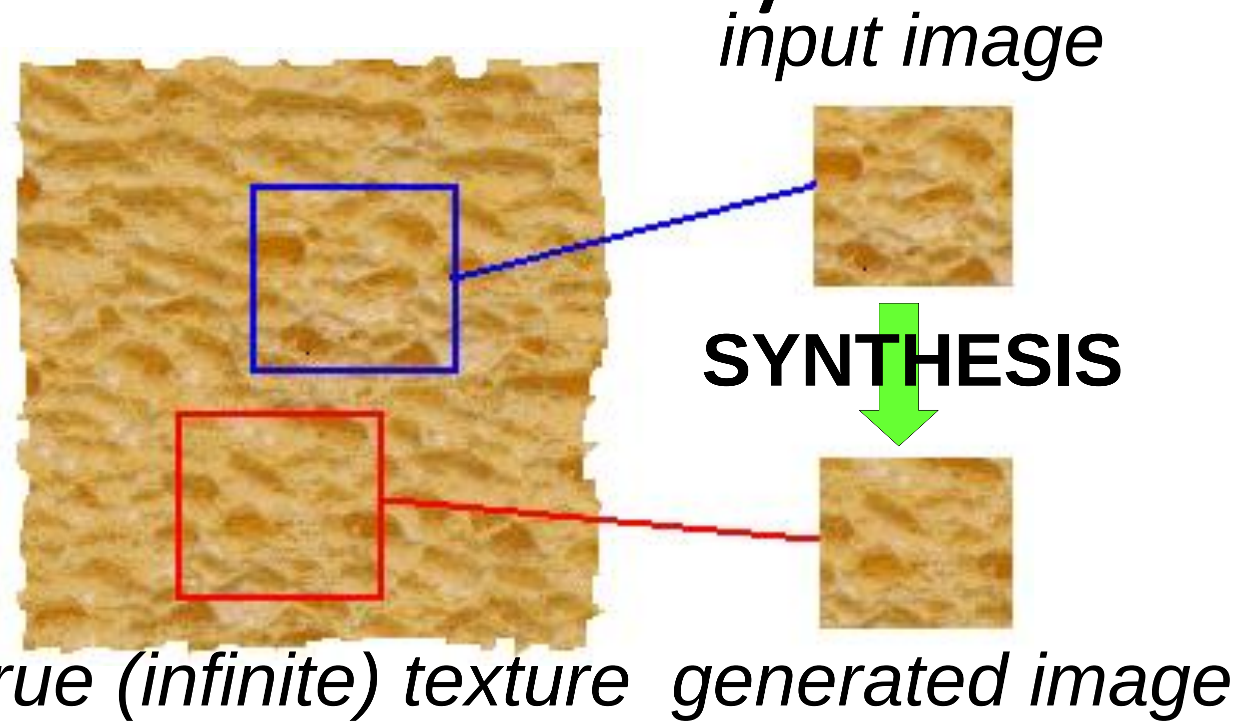
Texture Synthesis

- “Texture Synthesis by Non-parametric Sampling”, Efros and Leung, 1999



(slides from their presentation)

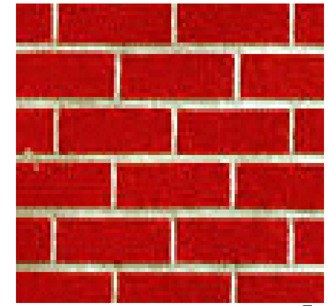
Goal of Texture Synthesis



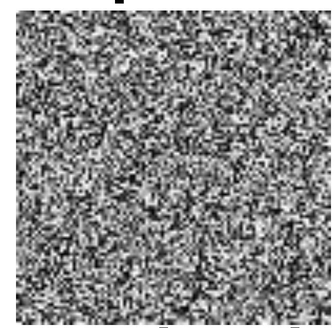
- Given a finite sample of some texture, the goal is to synthesize other samples from that same texture.
  - The sample needs to be "large enough"

The Challenge

- Texture analysis: how to capture the essence of texture?
- Need to model the whole spectrum: from repeated to stochastic texture
- This problem is at intersection of vision, graphics, statistics, and image compression



repeated



stochastic

Both?

Our Approach

- Goals:
  - preserve local structure
  - model wide range of real textures
  - ability to do constrained synthesis
- Method:
  - Texture is “grown” one pixel at a time
  - conditional pdf of pixel given its neighbors synthesized thus far is computed directly from the sample image

Motivation from Language

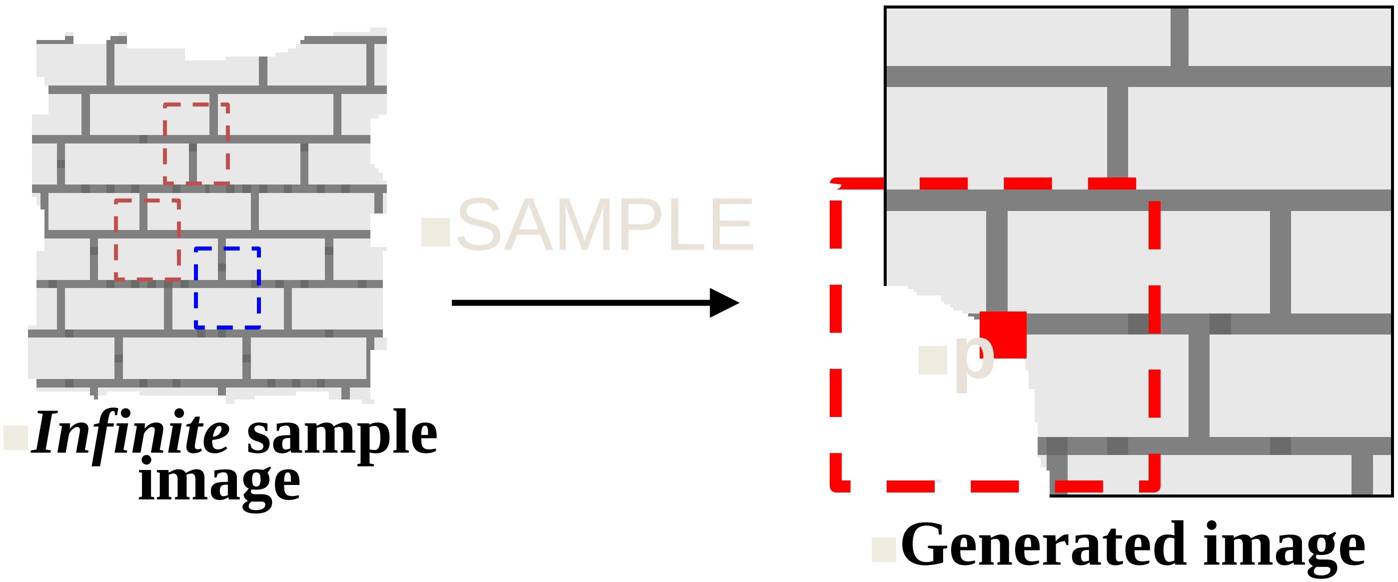
- [Shannon,'48] proposed a way to generate English-looking text using N-grams:
  - Assume a generalized Markov model (i.e., the next state is only dependent on the current state and is independent of anything in the past)
  - Use a large text to compute probability distributions of each letter given N-1 previous letters
    - precompute or sample randomly
  - Starting from a seed repeatedly sample this Markov chain to generate new letters
  - One can use whole words instead of letters too:

WE NEED TO EAT CAKE

Mark V. Shaney (Bell Labs)

- Results (using alt.singles corpus):
  - *"As I've commented before, really relating to someone involves standing next to impossible."*
  - *"One morning I shot an elephant in my arms and kissed him."*
  - *"I spent an interesting evening recently with a grain of salt"*
- Notice how well local structure is preserved!
- Jump 20 years: LLMs not so surprising…
- Now let's try this in 2D...

Synthesizing One Pixel

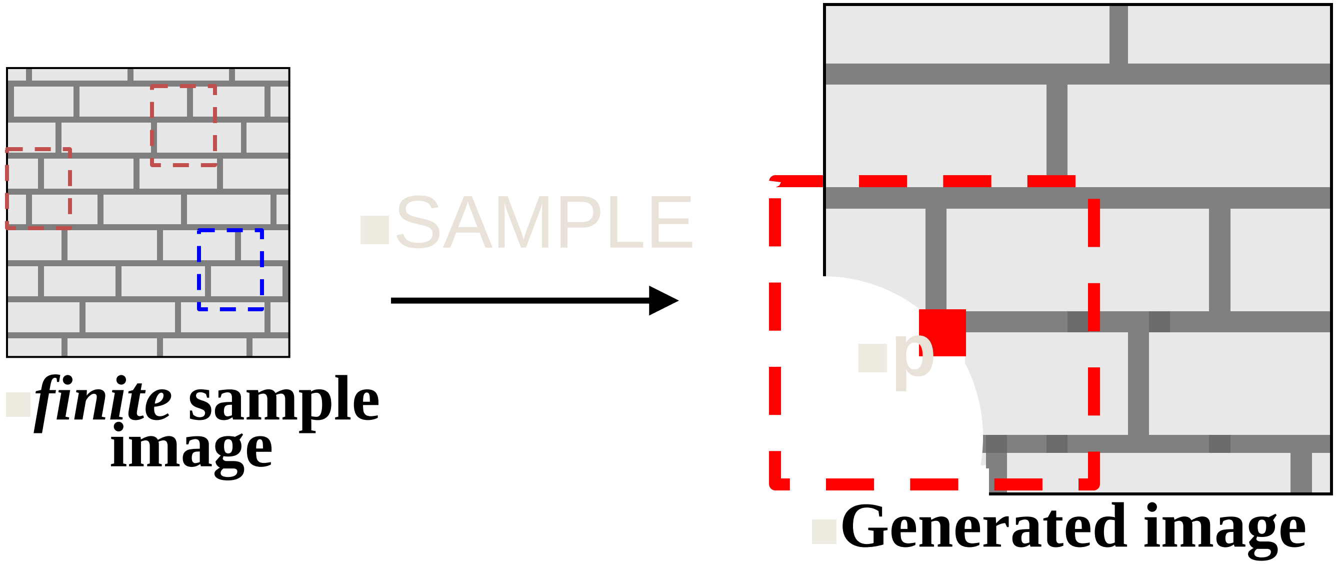


– Assuming Markov property, what is conditional probability distribution of p, given the neighbourhood window?

– Instead of constructing a model, let's directly search the input image for all such neighbourhoods to produce a histogram for p

– To synthesize p, just pick one match at random

Really Synthesizing One Pixel



- However, since our sample image is finite, an exact neighbourhood match might not be present
- So we find the **best** match using SSD error (weighted by a Gaussian to emphasize local structure), and take all samples within some distance from that match

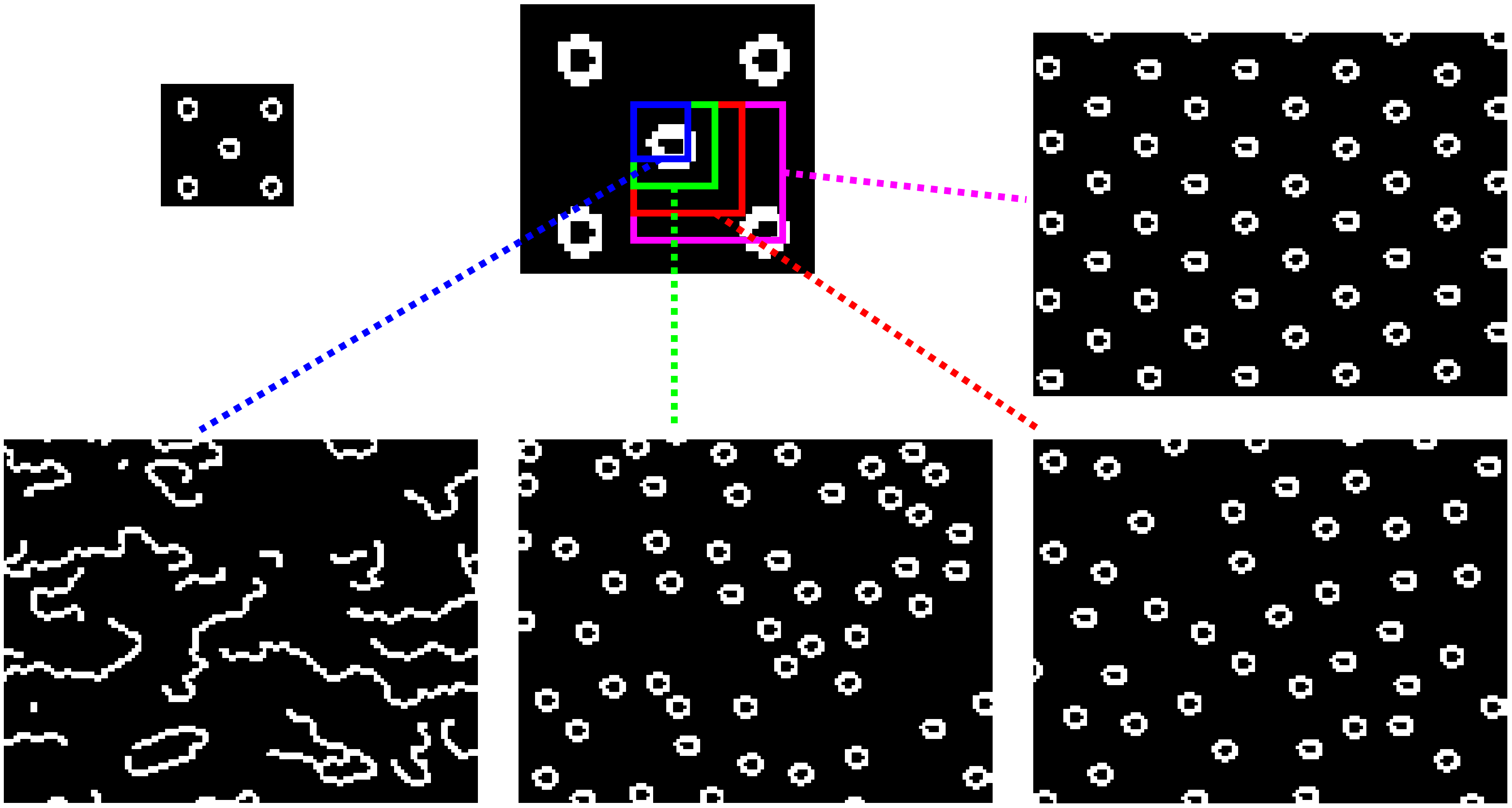
Growing Texture

- Starting from the initial configuration, we "grow" the texture one pixel at a time
- The size of the neighbourhood window is a parameter that specifies how stochastic the user believes this texture to be
- To grow from scratch, we use a random 3x3 patch from input image as seed

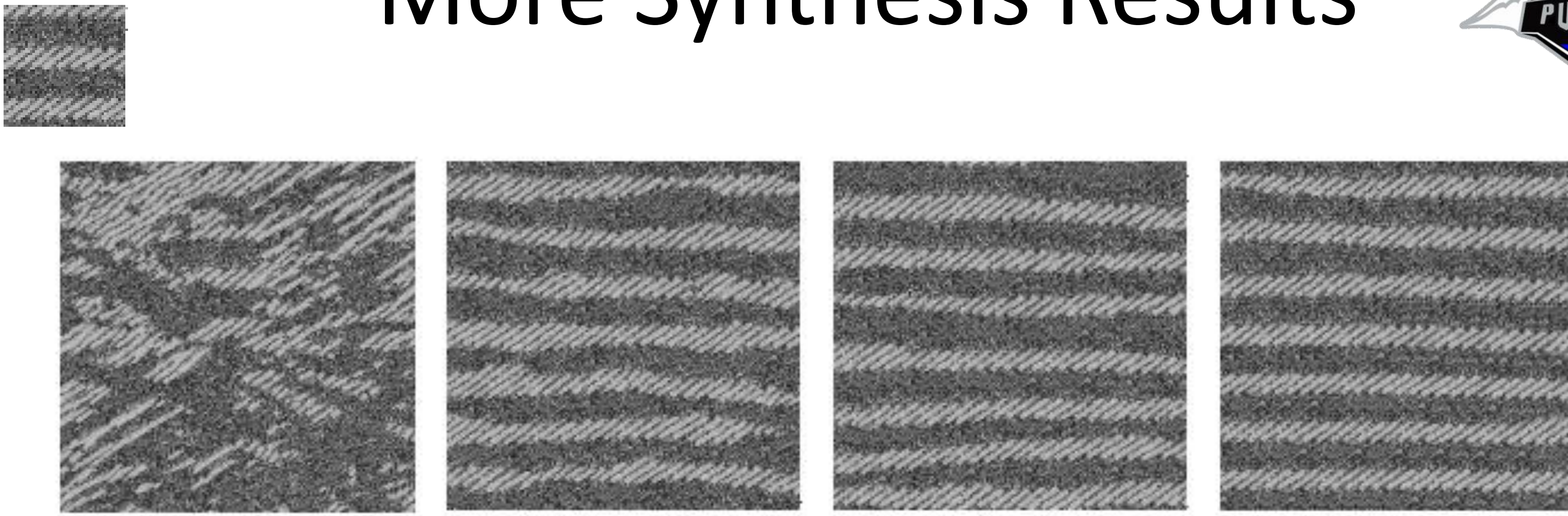
Some Details

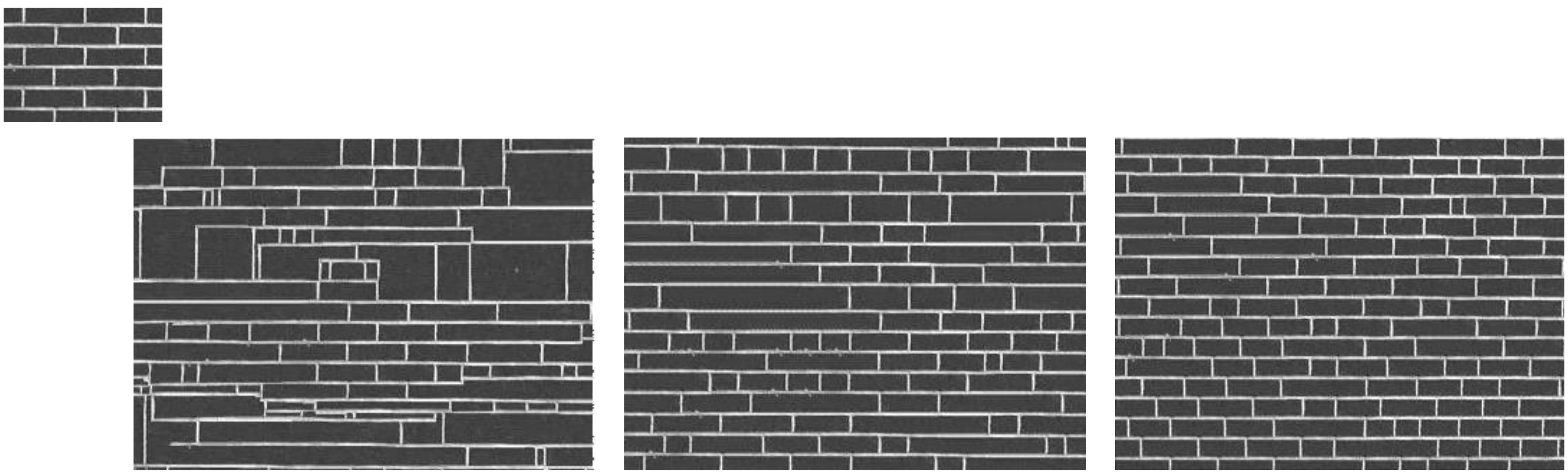
- Growing is in “onion skin” order
  - Within each “layer”, pixels with most neighbors are synthesized first
  - If no close match can be found, the pixel is not synthesized until the end
- Using *Gaussian-weighted* SSD is very important
  - to make sure the new pixel agrees with its closest neighbors
  - Approximates reduction to a smaller neighborhood window if data is too sparse

Randomness Parameter



More Synthesis Results





Increasing window size

More Results

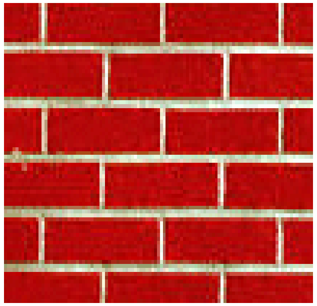
- wood

- granite

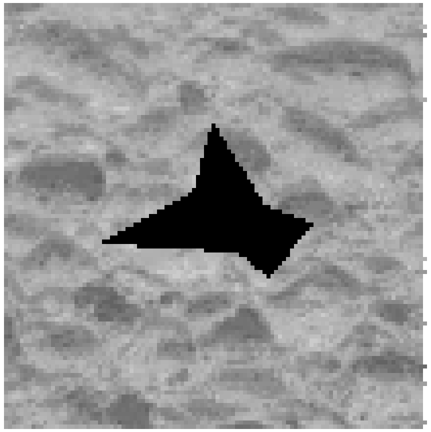
More Results

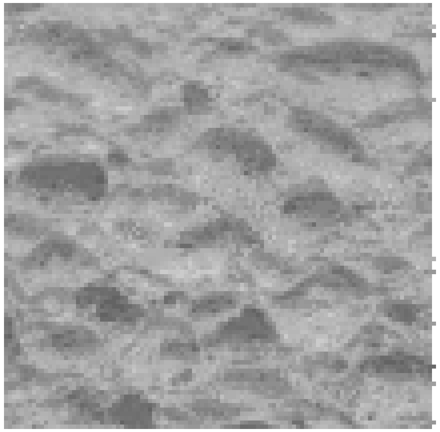
- white bread

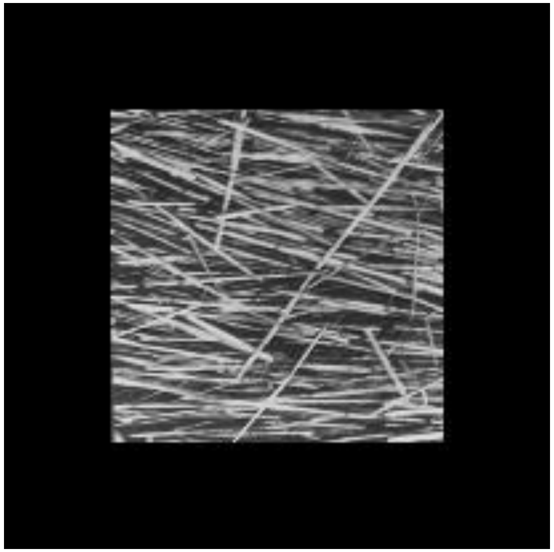
- brick wall

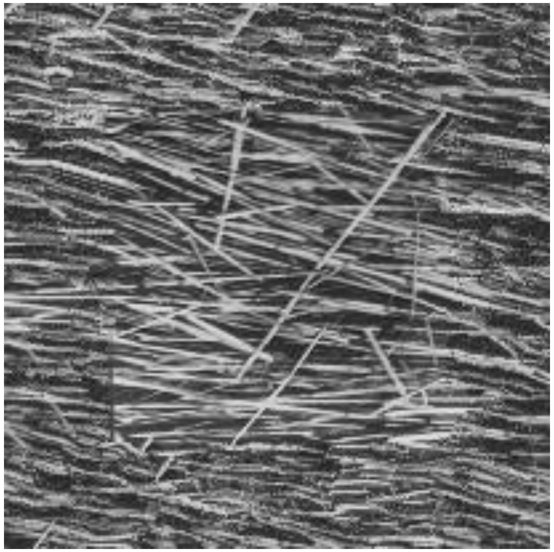


Constrained Synthesis



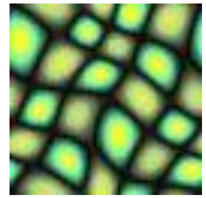


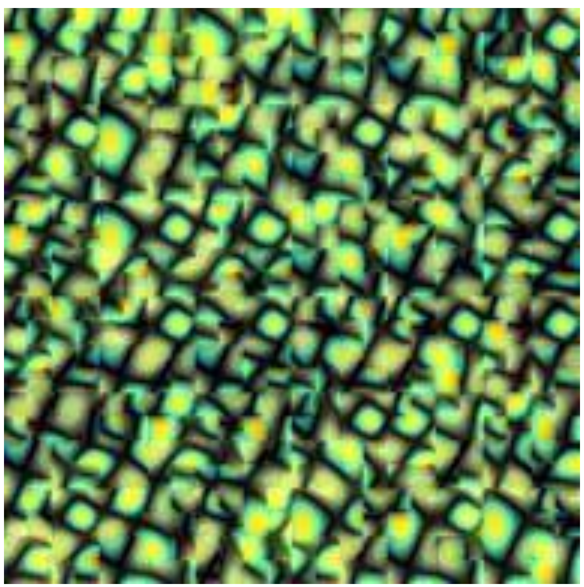




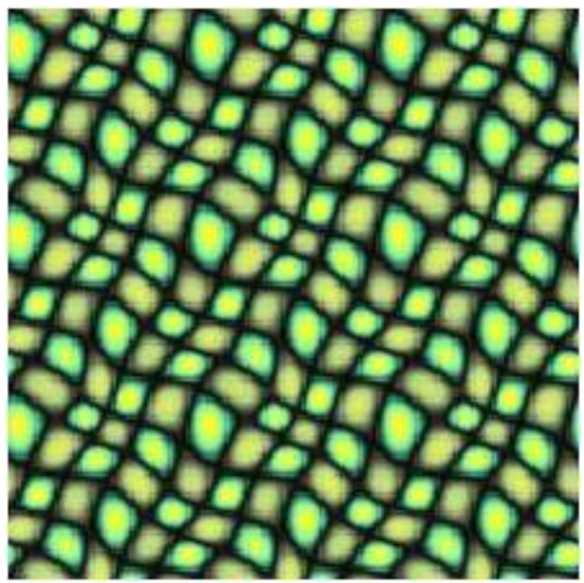
Visual Comparison

- *Synthetic tilable*
- *texture*

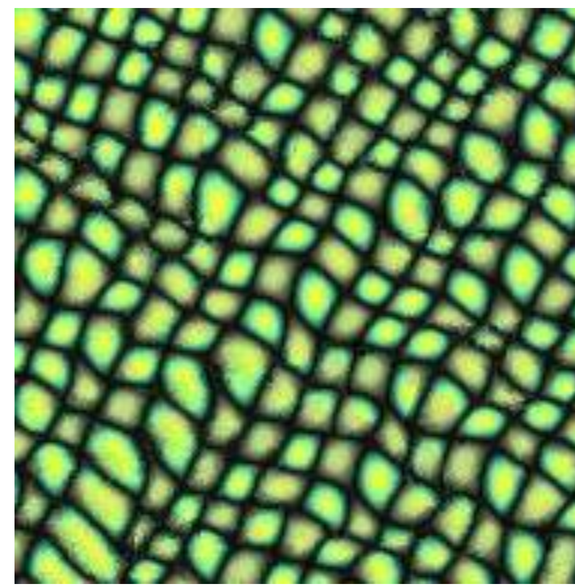




- **[DeBonet, ‘97]**

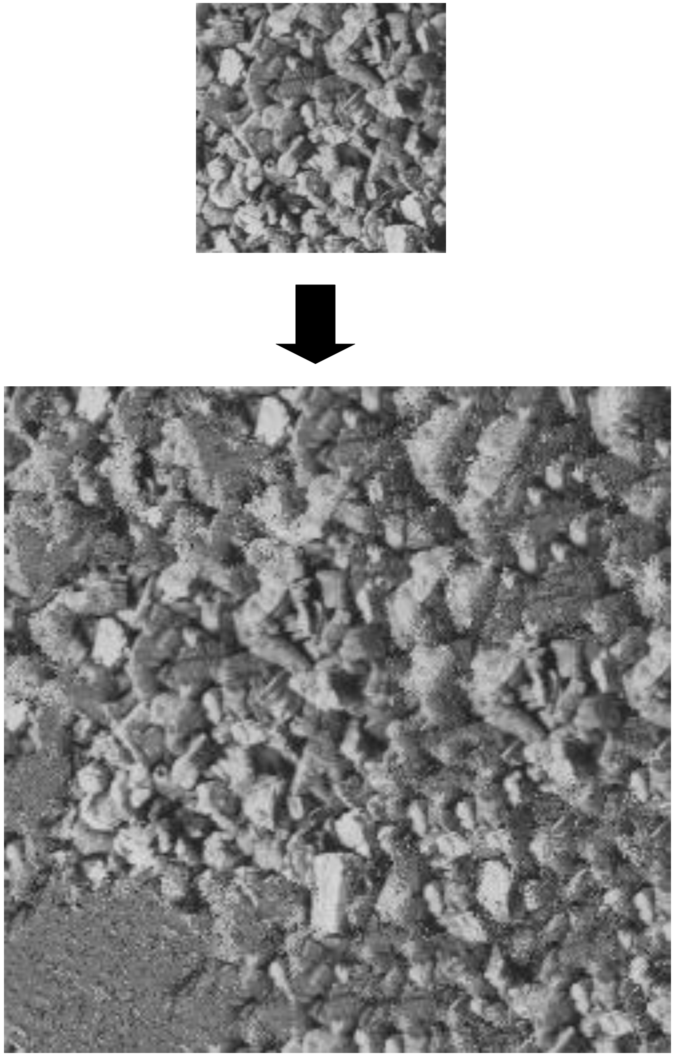


- **Simple tiling**

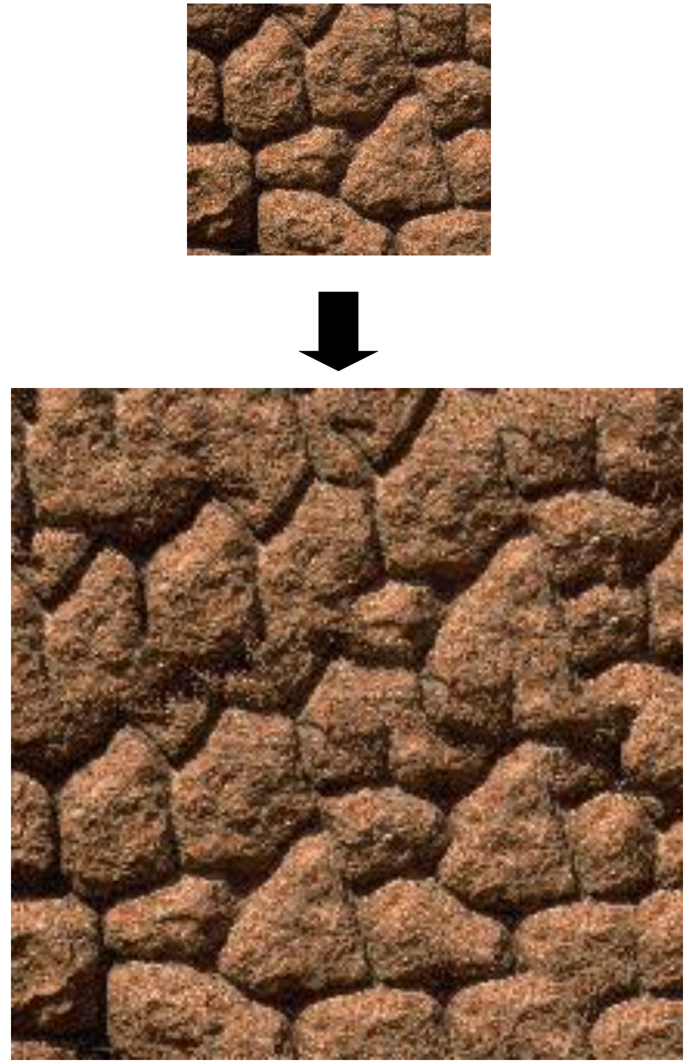


- **Our approach**

Failure Cases

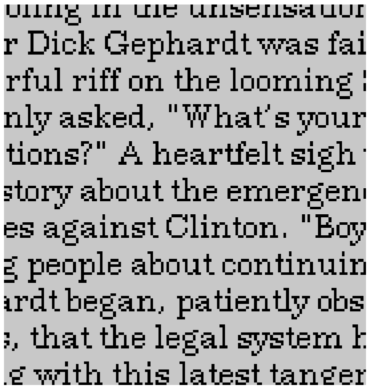


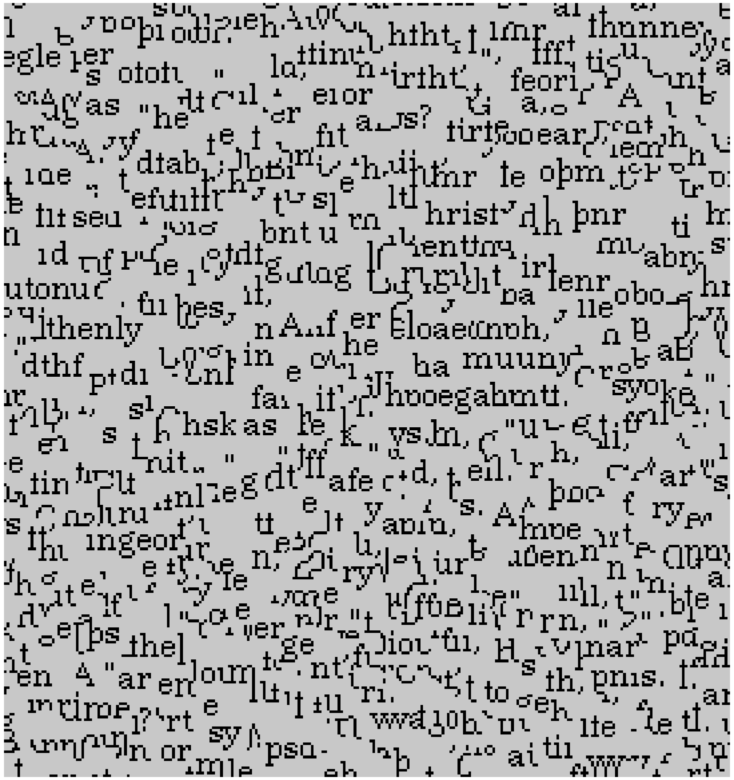
- Growing garbage

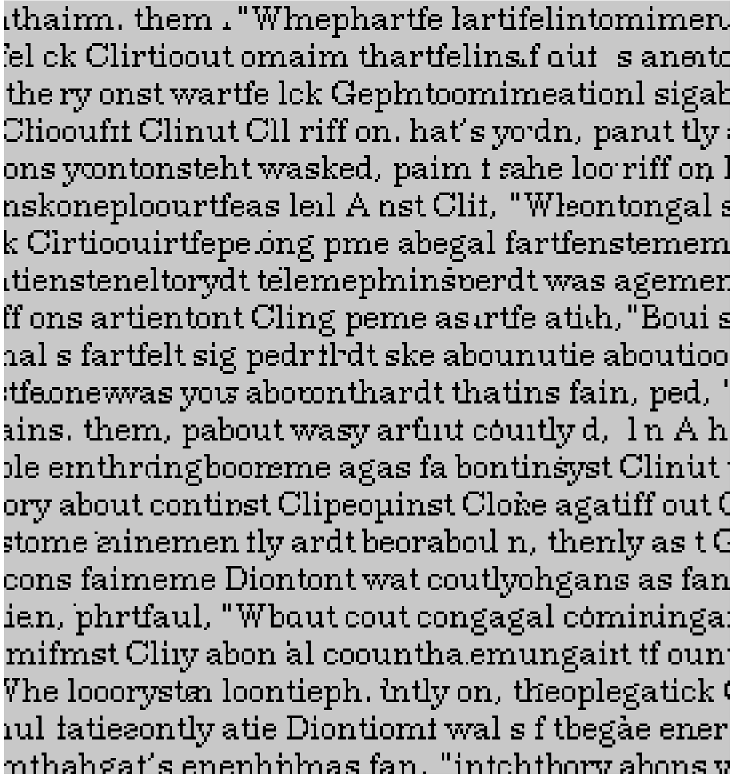


- Verbatim copying

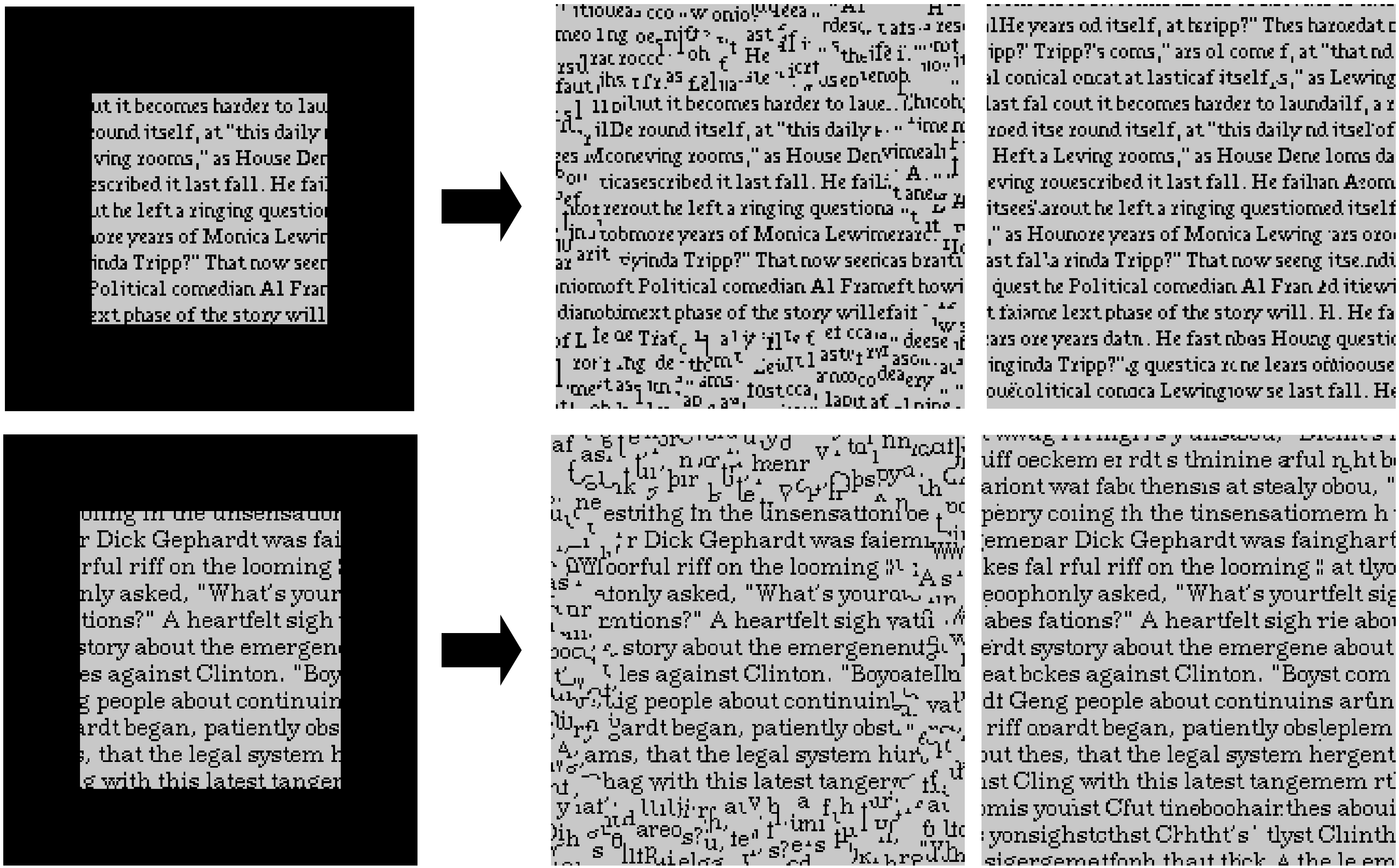
Homage to Shannon







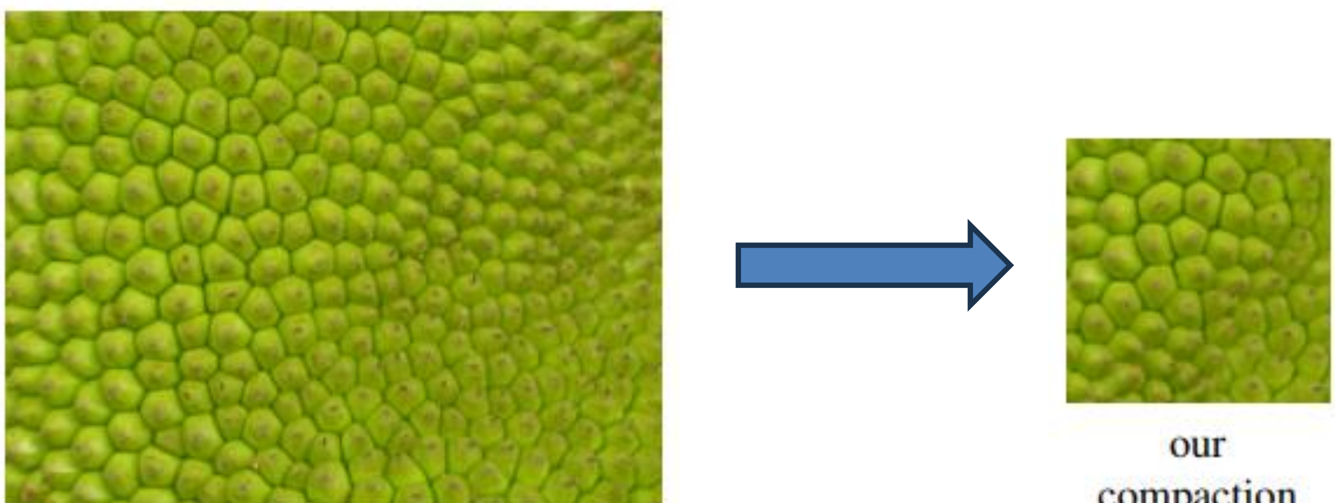
Constrained Text Synthesis



Texture Synthesis $^{-1}$

Inverse Texture Synthesis

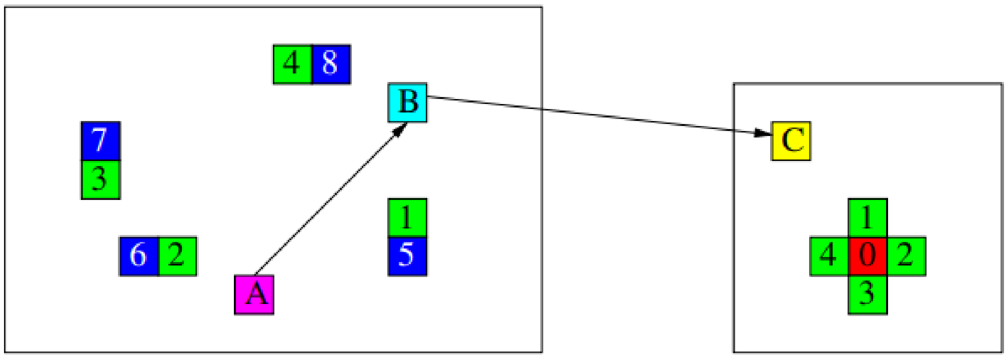
- "Inverse Texture Synthesis", Wei, Han, Zhou, Bao, Guo, Shum, 2008



original

our compaction

Inverse Texture Synthesis



X input $\qquad$ Z compaction

Figure 7: Illustrating of our improved solver. Here, we use a toy case with only four k-coherence neighbors as exemplified in pixels $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}$ around $\mathbf{0}$ in the compaction. The sources of these four pixels are marked with the same numbers in the input. $\mathbf{z}$ E-step: the value of $\mathbf{0}$ is chosen from $\{\mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}\}$ as determined by $\mathbf{0}$'s neighbors $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}$. Forward M-step: the best match for $\mathbf{0}$ is also chosen from $\{\mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}\}$. Inverse M-step: **B** is a cluster center where **A** belongs to. So **B** first finds the best match **C** through exhaustive search, and the best match for **A** is determined through **B**.

Inverse Texture Synthesis

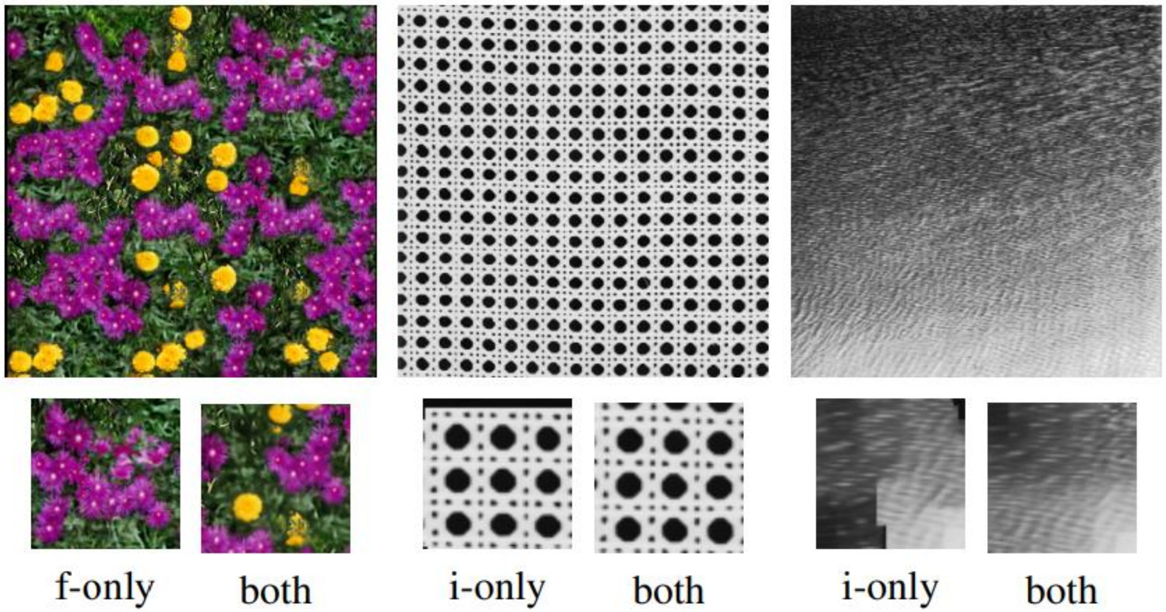


Figure 6: Why we need both forward and inverse terms in our energy function. With only the forward term the compaction will not provide sufficient coverage of the original (left case). With only the inverse term the compaction may contain garbage (middle case) or discontinuity (right case). All compactions are drawn in larger scale than the originals for clarity.

- https://www.youtube.com/watch?v=cJJkQoFjQPU

Making Stuff

- Texture Synthesis
  - Make/replicate small fragments

- **Image Synthesis, Generative 2D Images, Procedural Modeling**
  - **Make 2D images**

- Model Synthesis, Generative 3D Modeling, Procedural Modeling
  - Make 3D models

Procedural Modeling

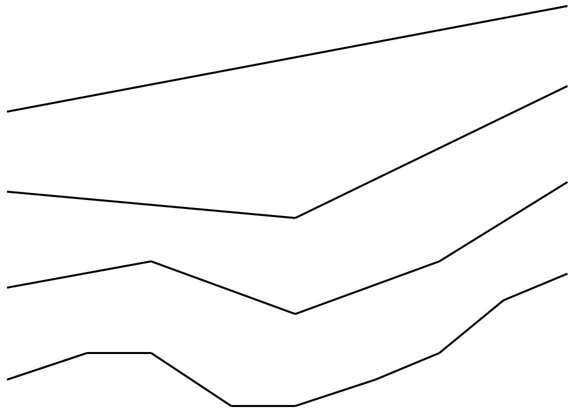
- Apply algorithms for producing objects and scenes
- The rules may either be embedded into the algorithm, configurable by parameters, or externally provided
- Key notions:
  - Detail amplification: from a little, make a lot, but not random
  - Kolmogorov Complexity: “a measure of the complexity or randomness of an object”

Procedural Modeling

- 1-2.5D:
  - Fractals
  - Terrains
  - Image-synthesis
    - Perlin Noise
    - Clouds
- 3D:
  - Plants
  - Cities
  - And procedures in general…

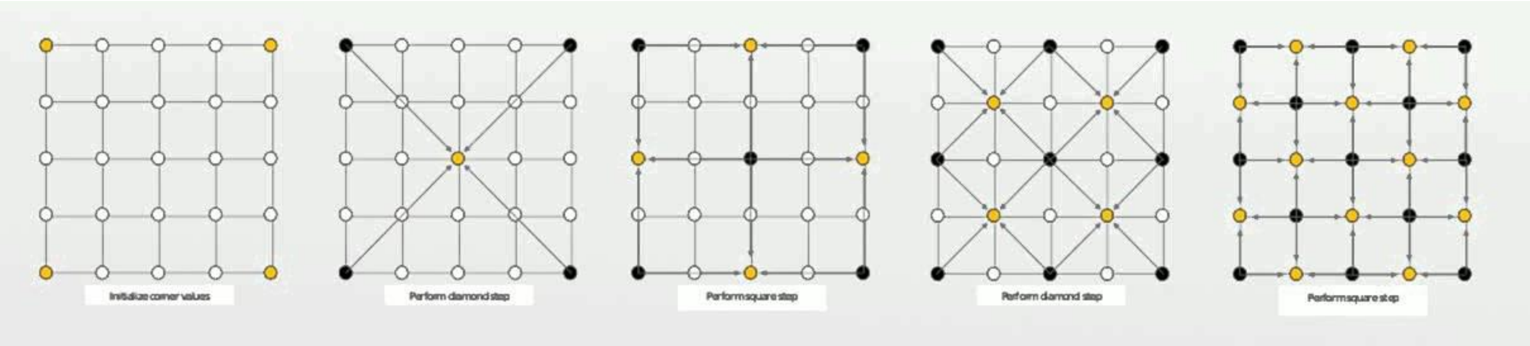
Fractals

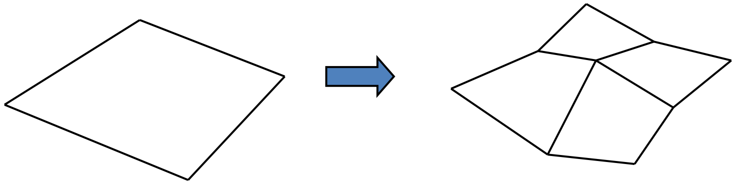
- Consider a simple line fractal
  - Split a line segment, randomize the height of the midpoint by some number in the [-*r*,*r*] range
  - Repeat and randomize by [-*r*/2,*r*/2]
  - Continue until a desired number of steps, randomizing by half as much each step



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

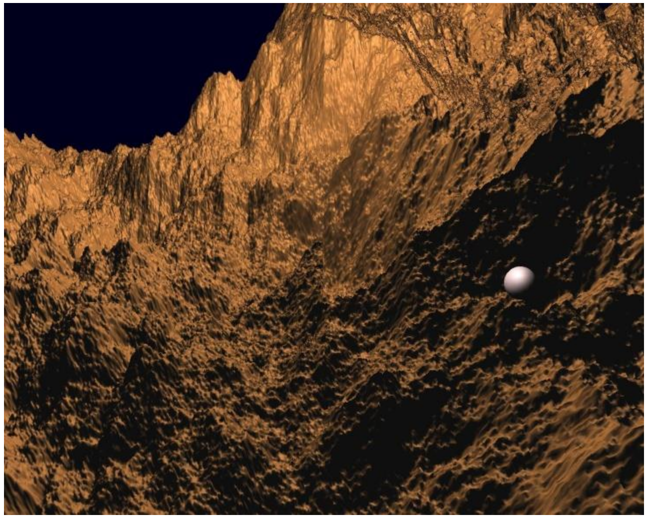
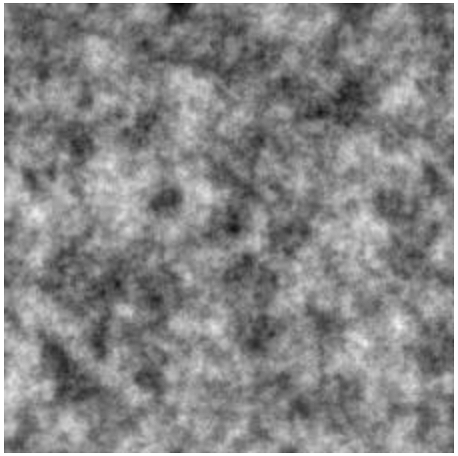


Image Synthesis

- Procedurally generate an image (pixels)

Idea: Perlin Noise

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



Plant Modeling

- The Algorithmic Beauty of Plants

Background: Chomsky Hierarchy

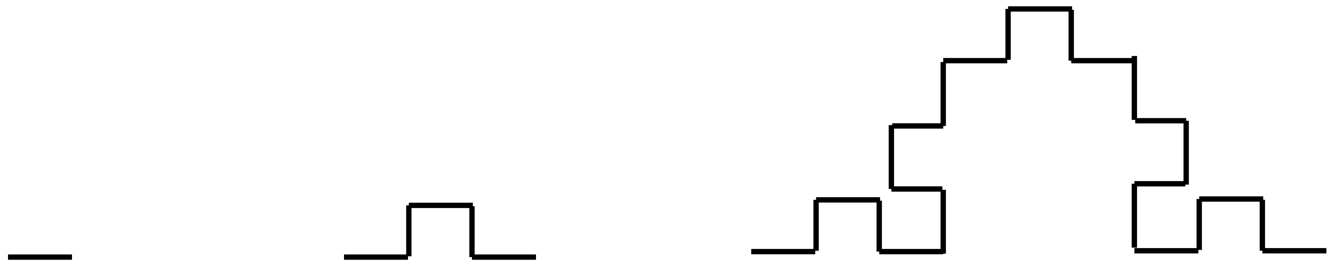
- Type 0 grammars
  - Unrestricted, recognized by Turing machine
- Type 1 grammars
  - Context-sensitive grammars
- Type 2 grammars
  - Context-free grammars
- Type 3 grammars
  - Regular grammars (e.g., regular expressions)

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



(Context-Free) L-system for Plants

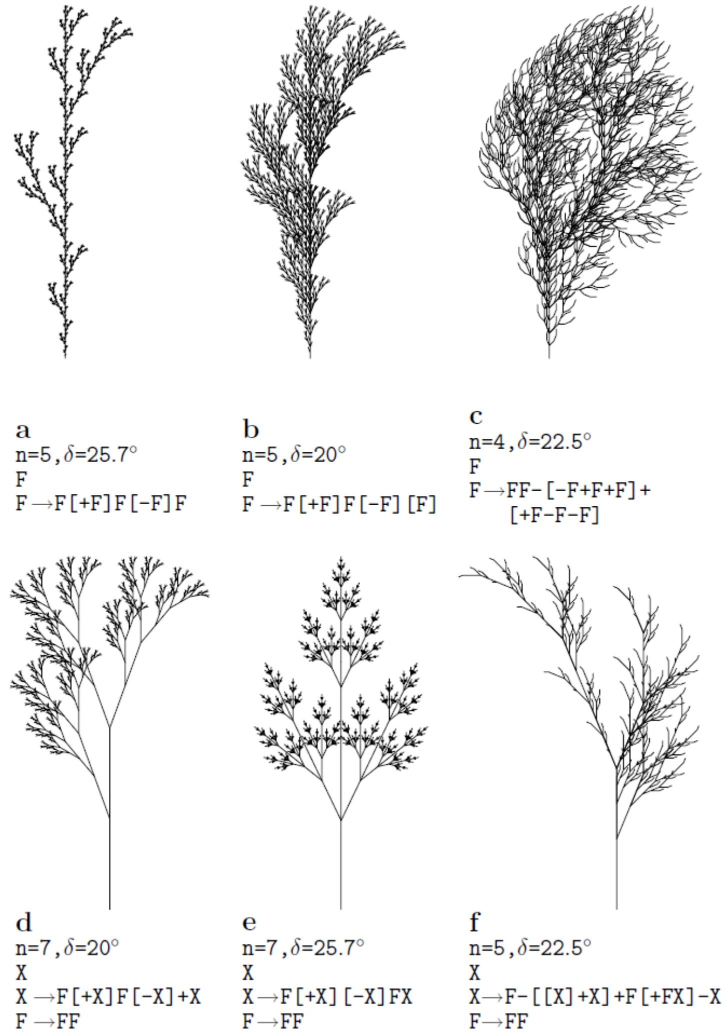


Figure 1.24: Examples of plant-like structures generated by bracketed OL-systems. L-systems (a), (b) and (c) are edge-rewriting, while (d), (e) and (f) are node-rewriting.

L-system for Plants (stochastic)

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

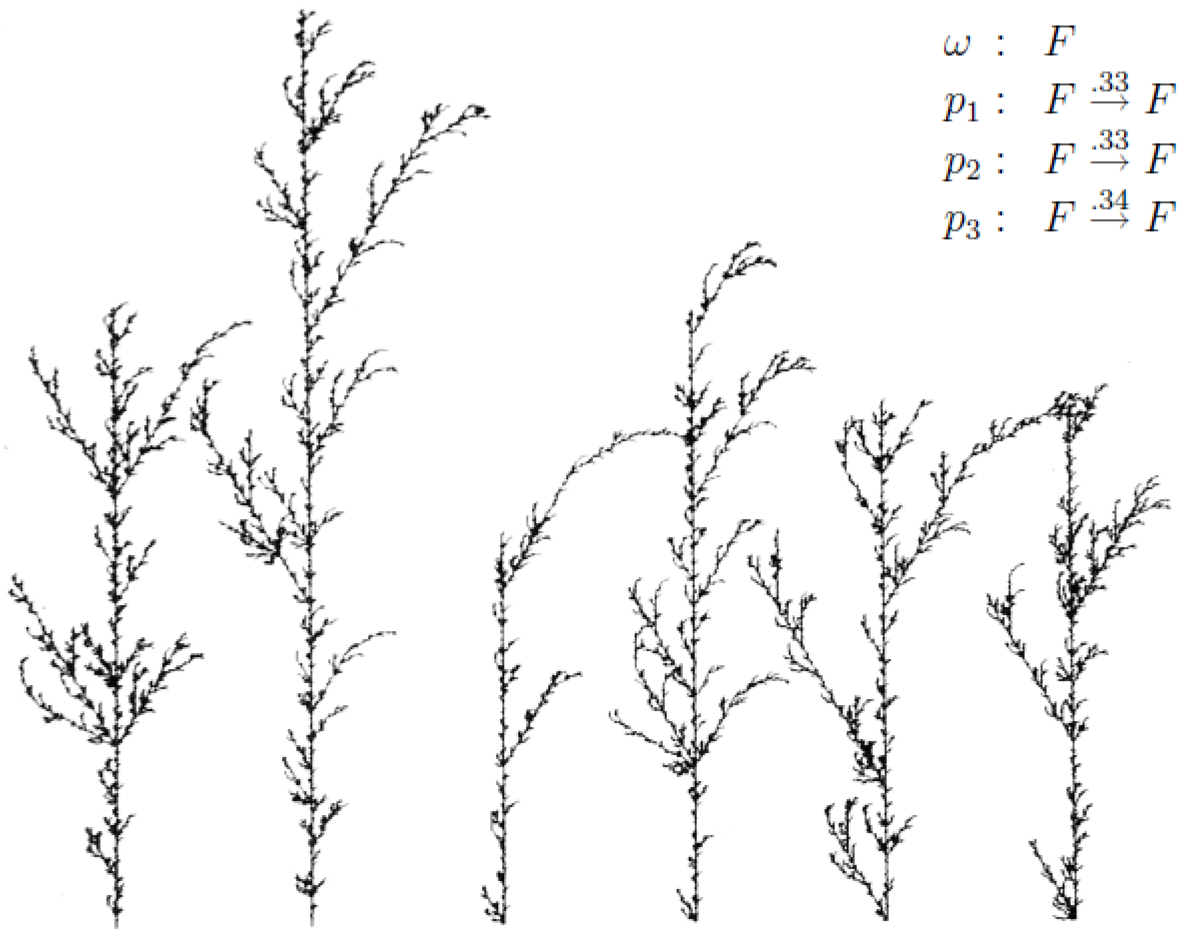


Figure 1.27: Stochastic branching structures

L-system for Plants (3D)

n=5, δ=18°

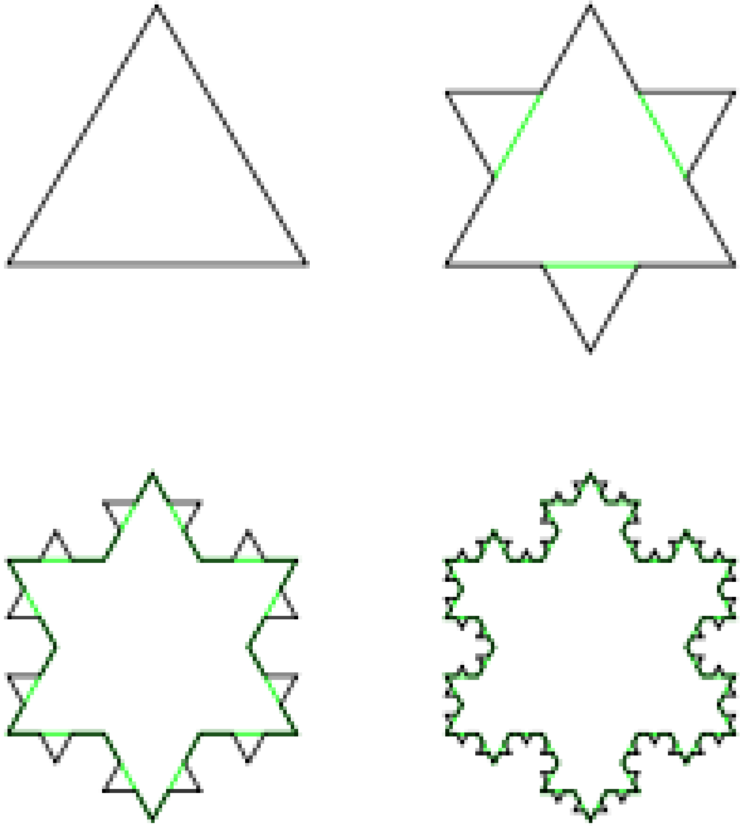
ω : plant
p_1 : plant → internode + [plant + flower] − − //
[− − leaf] internode [+ + leaf] −
[plant flower] + + plant flower
p_2 : internode → F seg [// & & leaf] [// ∧ ∧ leaf] F seg
p_3 : seg → seg F seg
p_4 : leaf → [' { +f−ff−f+ | +f−ff−f }]
p_5 : flower → [& & & pedicel ‘ / wedge //// wedge ////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

Virtual Ecosystem

Koch Snowflake



Demo

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

Shape Grammar

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

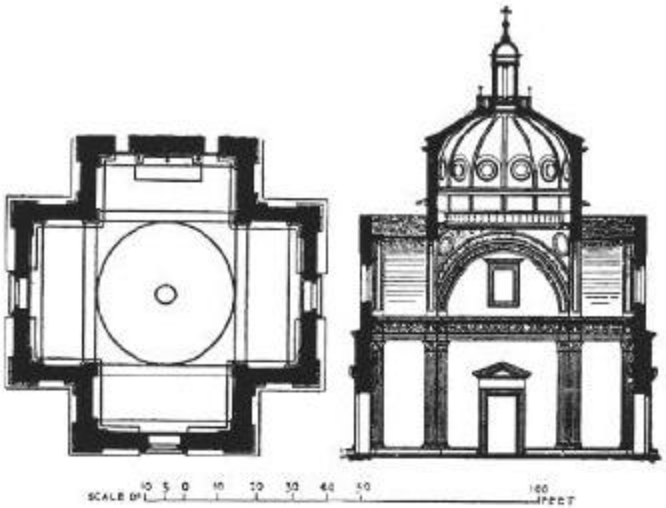
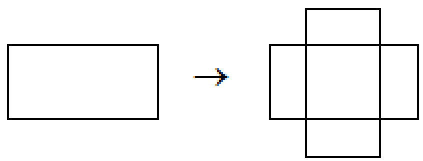


Illustration by Peter Murray, "the Artchitecture of the Italian Renaissance", Shocken Books Inc. 1963, Pp.96.

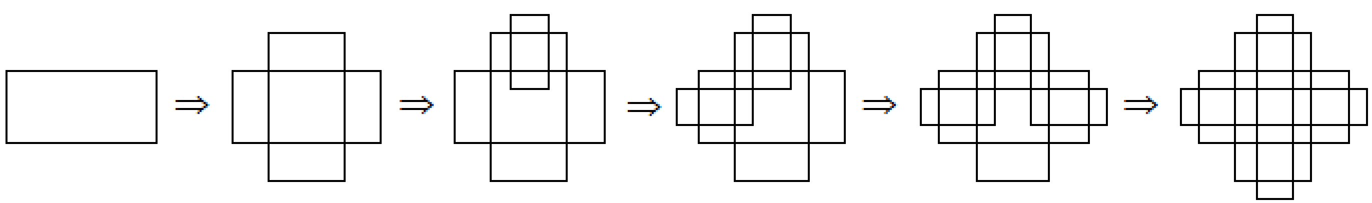
Shape Grammar

Shape Grammar

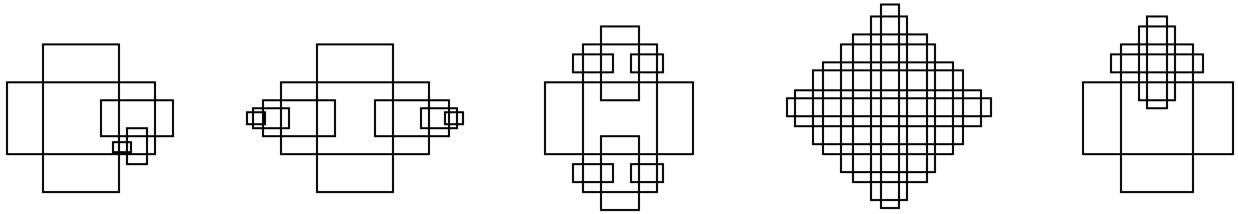


rule

DERIVATION



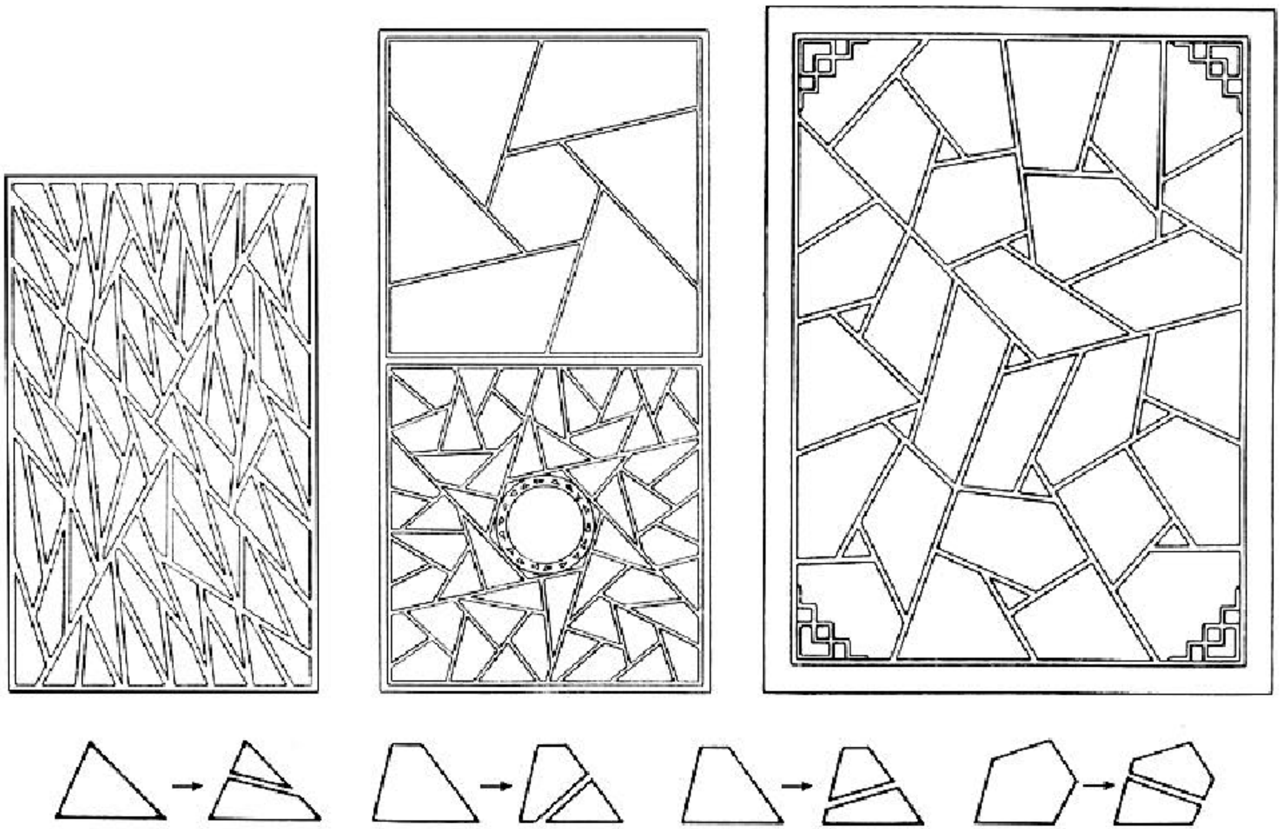
Shape Grammar



OTHER DESIGNS IN THE LANGUAGE

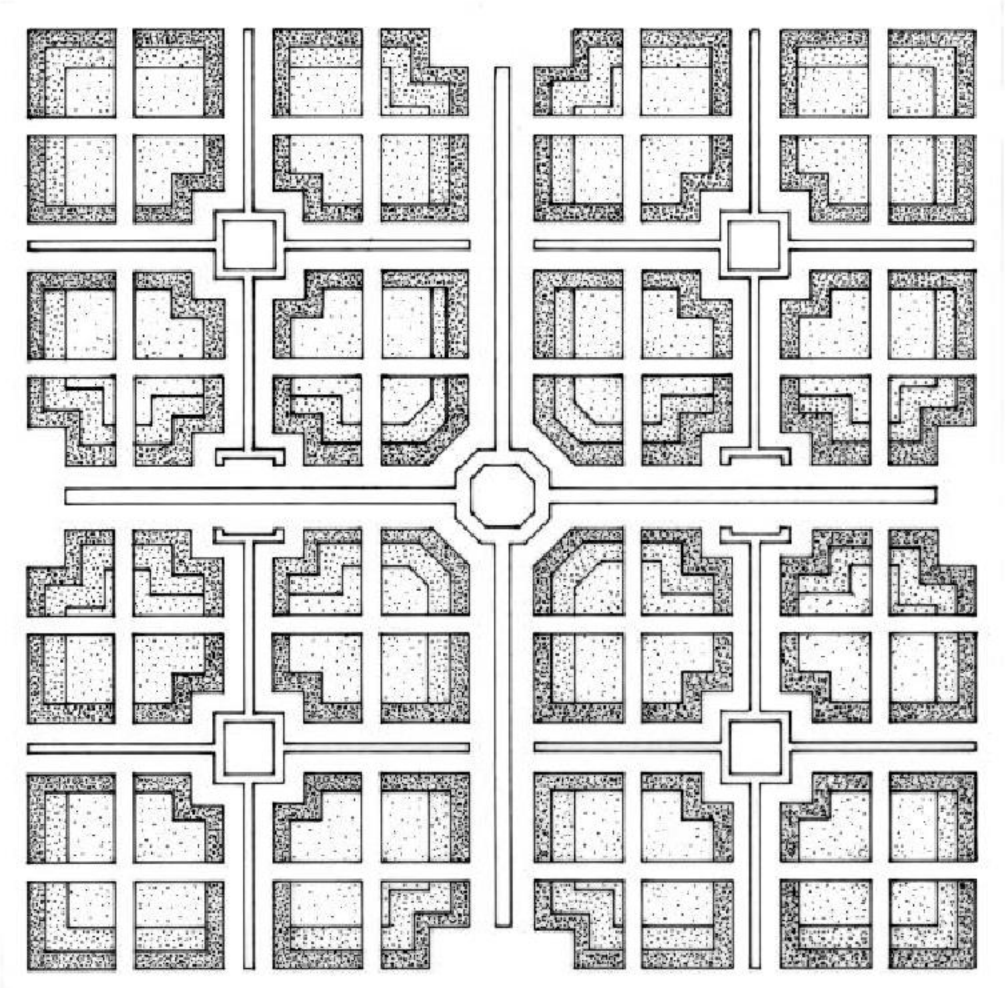
Exercise: let's make some art!

Shape Grammar



Ice-ray grammar

Shape Grammar



Mughul garden grammar

Shape Grammar

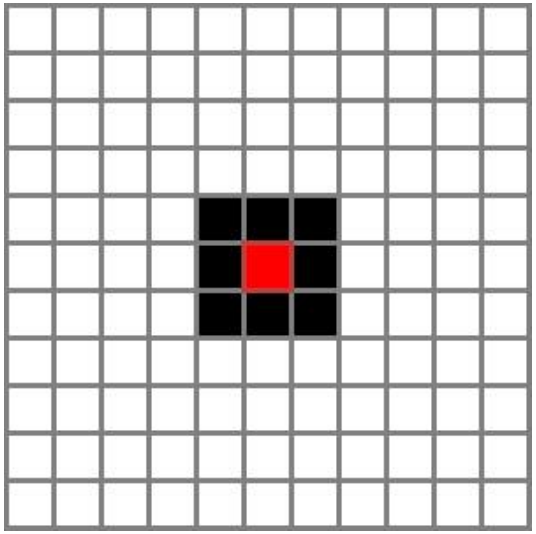
- Style: Mediterranean

Cellular Automata

- A 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

John Conway's Game of Life

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

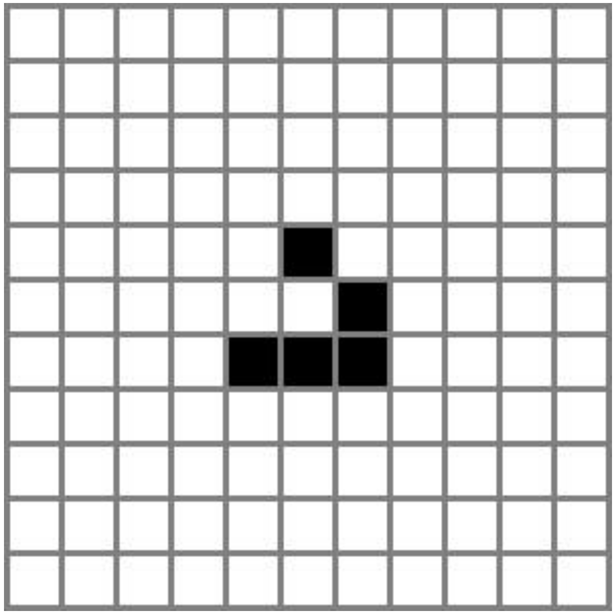


John Conway's Game of Life

- 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

- Cities
- Buildings
- CityEngine
  - CityEngine
  - http://proceedings.esri.com/library/userconf/devsummit12/papers/developing_with_esri_cityengine.pdf

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

Videos and more

- Image-based Modeling
  - http://www.ece.nus.edu.sg/stfpage/eletp/Projects/ImageBasedModeling/
  - Facades: http://www.youtube.com/watch?v=amD6_i3MVZM

- Inverse Procedural Modeling of Cities
  - https://www.youtube.com/watch?v=HntNsZbWIgg&feature=youtu.be
- Our Work:
  - CGVLAB Urban