
# Content Generation (1): Texture/Image Synthesis, Procedural Modeling, Neural Rendering, and Generative Modeling

CS535

Daniel G. Aliaga
Department of Computer Science
Purdue University

# Content Generation


- 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

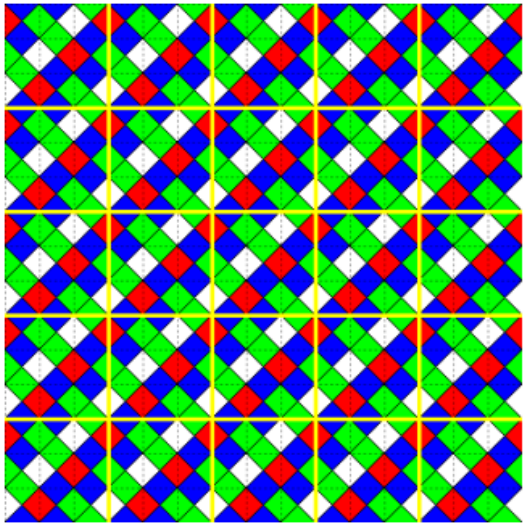
# Content Generation

- **Texture/Image Synthesis**
  - **Make/replicate small fragments**
- Procedural Modeling
- Neural Rendering
- Generative Modeling

# Synthesis by Tiling

- Simple tiling
- Wang tiles (~1960s)

Figure 2: A set of Wang tiles.



Periodic

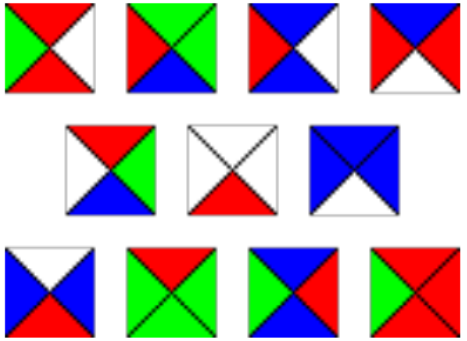
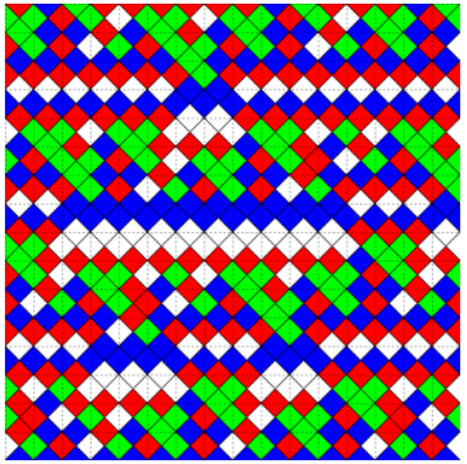


Figure 4: The smallest aperiodic set of Wang tiles.

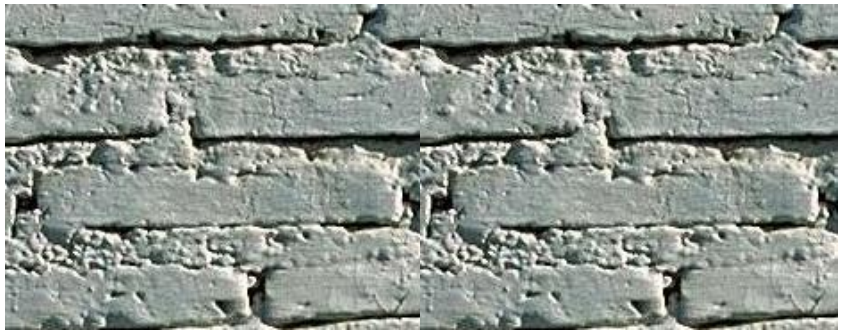


Aperiodic

# Synthesis by Tiling

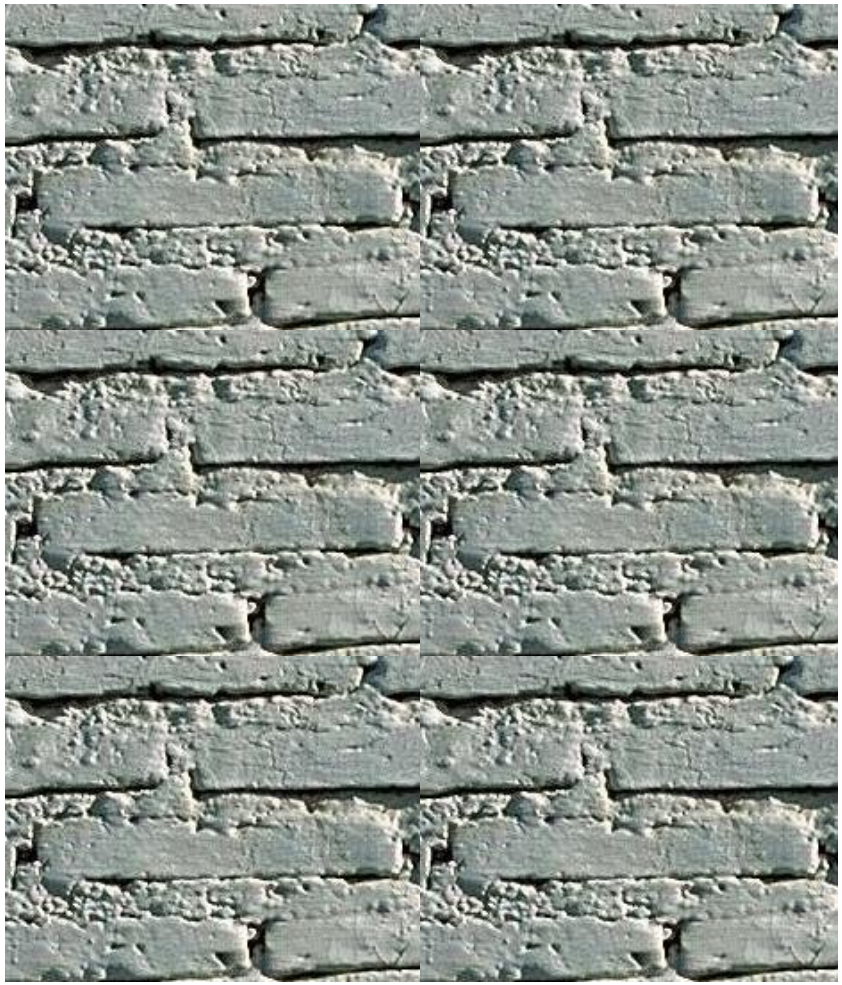
- Repeat pattern

# Synthesis by Tiling



- Repeat pattern

# Synthesis by Tiling



- Repeat pattern
- How can we improve?

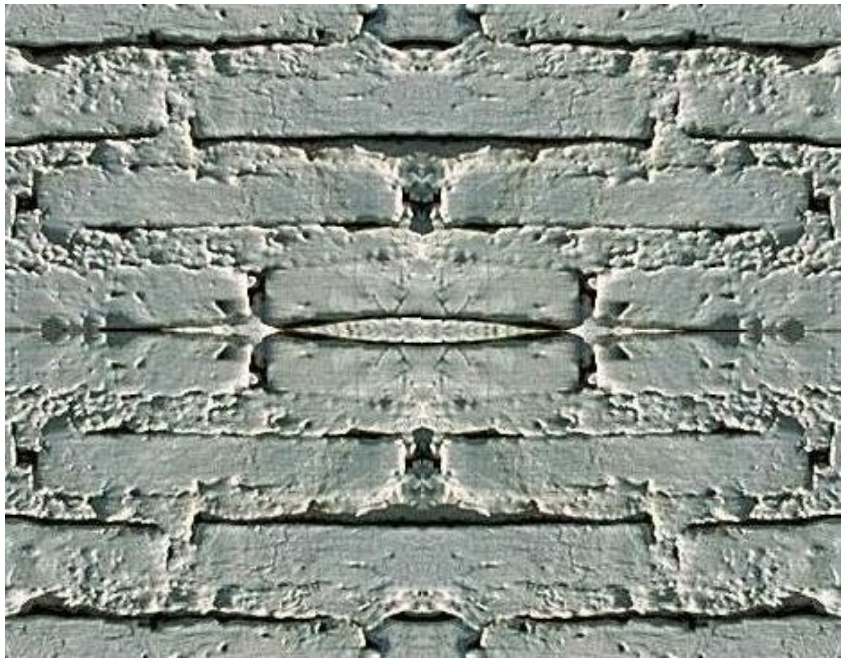
# Synthesis by Tiling

- Repeat pattern
  - reduce seems by mirroring

# Synthesis by Tiling

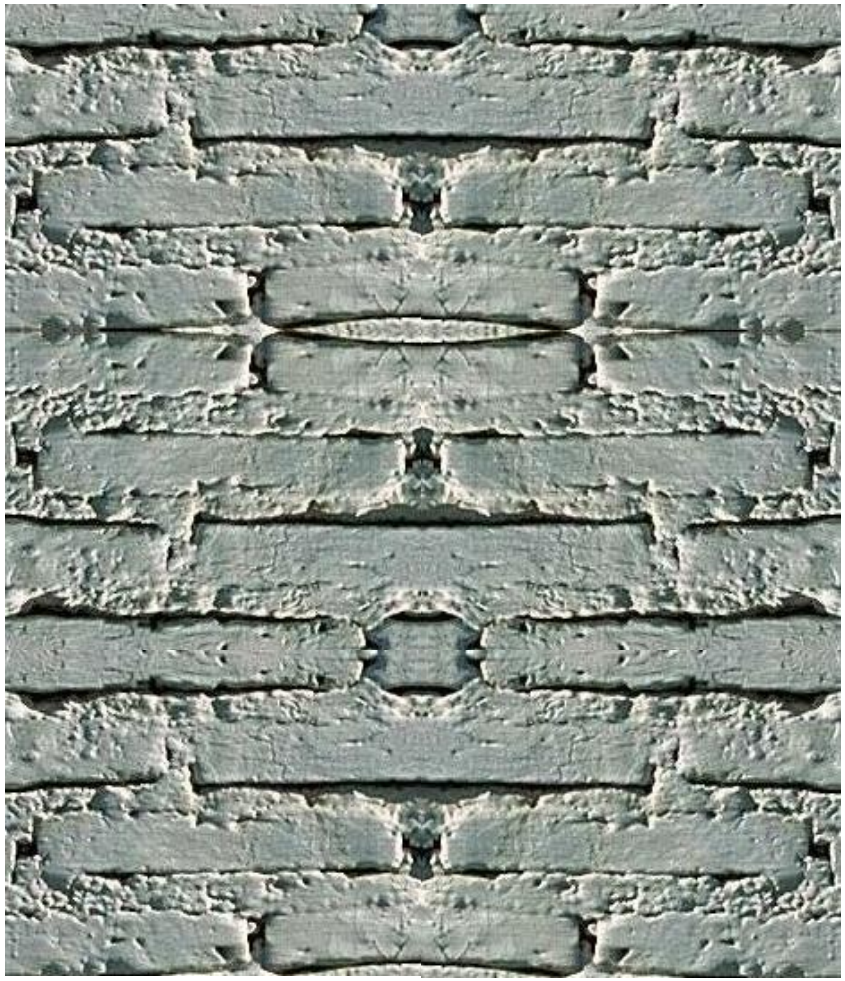
- Repeat pattern
  - reduce seems by mirroring

# Synthesis by Tiling



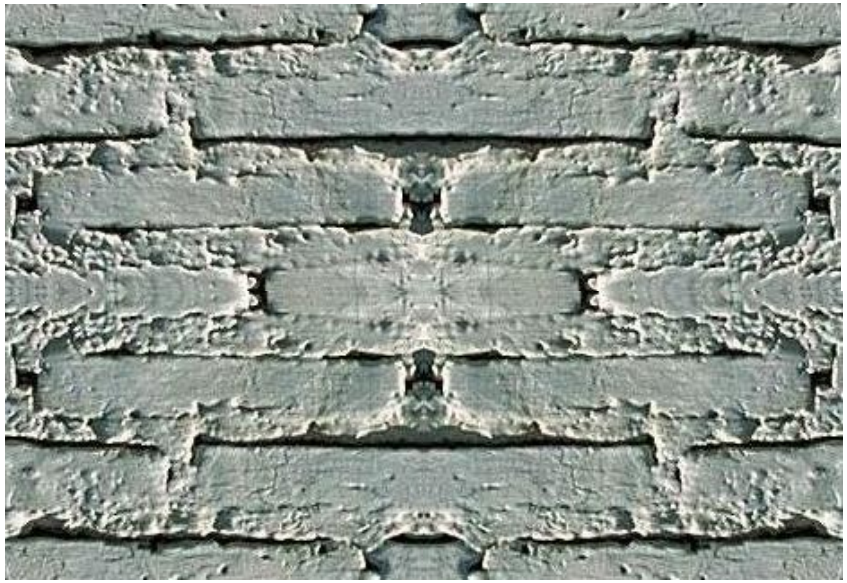
- Repeat pattern
  - reduce seems by mirroring

# Synthesis by Tiling



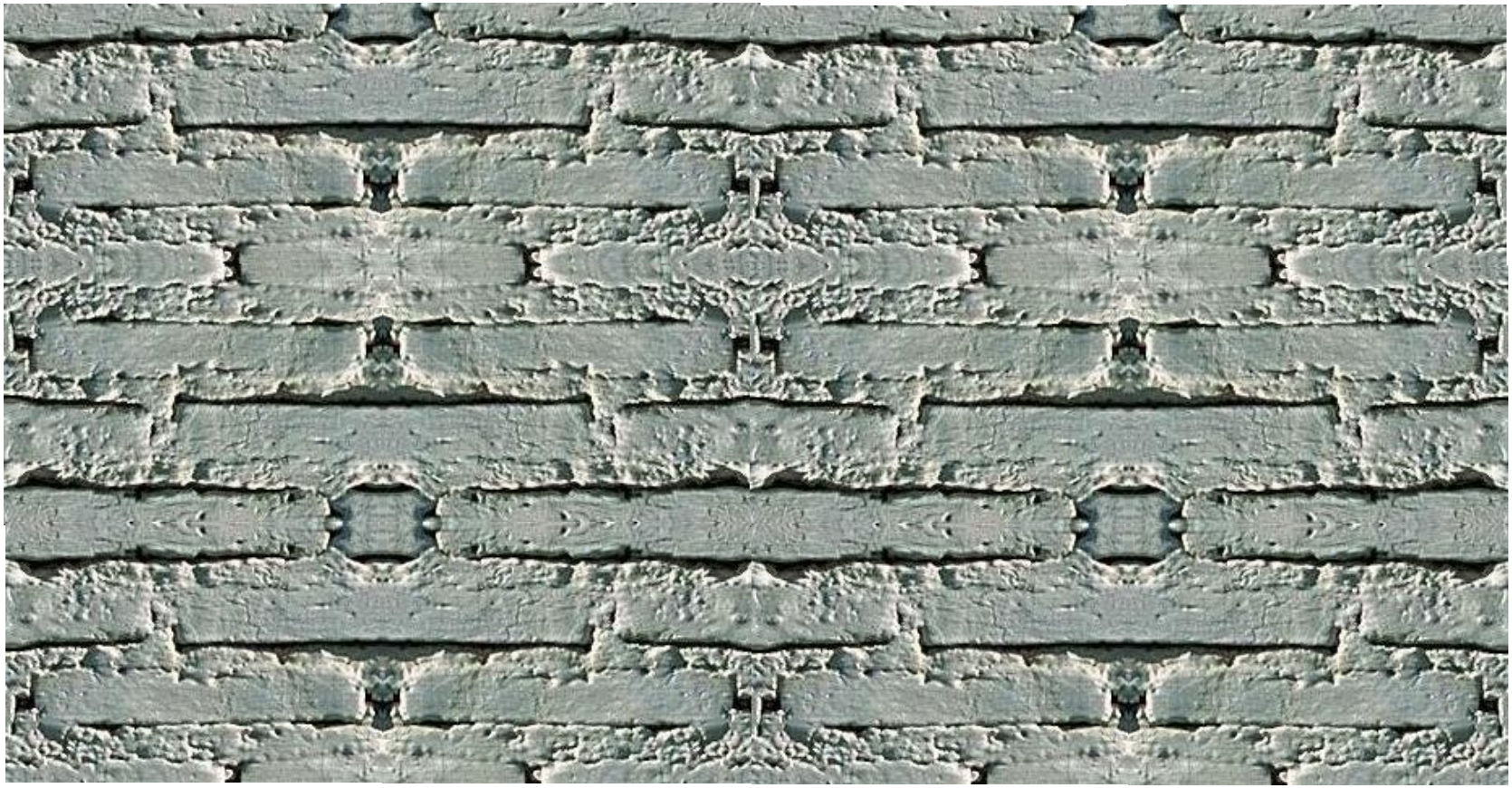
- Repeat pattern
  - reduce seems by mirroring
  - How we can further improve?

# Synthesis by Tiling



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

# Synthesis by Tiling



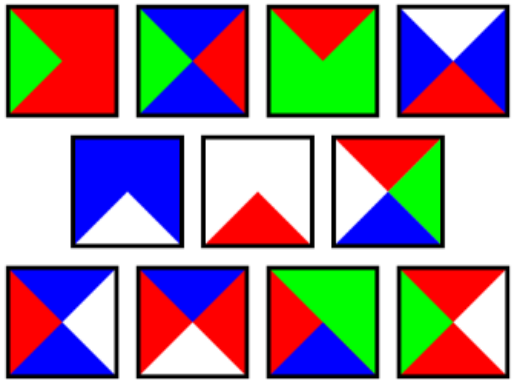
# Bricks are similar but not identical

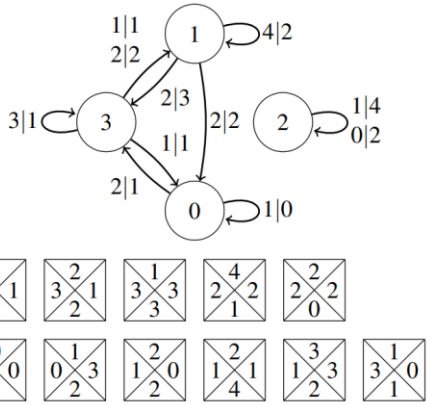
# Solution?

# Wang Tiles

- Can a set of (Wang) tiles fill an infinite plane?
  - In general, answer is undecidable
- Is the tiling periodic or aperiodic?
  - Both are possible

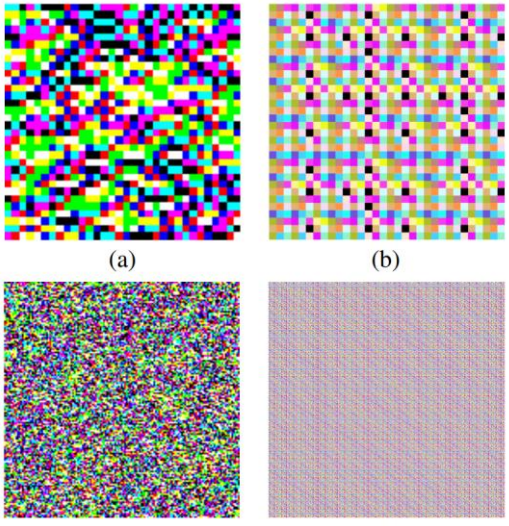
- Smallest set for aperiodic tiling: 11





# Wang Tiles [Cohen et al.]

- 8 tiles to stochastically tile the plane



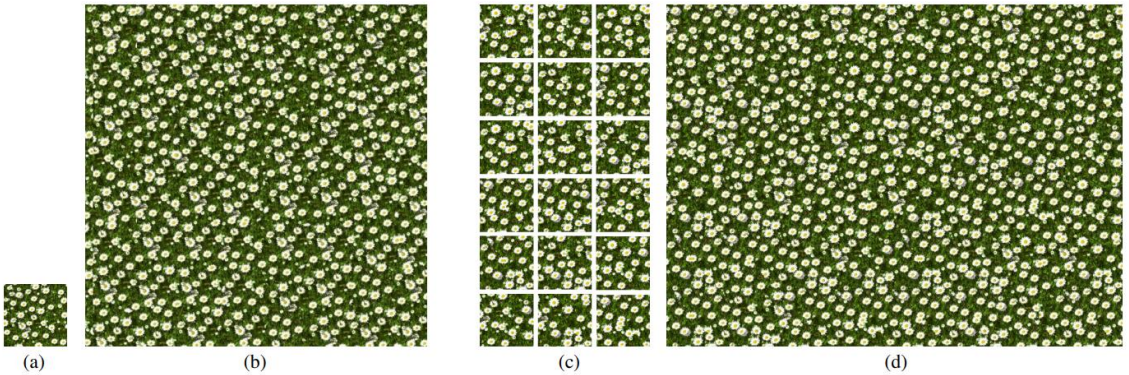




Figure 6: a) Source image; b) texture generated using image quilting; c) 18 Wang Tiles automatically generated based on set in Figure 1(d); d) resulting part of infinite texture (8×6 tiles) with some tile instances highlighted.

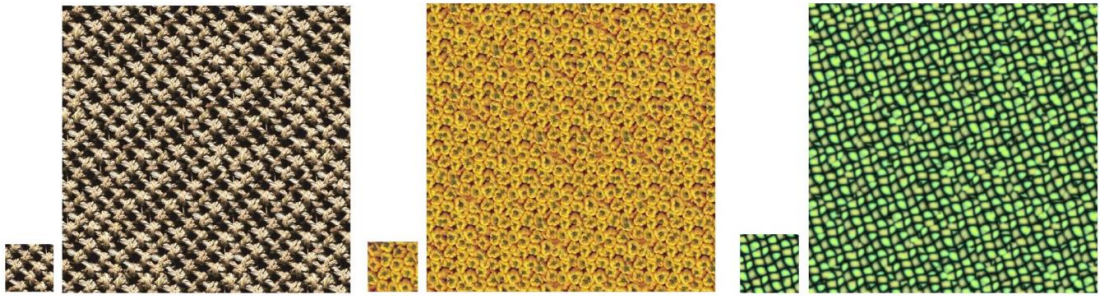
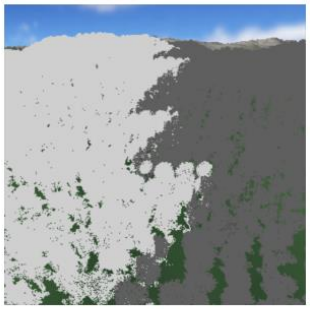


Figure 7: Three textures using 18 Wang Tiles.

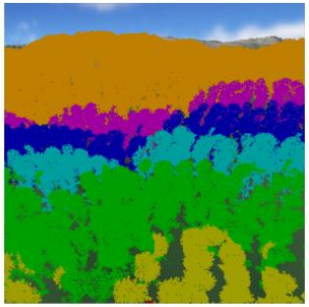
# Wang Tiles [Cohen et al.]

Figure 12: A Wang tiled field filled with sunflowers

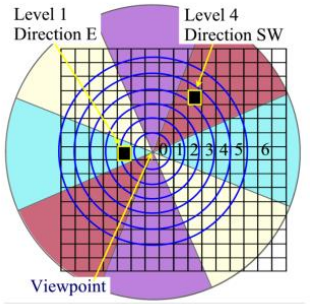
(a)



(b)



(c)

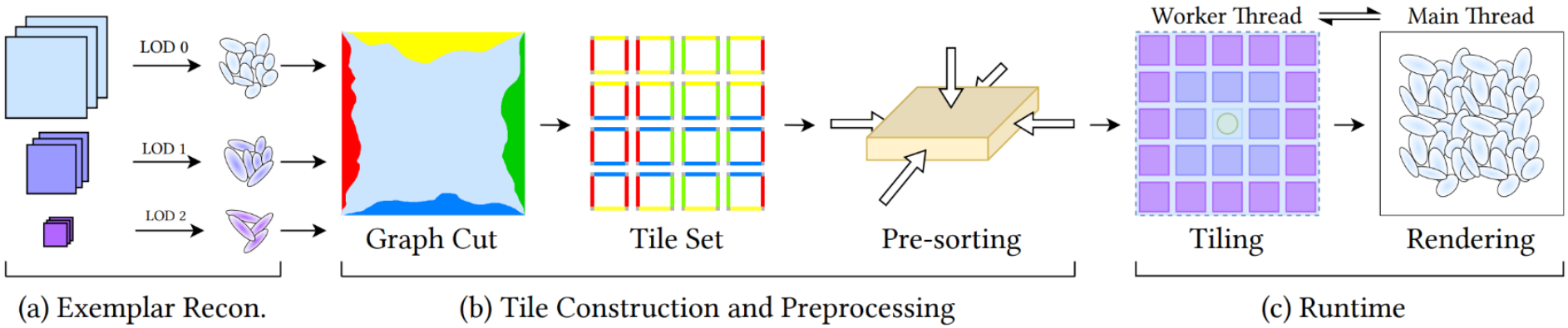



(d)

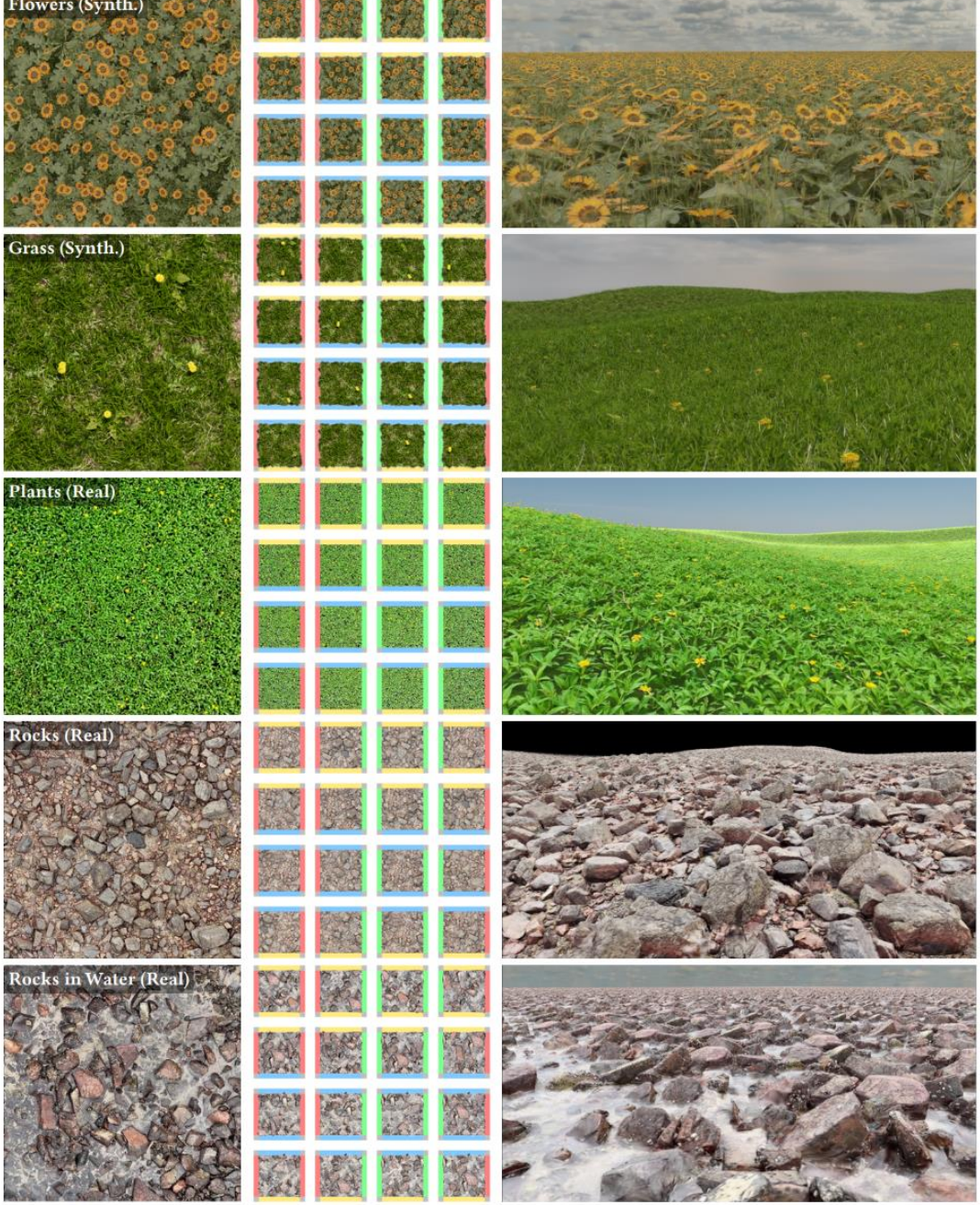
(e)

# Gaussian Splatting Wang Tiles

- Zeng et al. 2025




- https://yunfan.zone/gswt_webpage/

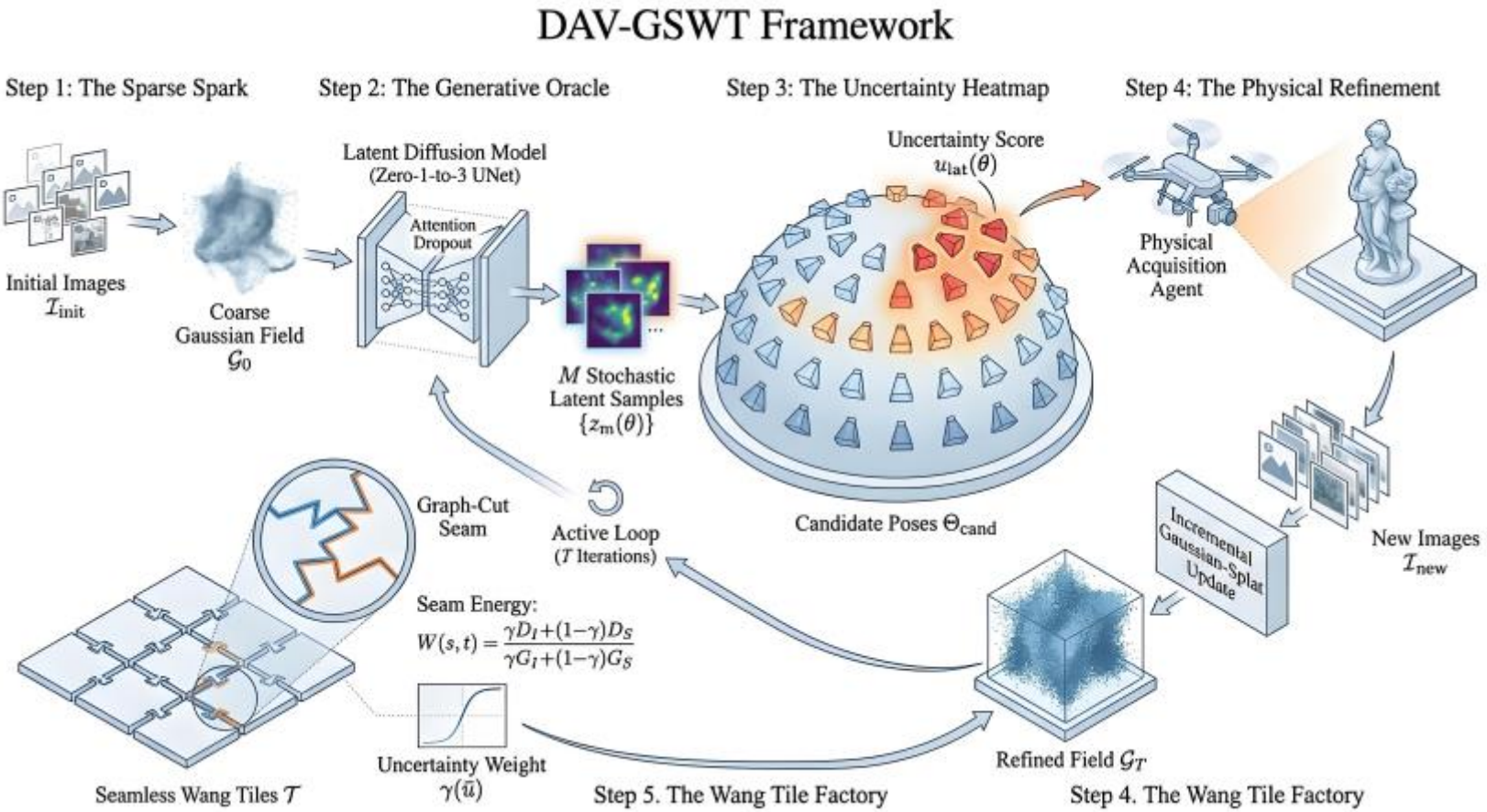



(a) Exemplar (b) Wang Tiles (c) Rendered scene

Fig. 9. Experimental results: (a) the exempla -coded edges, and (c) real-time rendering result. Please refer to the video for animated re

# Diffusion-based Generation of Wang Tiles from Sample Input

- Fu et al. 2026



w2 + LPIPS

GT

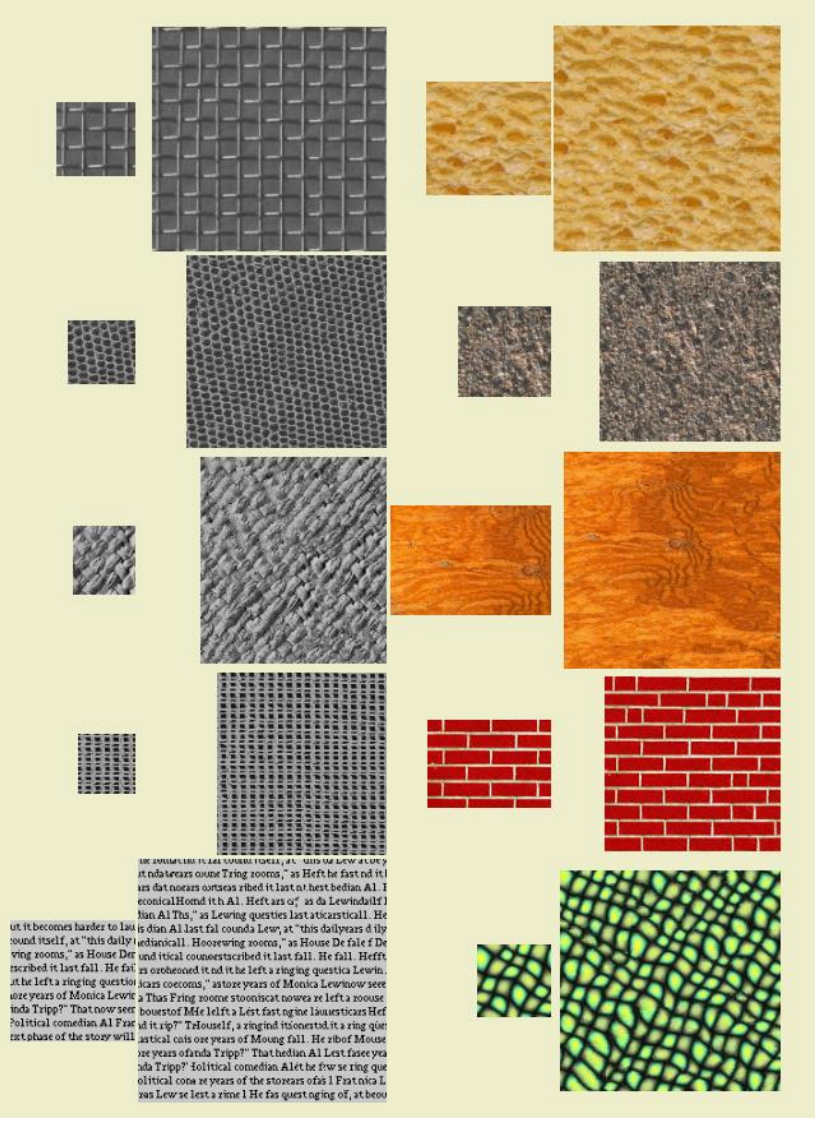
Image-grad + LPIPS
Seam-LPIPS: 0.039

w2 only
Seam-LPIPS: 0.034

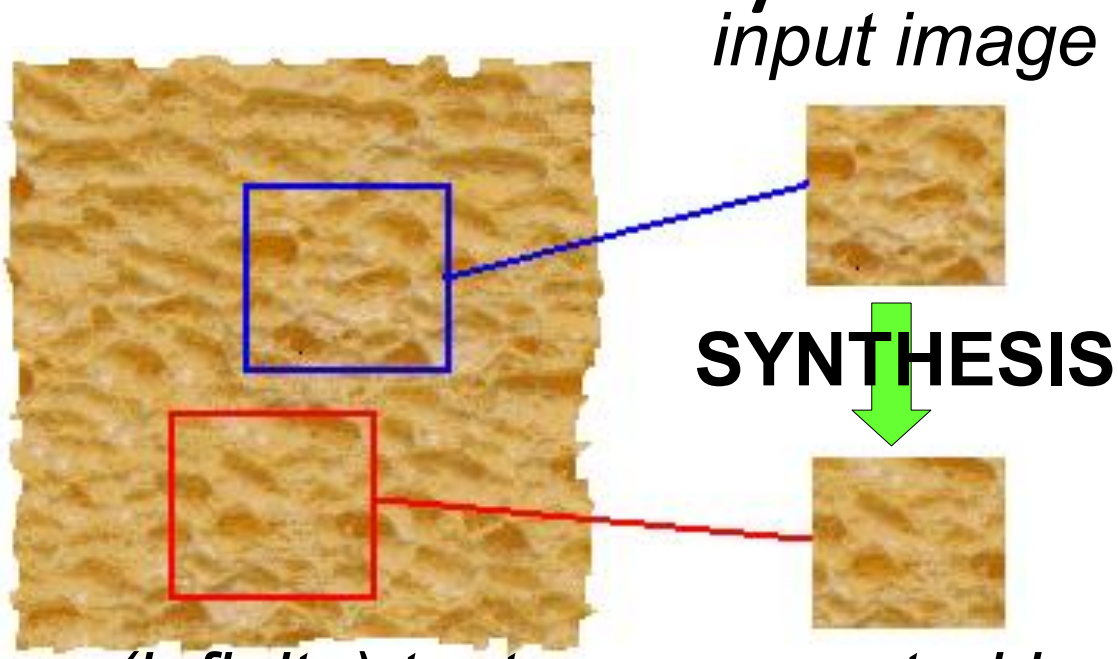
# Texture Synthesis

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

(slides from their presentation)



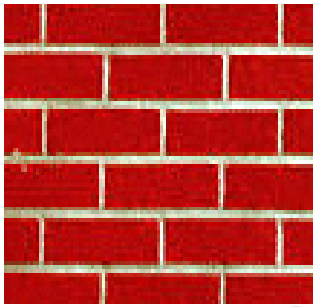
# Goal of Texture Synthesis



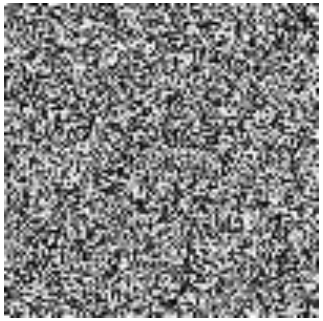

- Given a finite sample of some texture, the goal is to synthesize other samples from that same texture.
  - The sample needs to be "large enough"

# 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?

# 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

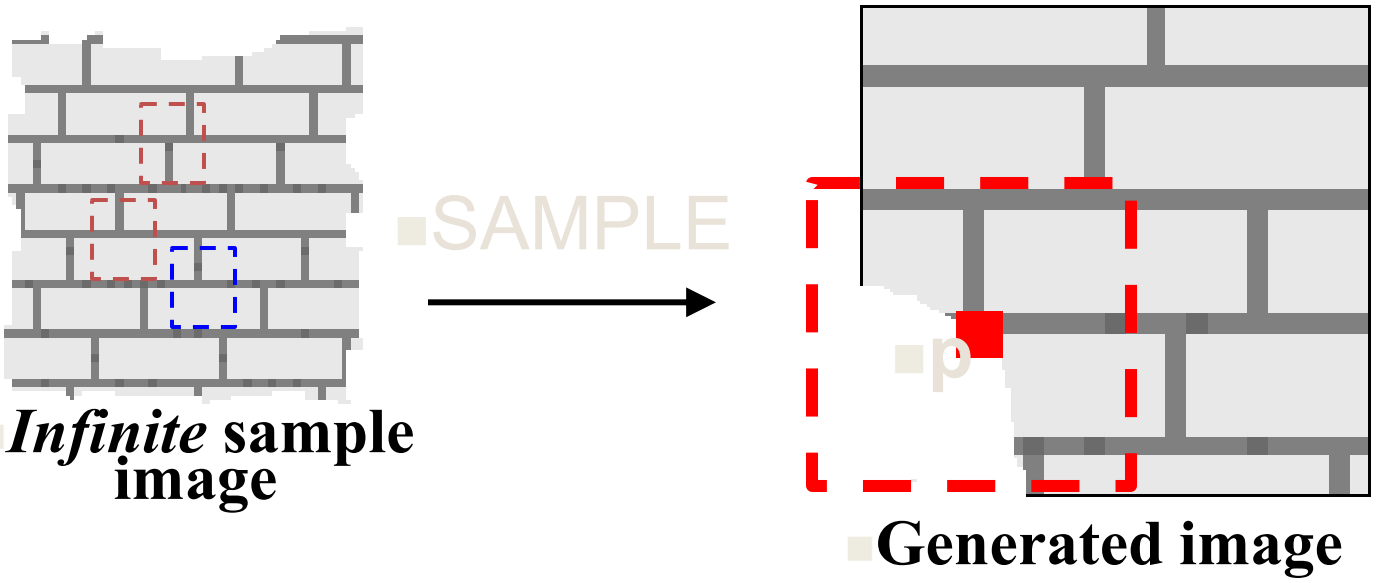

- [Shannon,'48] proposed a way to generate English-looking text using N-grams:
  - Assume a generalized Markov model (i.e., the next state is only dependent on the current state and is independent of anything in the past)
  - Use a large text to compute probability distributions of each letter given 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)

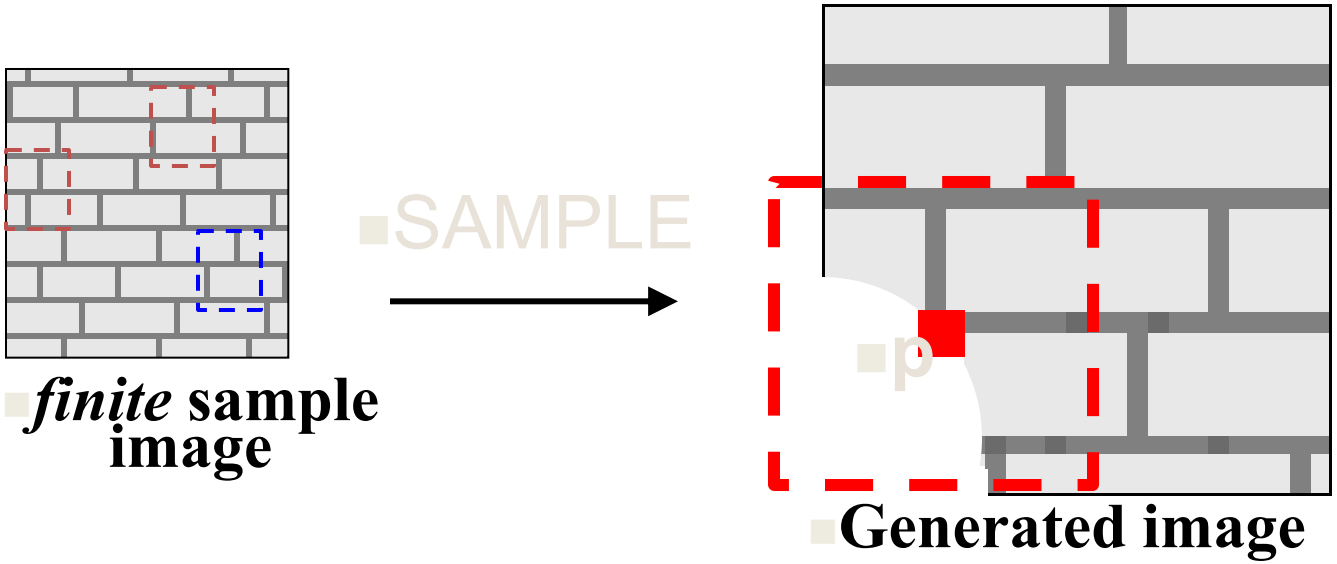

- Results (using alt.singles corpus):
  - *“As I've commented before, really relating to someone involves standing next to impossible.”*
  - *"One morning I shot an elephant in my arms and kissed him.”*
  - *"I spent an interesting evening recently with a grain of salt"*
- Notice how well local structure 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

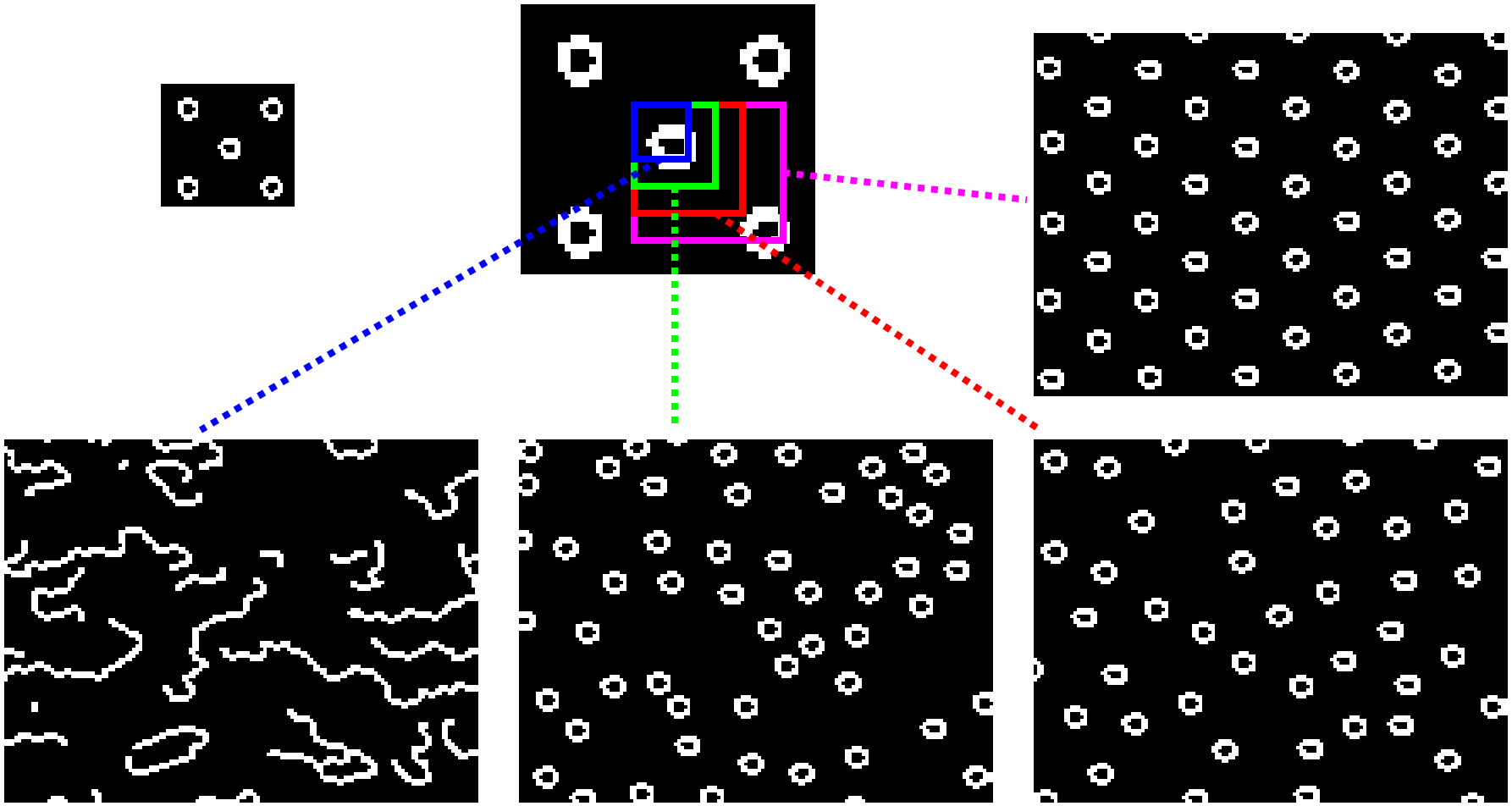
# Growing Texture


- 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

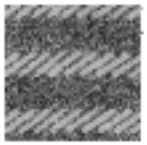
# Some Details

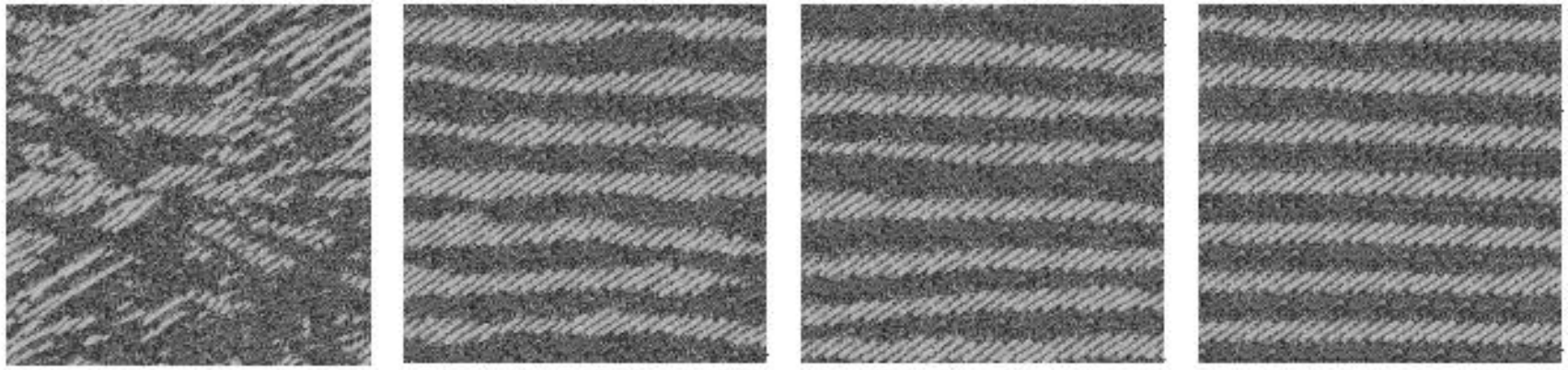

- Growing is in “onion skin” order
  - Within each “layer”, pixels with most neighbors are synthesized first
  - If no close match can be found, the pixel is not synthesized until the end
- Using *Gaussian-weighted* SSD is very important
  - to make sure the new pixel agrees with its closest neighbors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

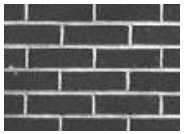
# Randomness Parameter

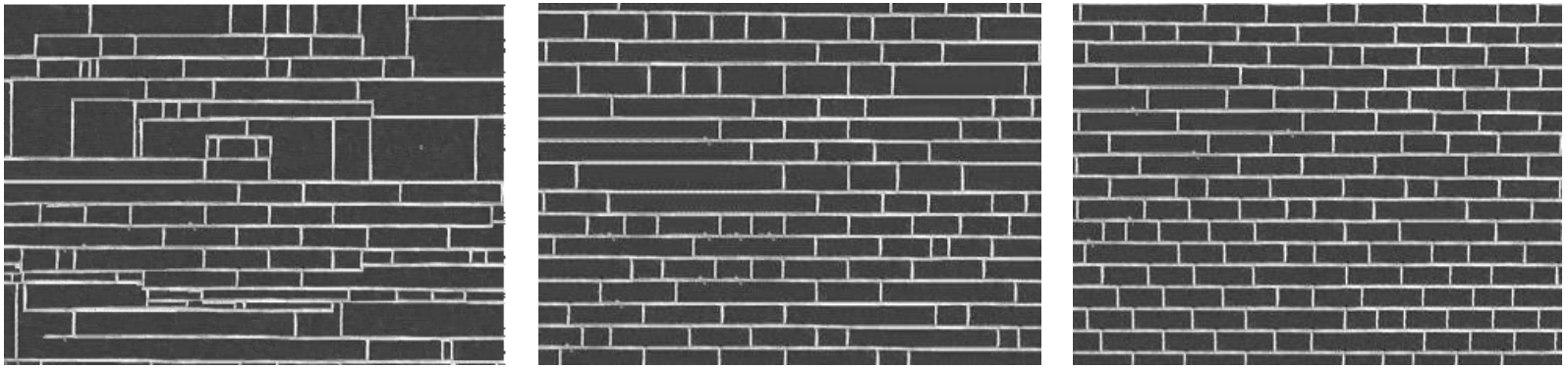


# More Synthesis Results



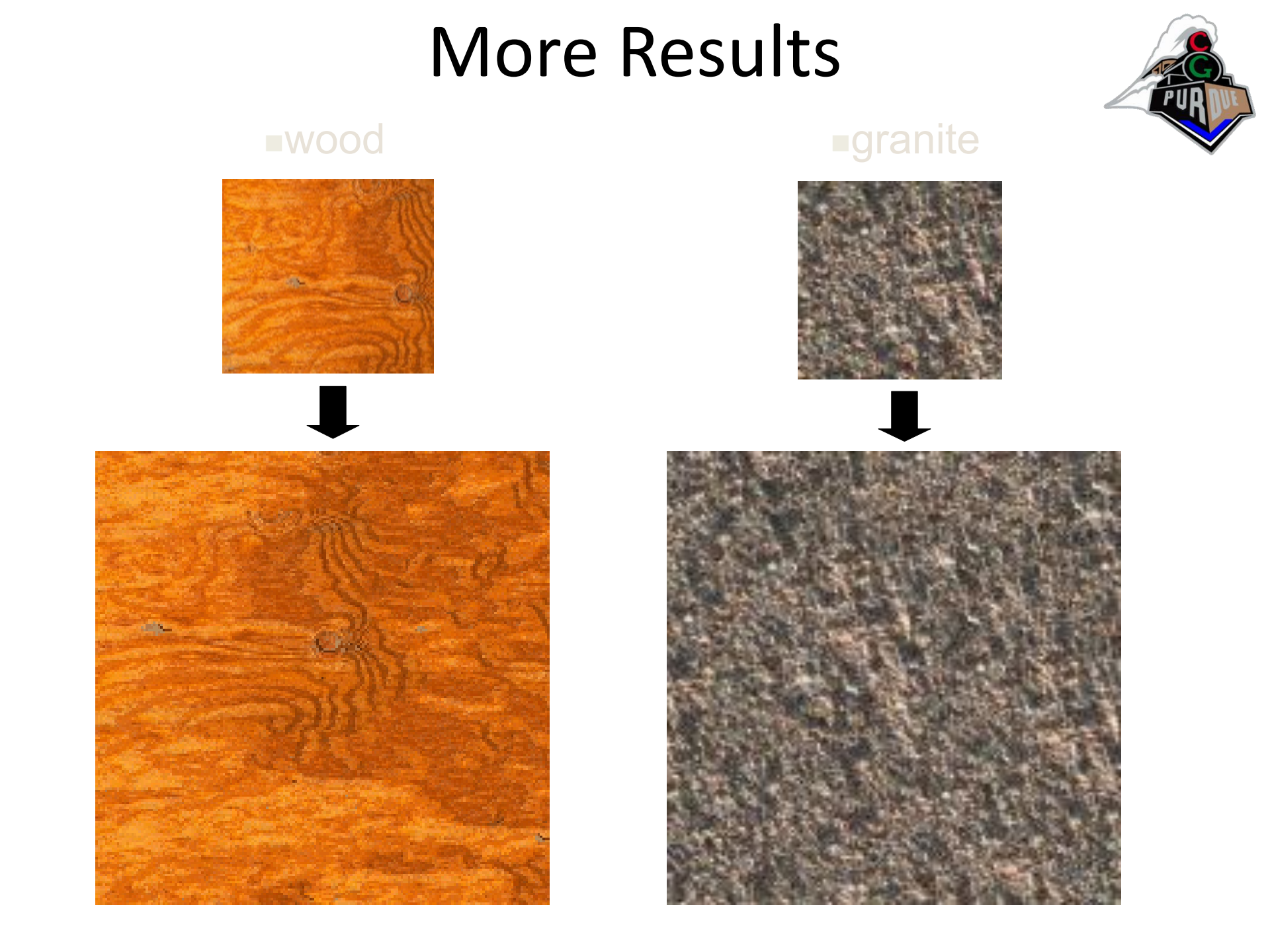






Increasing window size

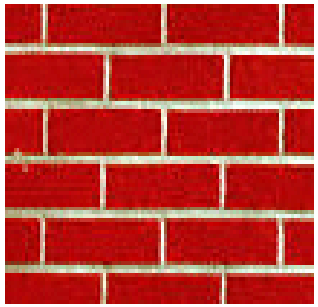
# More Results

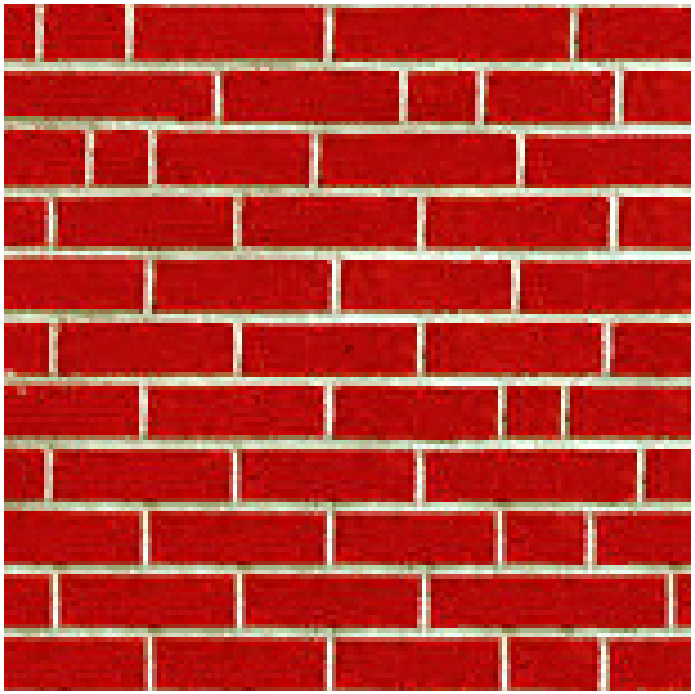


# More Results

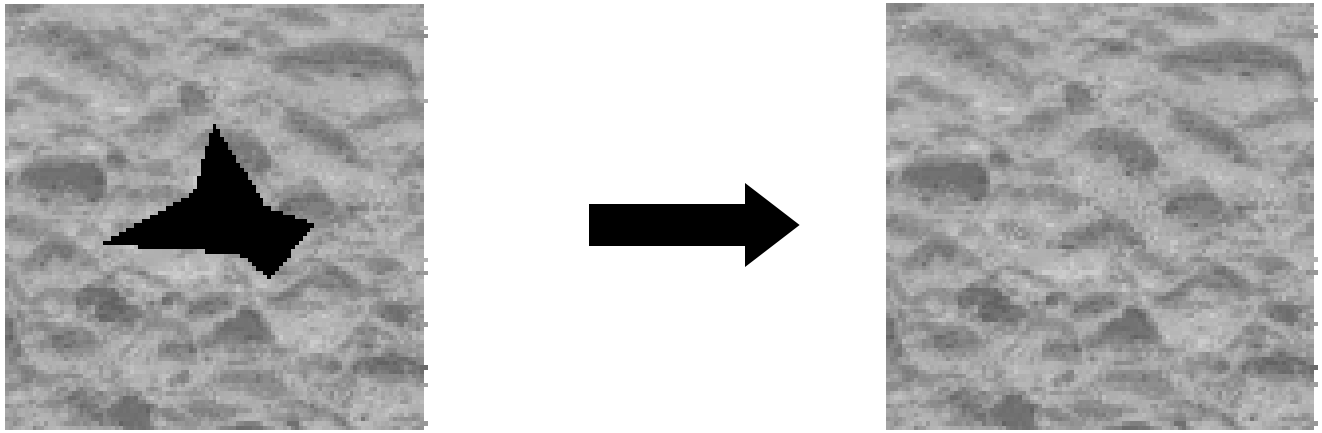
- white bread

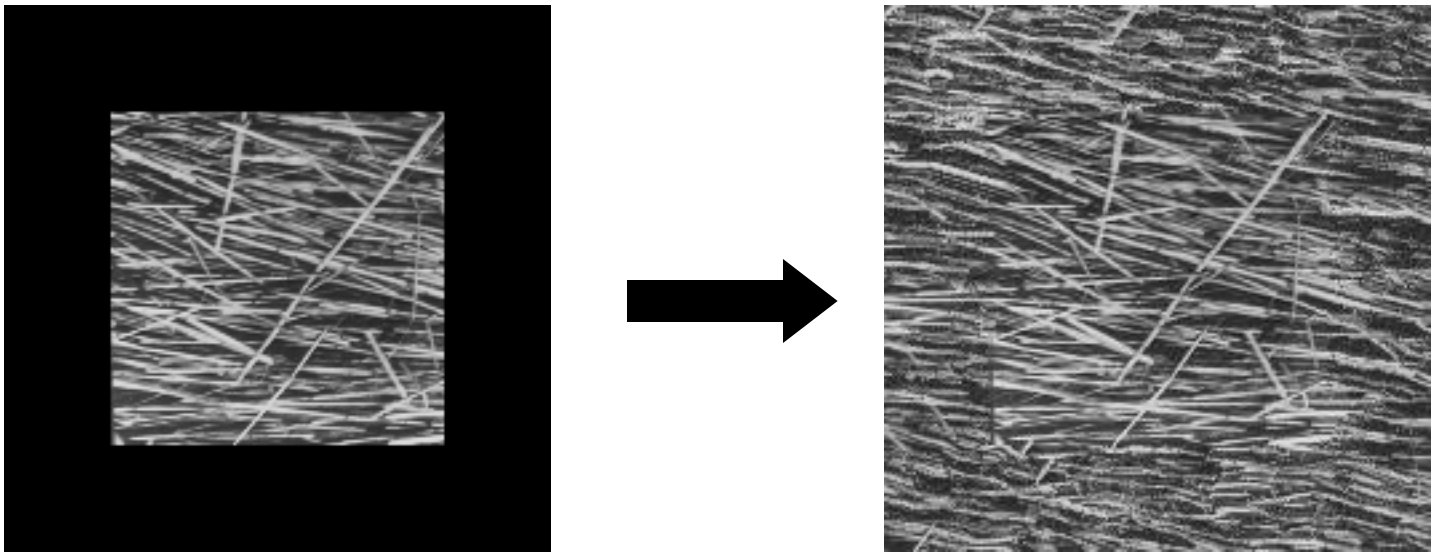
- brick wall





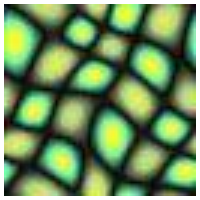
# Constrained Synthesis

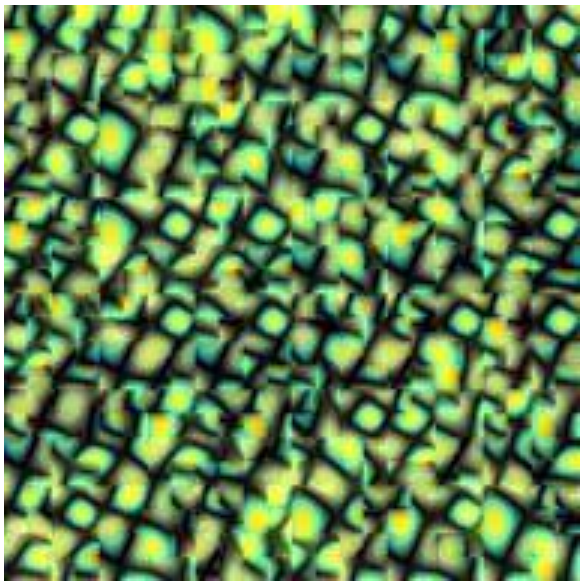




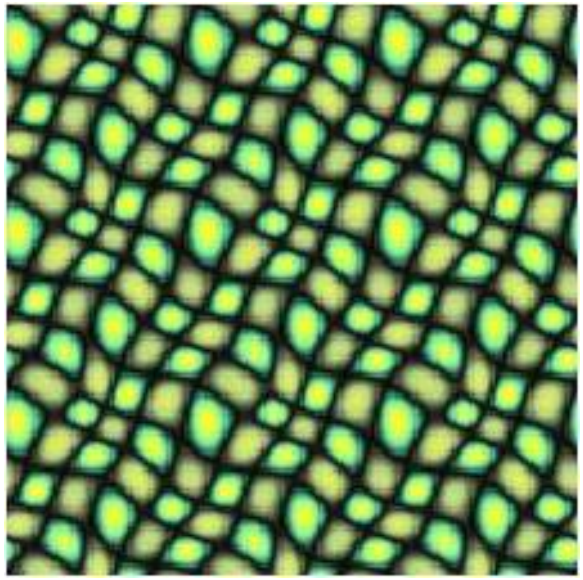
# Visual Comparison

- *Synthetic tilable texture*

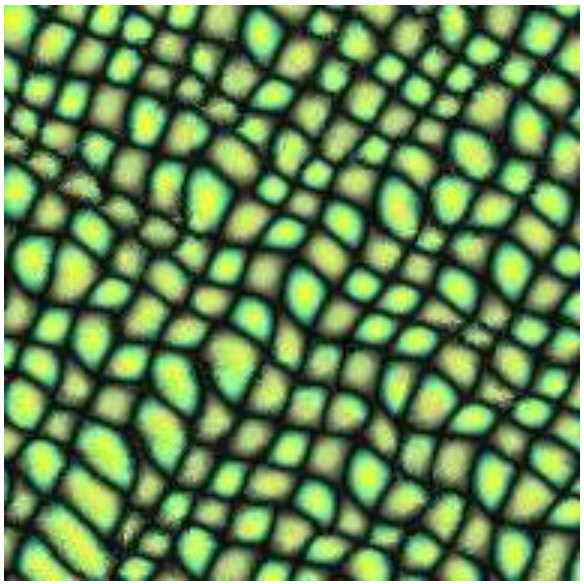




- [DeBonet, '97]

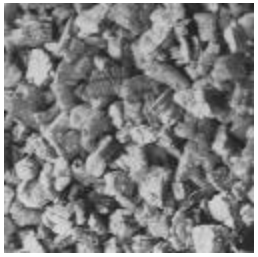


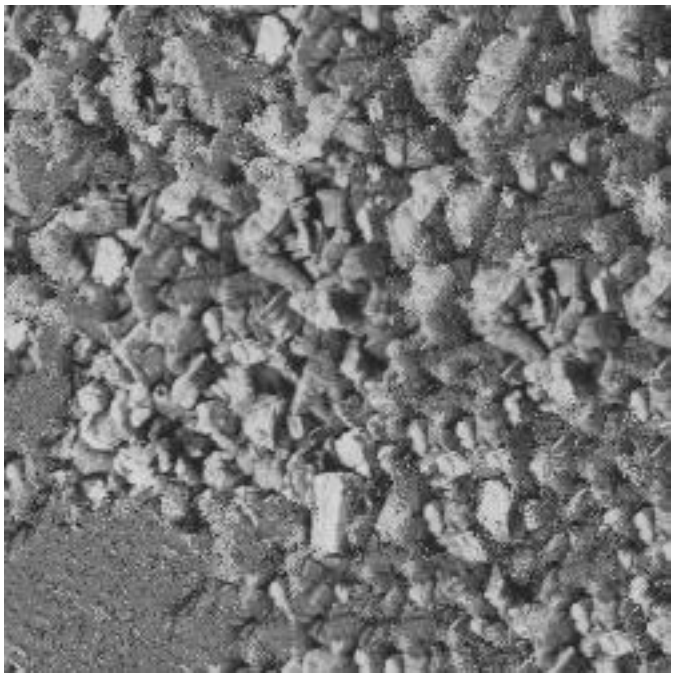
- Simple tiling



- Our approach

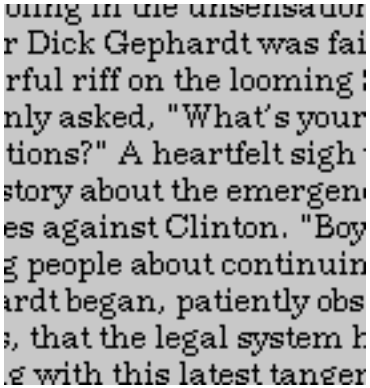
# Failure Cases

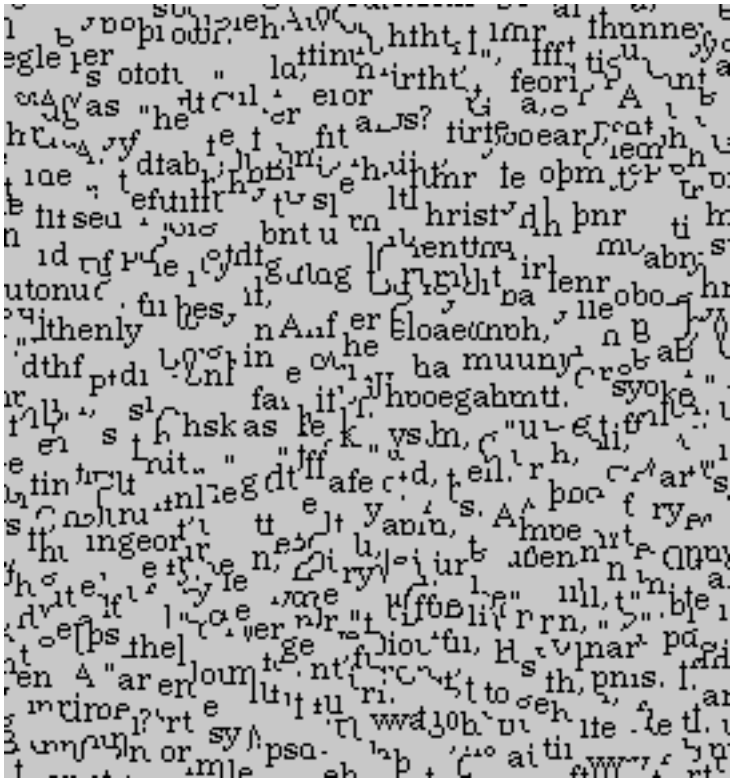




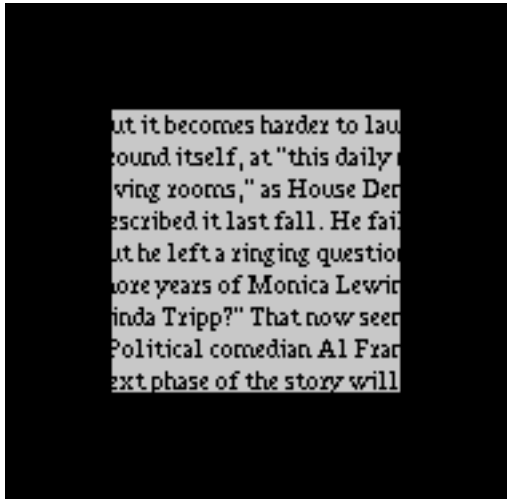
- Growing garbage
- Verbatim copying

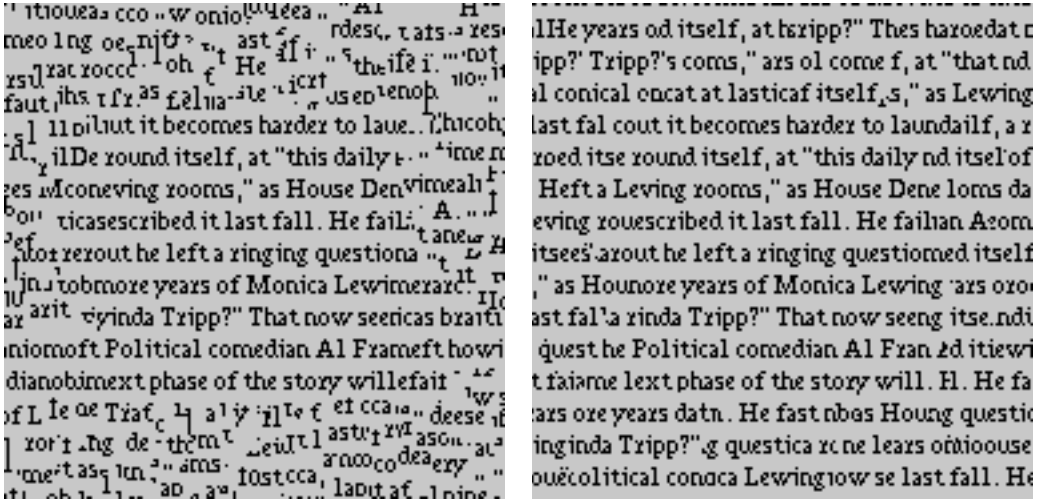
# Homage to Shannon

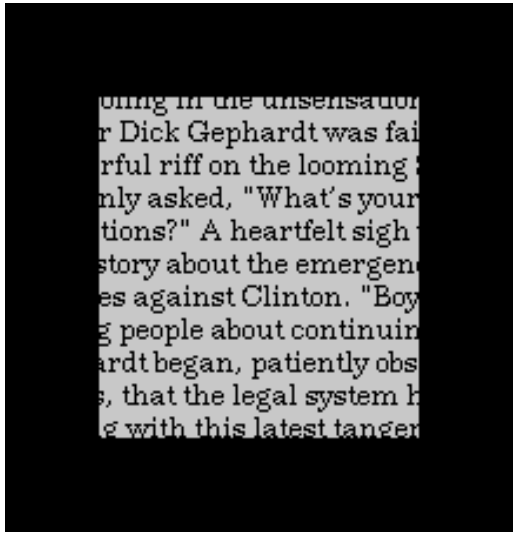


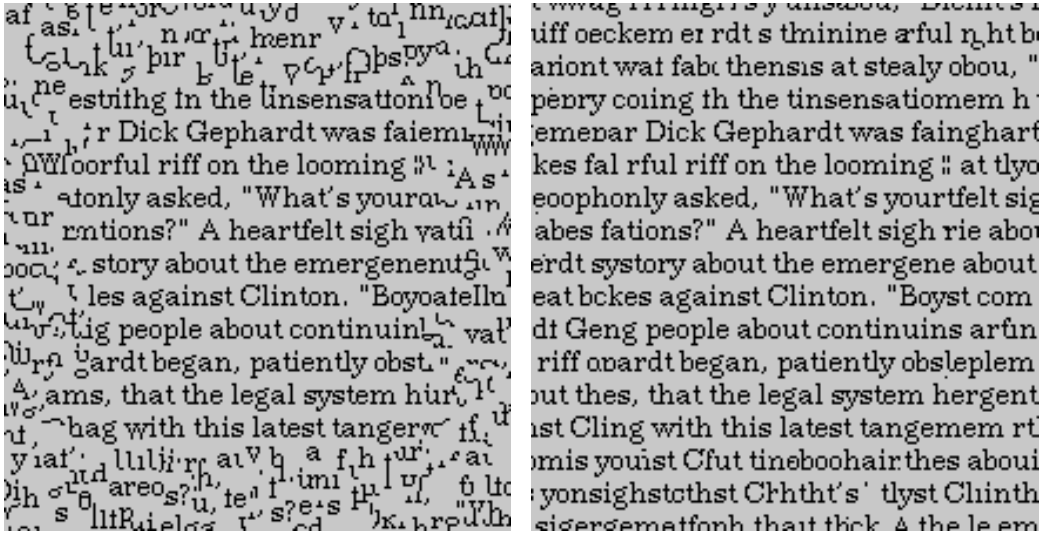



# Constrained Text Synthesis





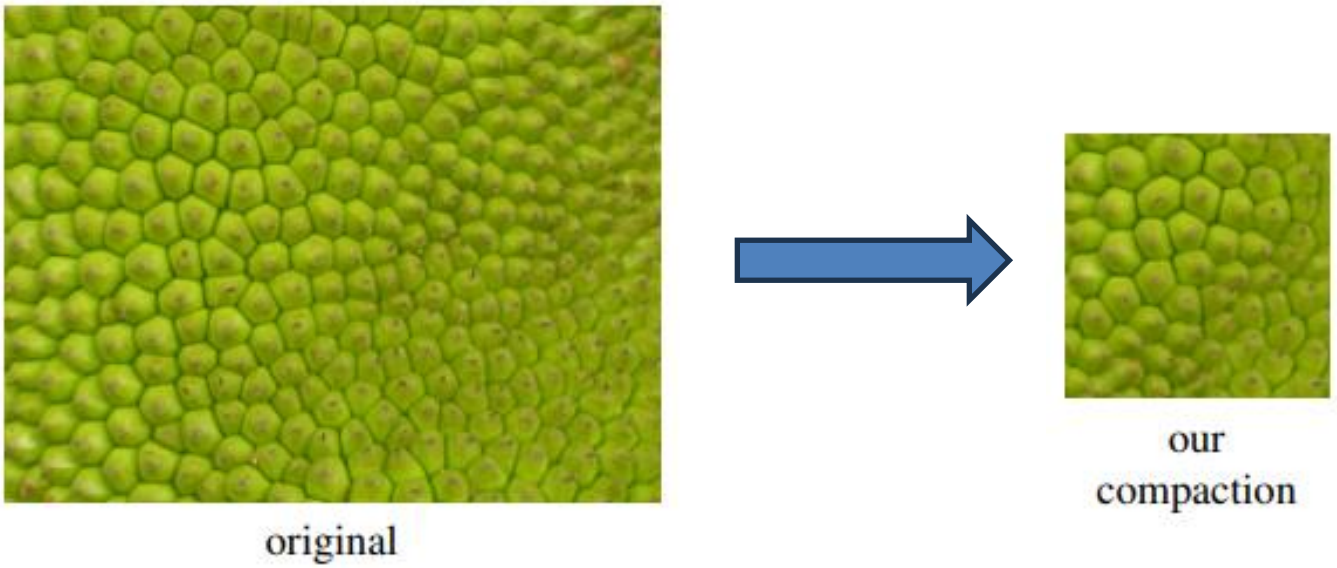




# Texture Synthesis $^{-1}$

# Inverse Texture Synthesis

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



# Inverse Texture Synthesis

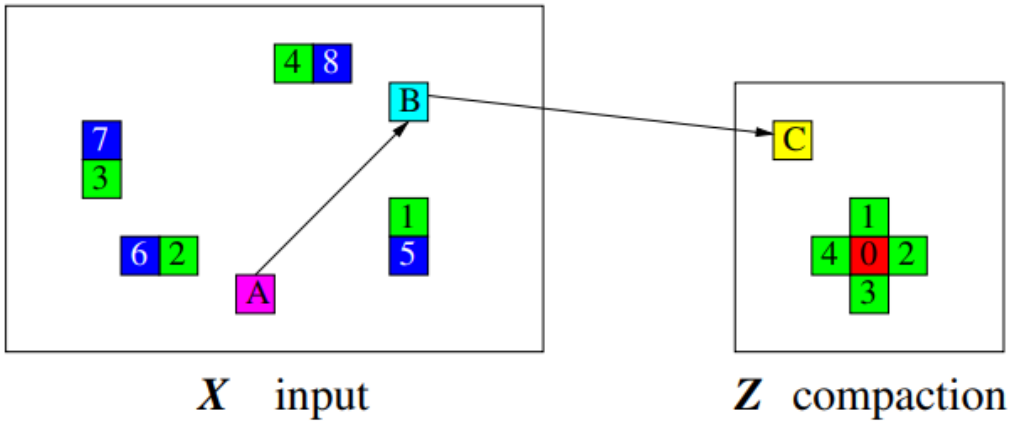



Figure 7: Illustrating of our improved solver. Here, we use a toy case with only four k-coherence neighbors as exemplified in pixels {**1, 2, 3, 4**} around **0** in the compaction. The sources of these four pixels are marked with the same numbers in the input. **z** E-step: the value of **0** is chosen from {**5, 6, 7, 8**} as determined by **0**'s neighbors {**1, 2, 3, 4**}. Forward M-step: the best match for **0** is also chosen from {**5, 6, 7, 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

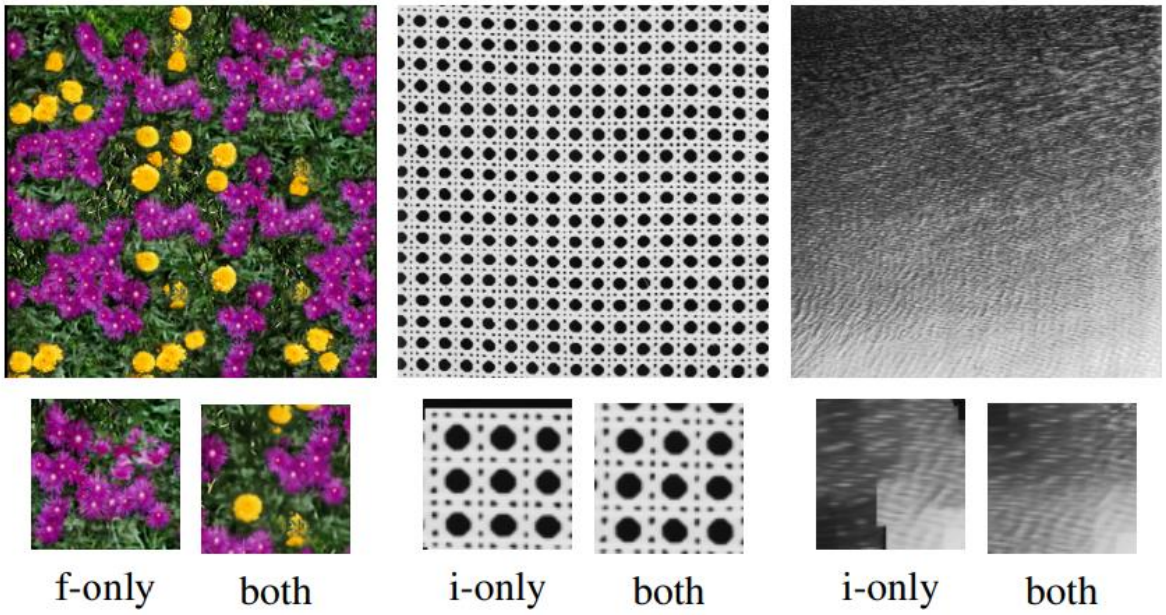



Figure 6: Why we need both forward and inverse terms in our energy function. With only the forward term the compaction will not provide sufficient coverage of the original (left case). With only the inverse term the compaction may contain garbage (middle case) or discontinuity (right case). All compactions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

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

# Content Generation

- Texture Synthesis
  - Make/replicate small fragments
- **Procedural Modeling**
  - **Make 3D models**
- Neural Rendering
- Generative Modeling

# Procedural Modeling

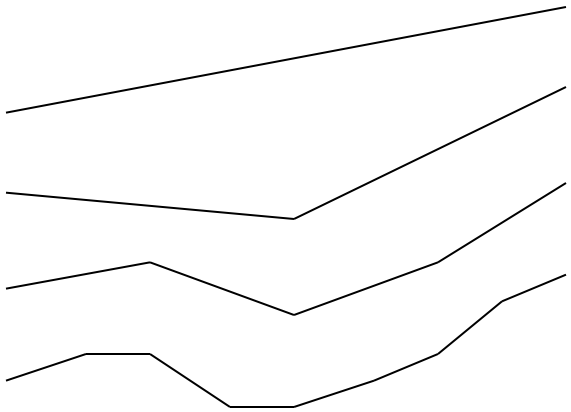

- Apply algorithms for producing objects and scenes
- The rules may either be embedded into the algorithm, configurable by parameters, or externally provided
- Key notions:
  - Detail amplification: from a little, make a lot, but not random
  - Kolmogorov Complexity: “a measure of the complexity or randomness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

# Procedural Modeling


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

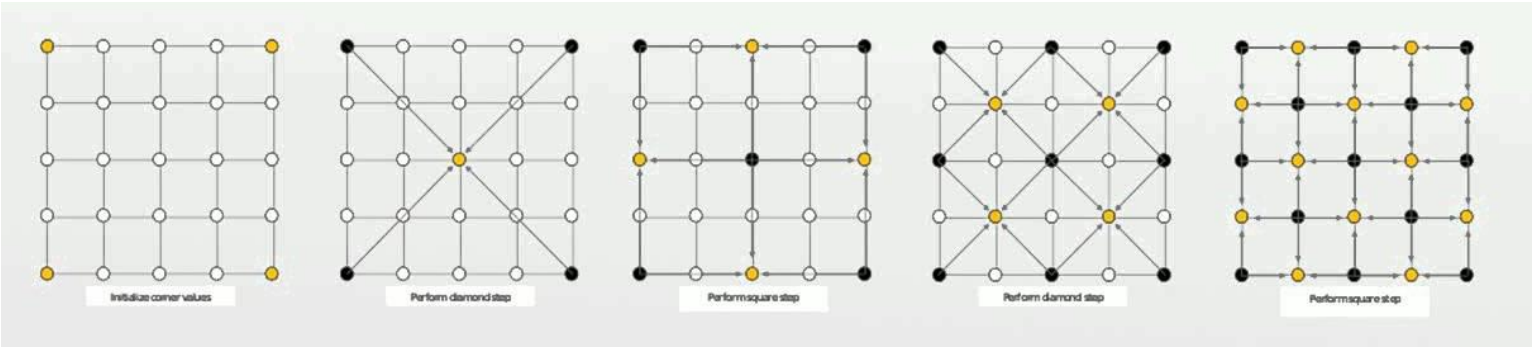
# Fractals

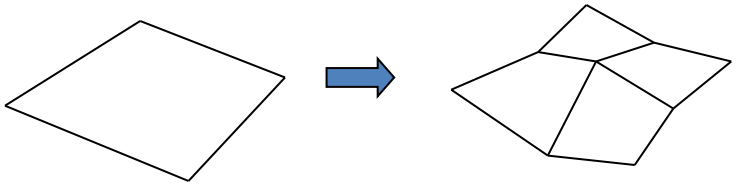
- Consider a simple line fractal
  - Split a line segment, randomize the height of the midpoint by some number in the [-*r*,*r*] range
  - Repeat and randomize by [-*r*/2,*r*/2]
  - Continue until a desired number of steps, randomizing by half as much each step



# Fractals and Terrains

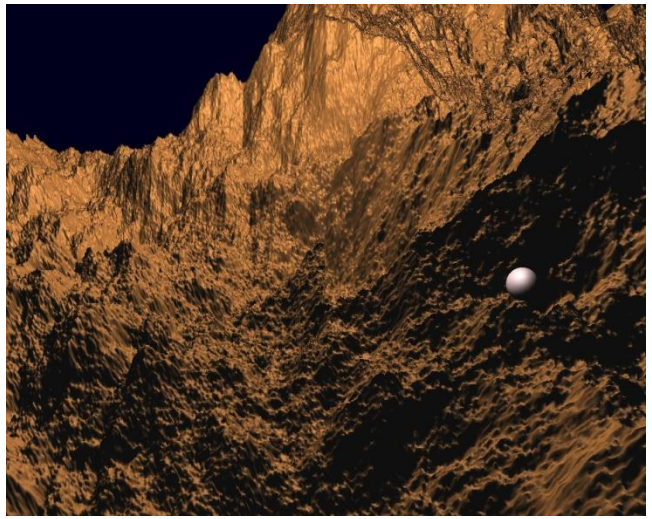
- A similar process can be applied to squares in the xz plane (Diamond-Square Algorithm):
  - At each step, an xz square is subdivided into 4 squares, and the y component of each new point is randomized
  - By repeating this process recursively, we can generate a mountain landscape



# 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

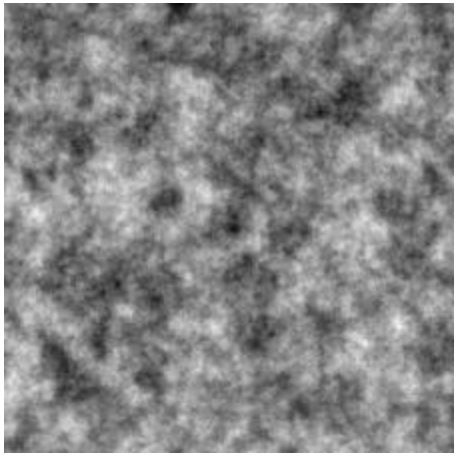


# Image Synthesis

- 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

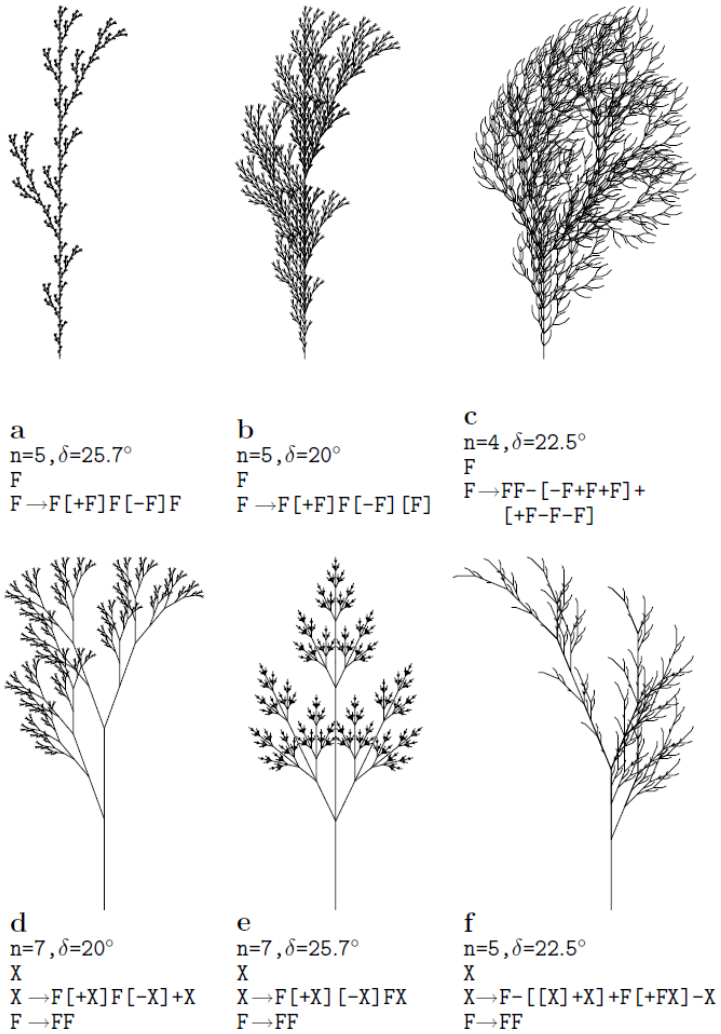
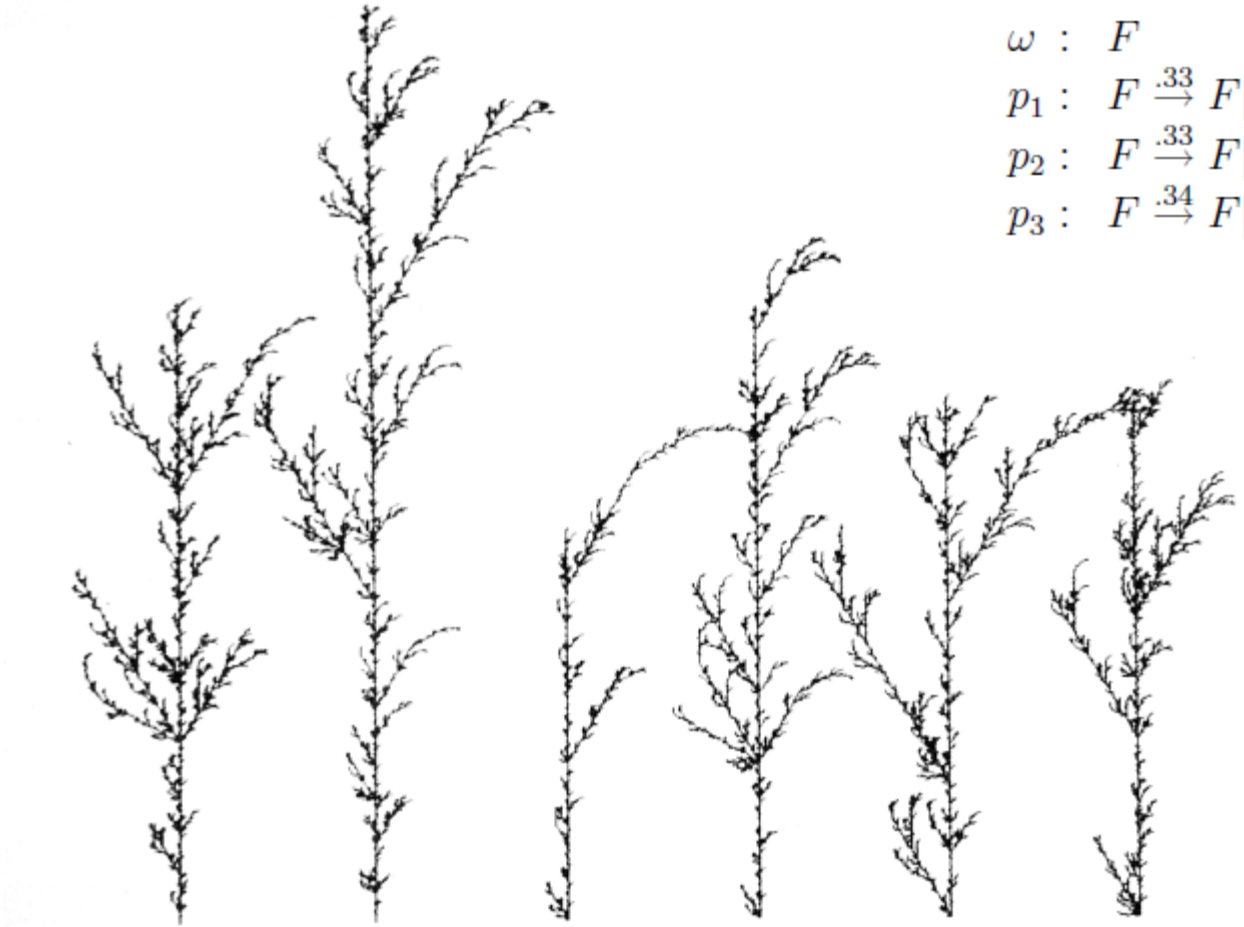



Figure 1.24: Examples of plant-like structures generated by bracketed OL-systems. L-systems (a), (b) and (c) are edge-rewriting, while (d), (e) and (f) are node-rewriting.

# L-system for Plants (stochastic)



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

Figure 1.27: Stochastic branching structures

# L-system for Plants (3D)

n=5, $\delta$=18°

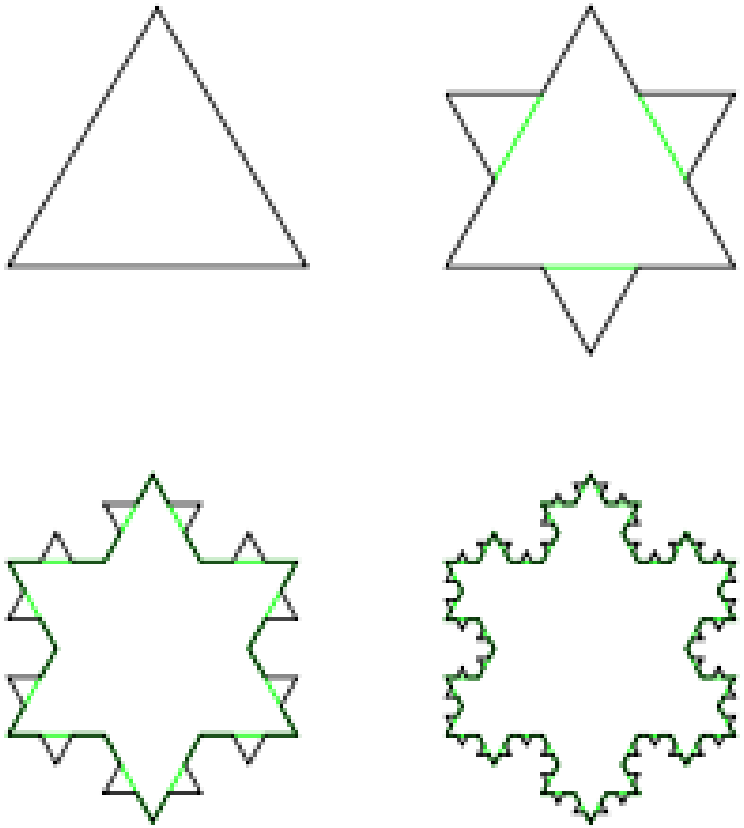
$\omega$ : plant
$p_1$ : plant → internode + [ plant + flower] − − //
[ − − leaf ] internode [ + + leaf ] −
[ plant flower ] + + plant flower
$p_2$ : internode → F seg [// & & leaf ] [// ∧ ∧ leaf ] F seg
$p_3$ : seg → seg F seg
$p_4$ : leaf → [' { +f−ff−f+ | +f−ff−f } ]
$p_5$ : flower → [ & & & pedicel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

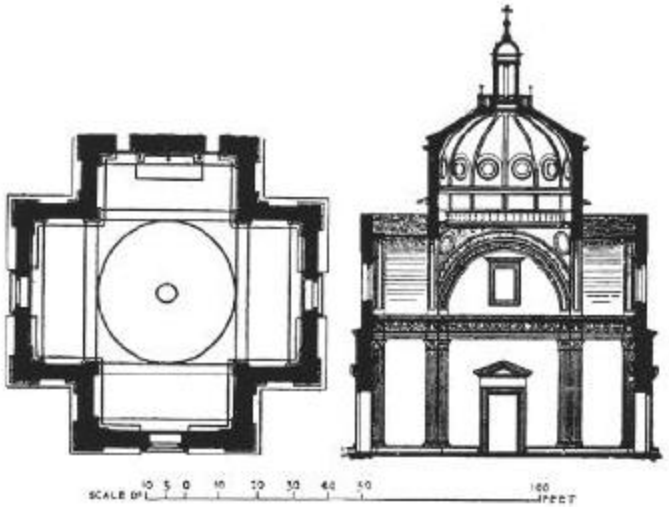
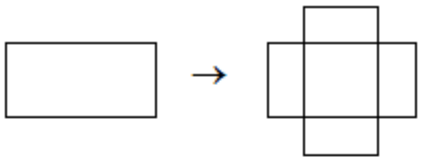


Illustration by Peter Murray, "the Artchitecture of the Italian Renaissance", Shocken Books Inc. 1963, Pp.96.

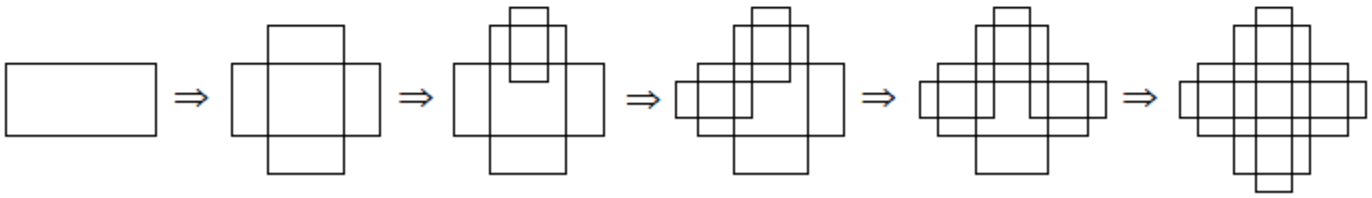
# Shape Grammar

# Shape Grammar

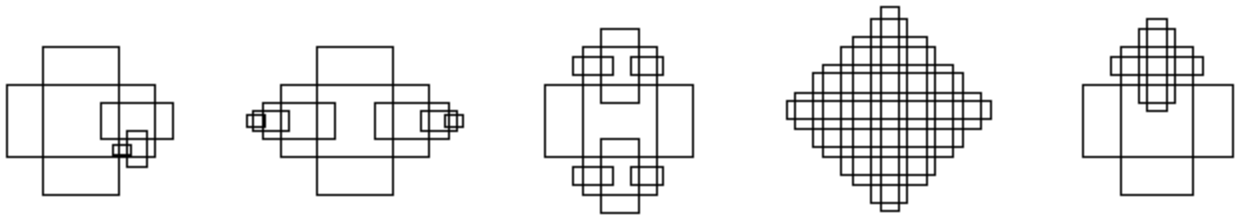


rule

DERIVATION



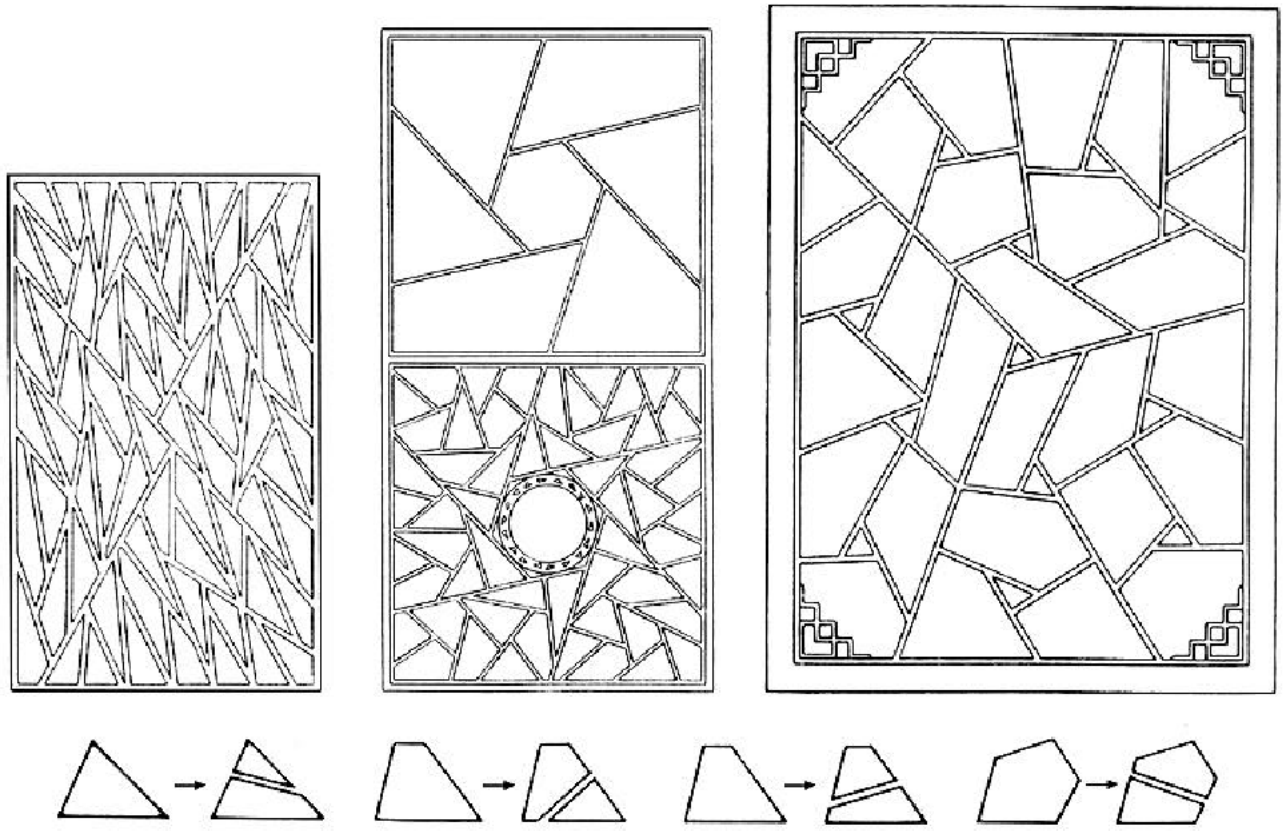
# Shape Grammar



OTHER DESIGNS IN THE LANGUAGE

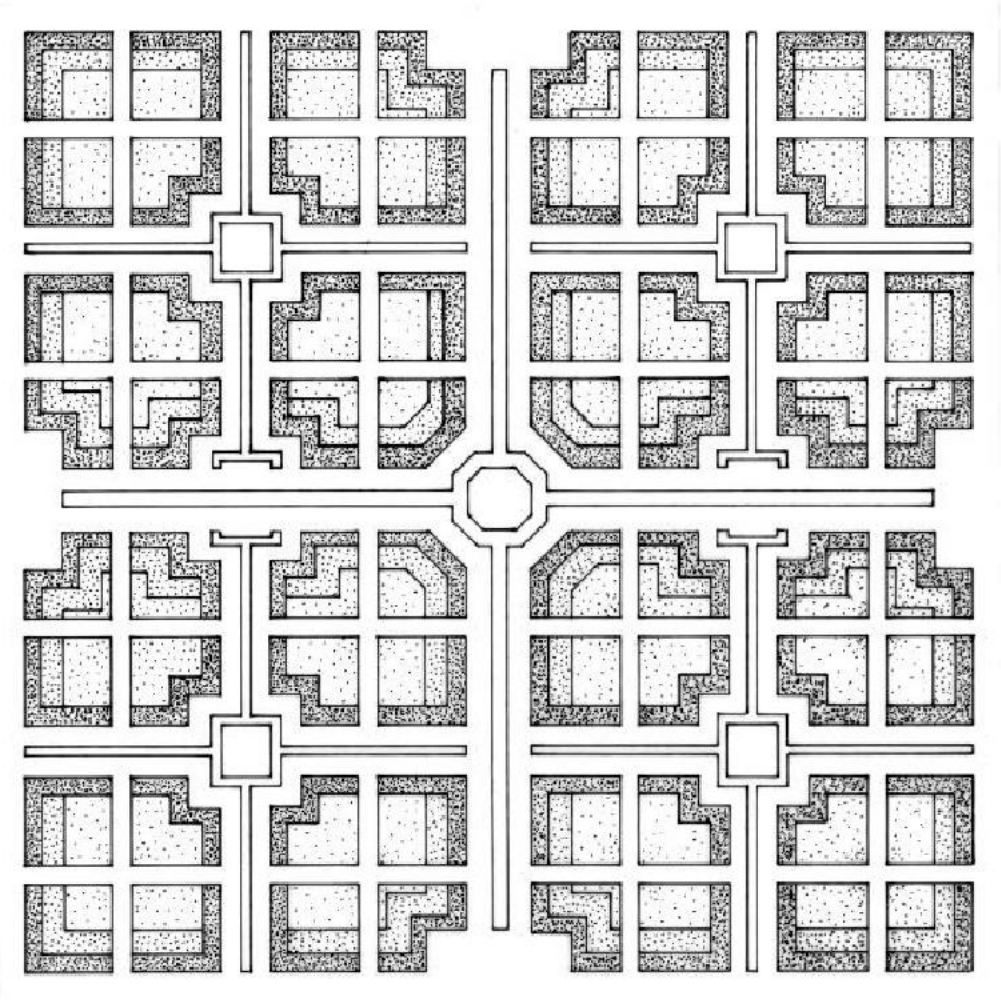
# Exercise: let’s make some art!

# Shape Grammar



Ice-ray grammar

# Shape Grammar



Mughul garden grammar

# Shape Grammar

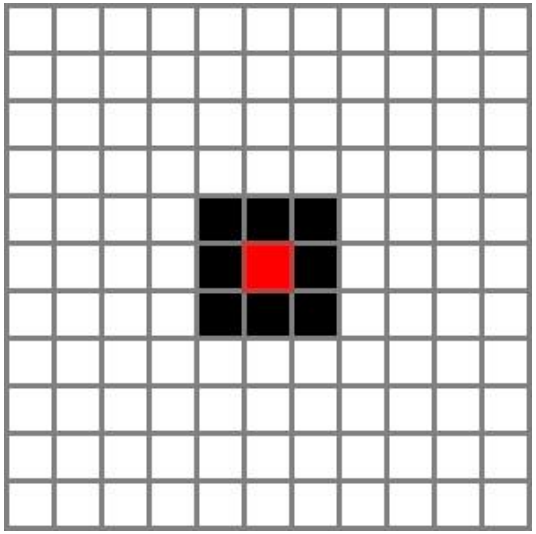
- Style: Mediterranean

# 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.



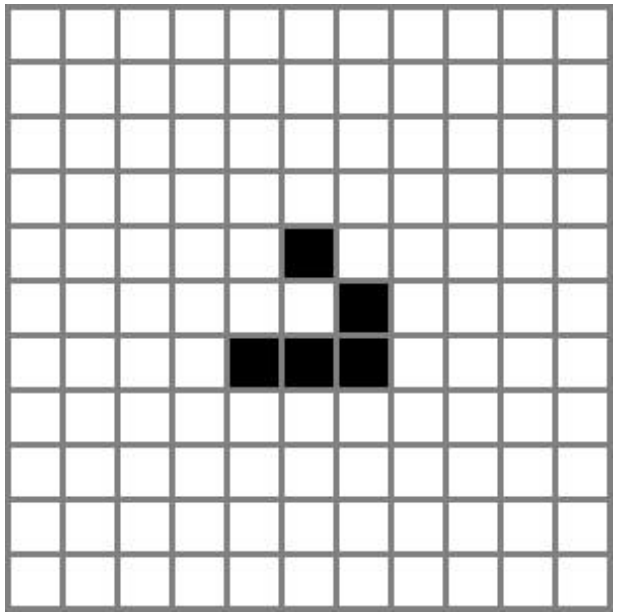
# John Conway's Game of Life


- 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

# Content Generation


- Texture Synthesis
  - Make/replicate small fragments
- Procedural Modeling
  - Make 3D models
- **Neural Rendering**
  - **Render using neural networks (first wave)**
- Generative Modeling

# Sources:

- State of the Art on Neural Rendering
  - Tewari, Fried, Thies, Sitzmann, Lombardi, Sunkavalli, Martin-Brualla, Simon, Saragih, Nießner, Pandey, Fanello, Wetzstein, Zhu, Theobalt, Agrawala, Shechtman, Goldman, Zollhöfer

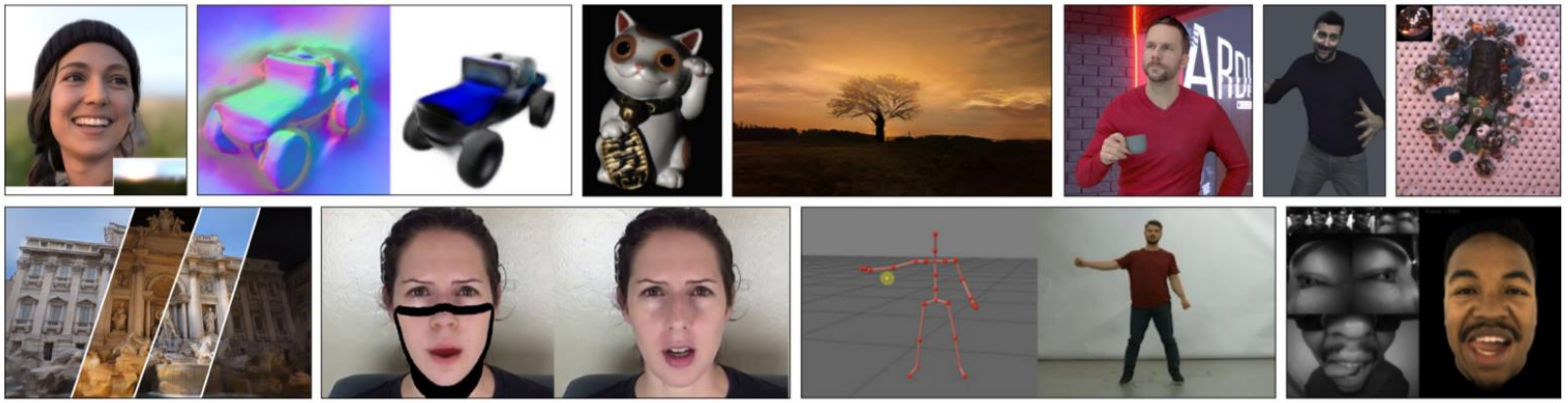


**Figure 1:** *Neural renderings of a large variety of scenes. See Section 6 for more details on the various methods. Images from [SBT* 19, SZW19, XBS* 19, KHM17, GLD* 19, MBPY* 18, XSHR18, MGK* 19, FTZ* 19, LXZ* 19, WSS* 19].*

# Learning to Generate Chairs with Convolutional Neural Networks

Dosovitsky, Springenberg, Brox


- Early work in deep neural rendering (CVPR 2015)

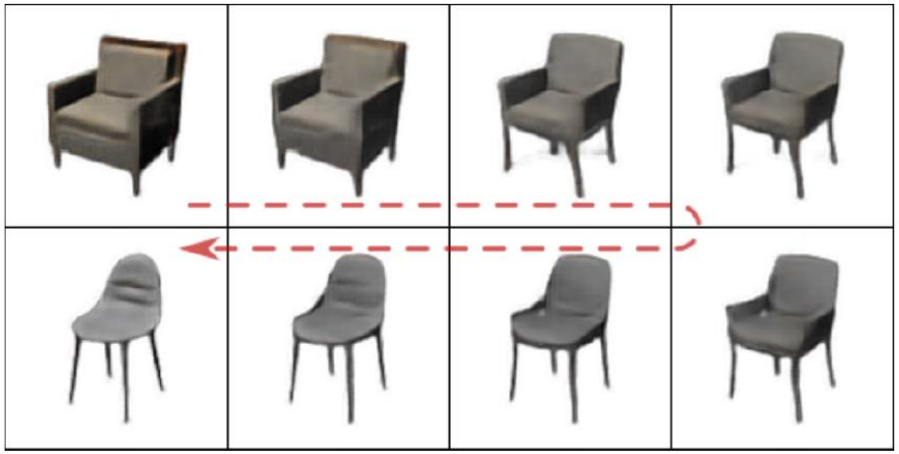


Figure 1. Interpolation between two chair models (original: top left, final: bottom left). The generative convolutional neural network learns the manifold of chairs, allowing it to interpolate between chair styles, producing realistic intermediate styles.

# Learning to Generate Chairs with Convolutional Neural Networks

Dosovitsky, Springenberg, Brox

- Architecture is basically a pair of inverted CNNs
- Trained on 809 chair models from 62 viewpoints (~50,000 training samples)

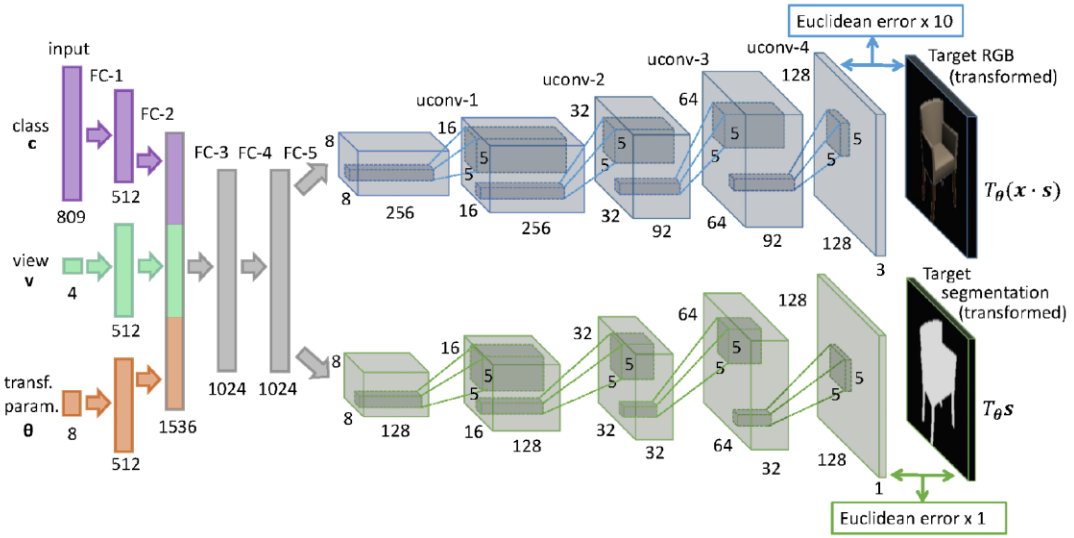



Figure 2. Architecture of the 128 × 128 network. Layer names are shown above: FC - fully connected, uconv - unpooling+convolution.

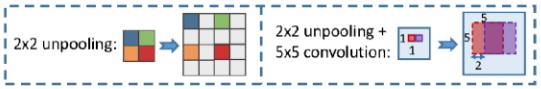



Figure 3. Illustration of unpooling (left) and unpooling+convolution (right) as used in the generative network.

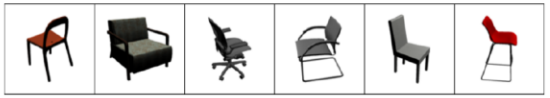


Figure 4. Several representative chair images used for training the network.

# Learning to Generate Chairs with Convolutional Neural Networks

Dosovitsky, Springenberg, Brox

- Analysis:
  - They looked into activating single neurons, single layers, and interpolating input viewing parameters

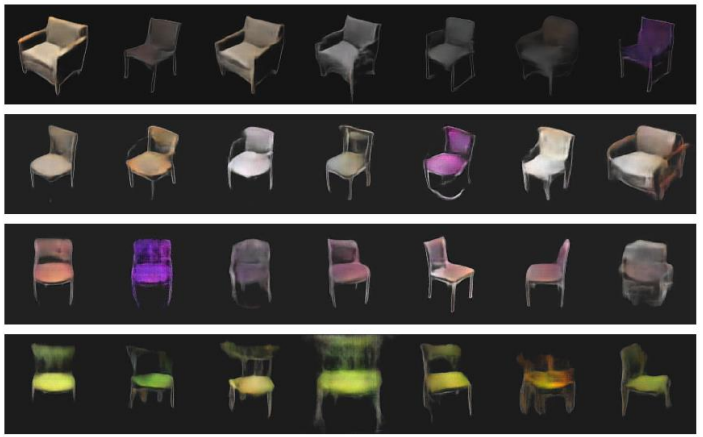


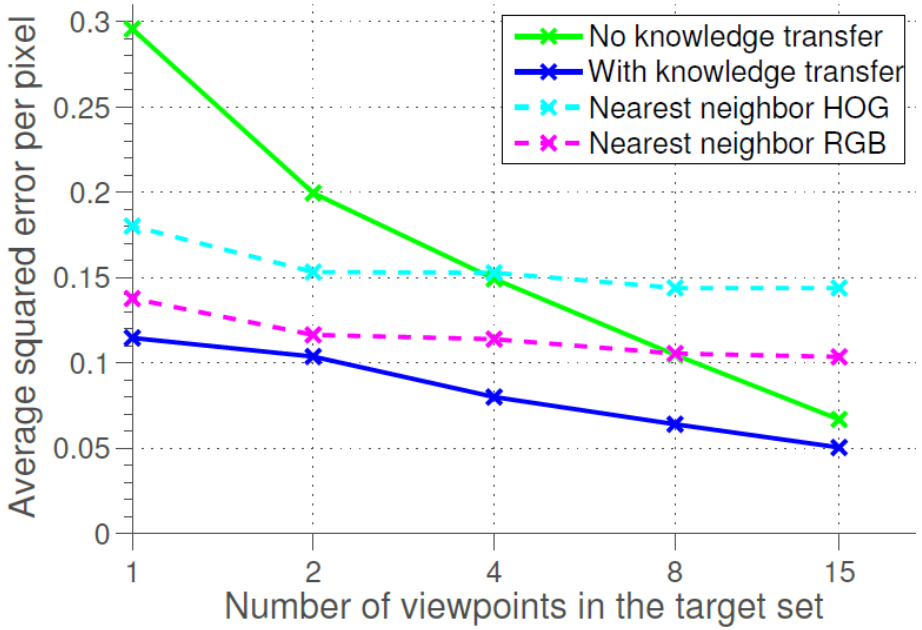
Figure 7. Images generated from single unit activations in feature maps of different fully connected layers of the $128 \times 128$ network. **From top to bottom:** FC-1 and FC-2 of the class stream, FC-3, FC-4.

Figure 8. The effect of increasing the activation of the 'zoom neuron' we found in the layer FC-4 feature map.

# Learning to Generate Chairs with Convolutional Neural Networks

Dosovitsky, Springenberg, Brox


- Analysis:
  - They looked into activating single neurons, single layers, and interpolating input viewing parameters



# Learning to Generate Chairs with Convolutional Neural Networks

Dosovitsky, Springenberg, Brox

- Analysis:
  - They looked into activating single neurons, single layers, and interpolating input viewing parameters

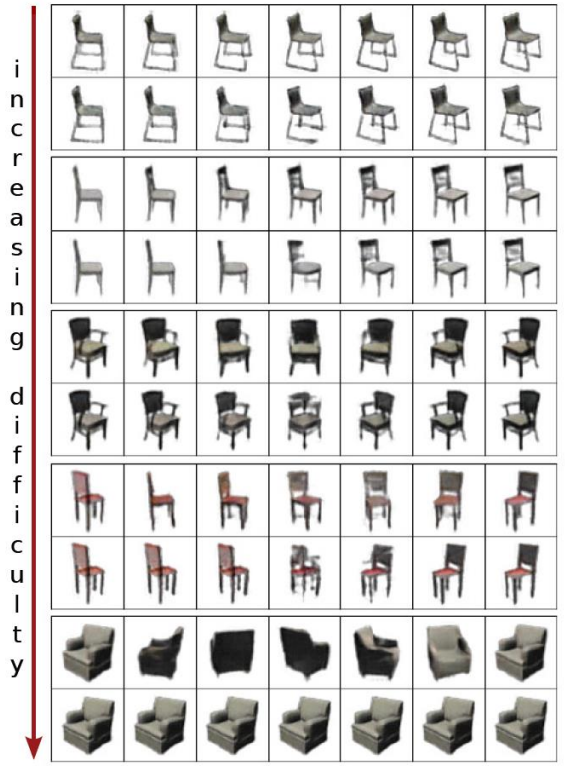



Figure 12. Examples of interpolation between angles. In each pair of rows the top row is with knowledge transfer and the second - without. In each row the leftmost and the rightmost images are the views presented to the network during training, while all intermediate ones are not and hence are results of interpolation. The number of different views per chair available during training is 15, 8, 4, 2, 1 (top-down). Image quality is worse than in other figures because we use the $64 \times 64$ network here.

# Learning to Generate Chairs with Convolutional Neural Networks

Dosovitsky, Springenberg, Brox

- Analysis:
  - They looked into activating single neurons, single layers, and interpolating input viewing parameters

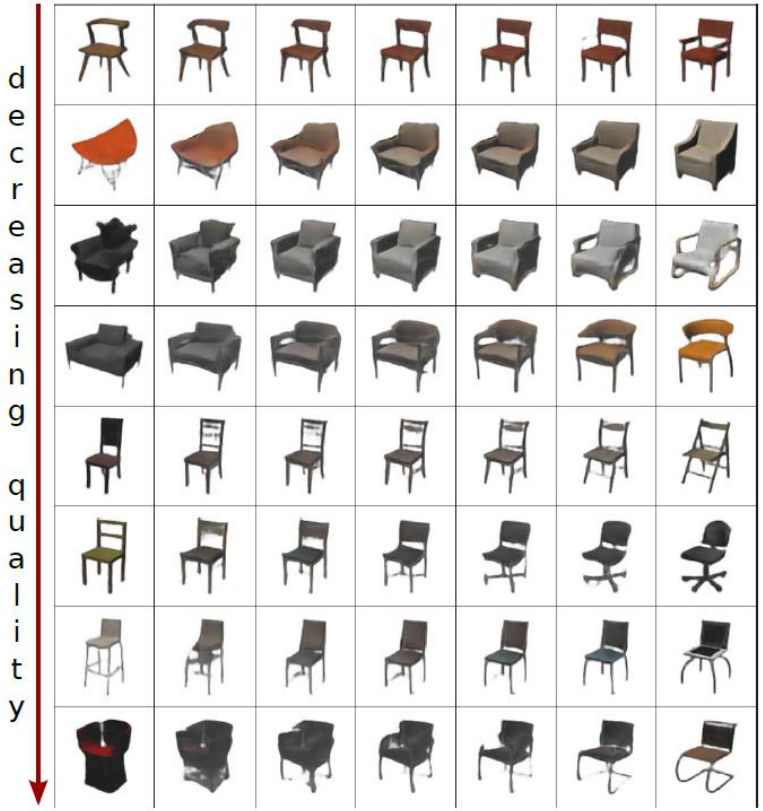



Figure 14. Examples of morphing different chairs, one morphing per row. Leftmost and rightmost chairs in each row are present in the training set, all intermediate ones are generated by the network. Rows are ordered by decreasing subjective quality of the morphing, from top to bottom.

# Neural Rendering


- So how to improve from the shown "baseline"?

- One next wave of methods improved neural-rendering quality by using perceptual similarity metrics…

# Perceptual/Image Similarity Metrics


- L2 loss (=“square of pixel to pixel difference”)
  - Yields overall similarity
- L1 loss (=“absolute value of pixel to pixel difference”)
  - More sensitive to strong changes, thus focuses on details but does not do well with the overall picture
- PSNR (=“signal to noise ratio”)
  - Used lots of compression

# Perceptual/Image Similarity Metrics

- SSIM (Structured Similarity Image Metric)
  - Combine luminance, contrast, and structure

$$I(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \quad C(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$$

$$S(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

$$\mathrm{SSIM}(x,y) = I(x,y)^\alpha C(x,y)^\beta S(x,y)^\gamma$$

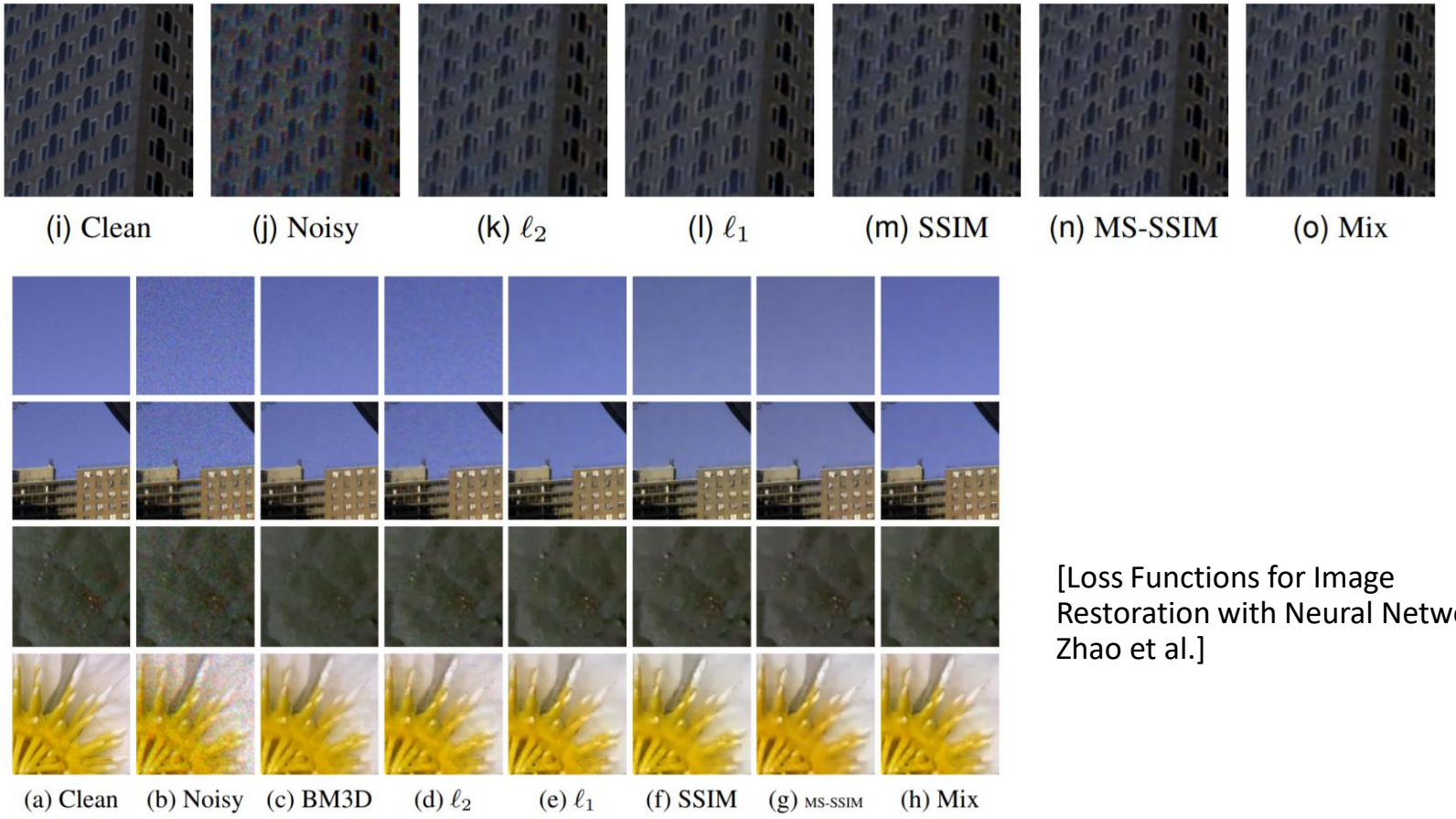



Fig. 5: Results for denosing+demosaicking for different approaches. The noisy patches are obtained by simple bilinear interpolation. Note the splotchy artifacts $\ell_2$ produces in flat regions. Also note the change in colors for SSIM-based losses. The proposed metric, MS-SSIM+$\ell_1$ referred to as Mix, addresses the former issues. Reference images are shown in Figure 4.

[Loss Functions for Image Restoration with Neural Networks, Zhao et al.]

# What else? How to use deep learning?

# Generating Images with Perceptual Similarity Metrics based on Deep Networks

Dosovitskiy, Brox

- Proposed a class of loss functions, called deep perceptual similarity metrics (DeePSiM), to reduce over-smoothed results

- Losses are weighted sums of
  - Feature loss (C)
  - Adversarial loss (G)
  - Pixel-space loss (D)

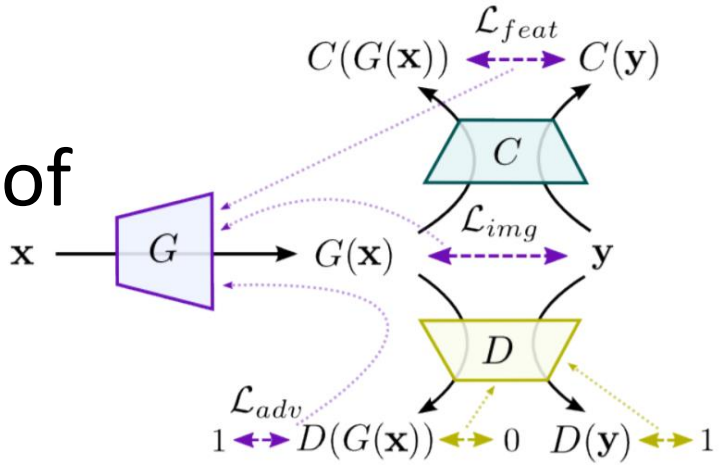



Figure 2: Schematic of our model. Black solid lines denote the forward pass. Dashed lines with arrows on both ends are the losses. Thin dashed lines denote the flow of gradients.

# Generating Images with Perceptual Similarity Metrics based on Deep Networks

Dosovitskiy, Brox

- Feature loss

$$\mathcal{L}_{feat} = \sum_i ||C(G_\theta(\mathbf{x}_i)) - C(\mathbf{y}_i)||_2^2.$$

- Generator loss

$$\mathcal{L}_{discr} = -\sum_i \log(D_\varphi(\mathbf{y}_i)) + \log(1 - D_\varphi(G_\theta(\mathbf{x}_i))), \quad (3)$$

$$\mathcal{L}_{adv} = -\sum_i \log D_\varphi(G_\theta(\mathbf{x}_i)).$$

- Pixel-space loss

$$\mathcal{L}_{img} = \sum_i ||G_\theta(\mathbf{x}_i) - \mathbf{y}_i||_2^2.$$

# Generating Images with Perceptual Similarity Metrics based on Deep Networks

Dosovitskiy, Brox

- Used with for compression with an autoencoder, generative model with a VAE, and inversion of AlexNet

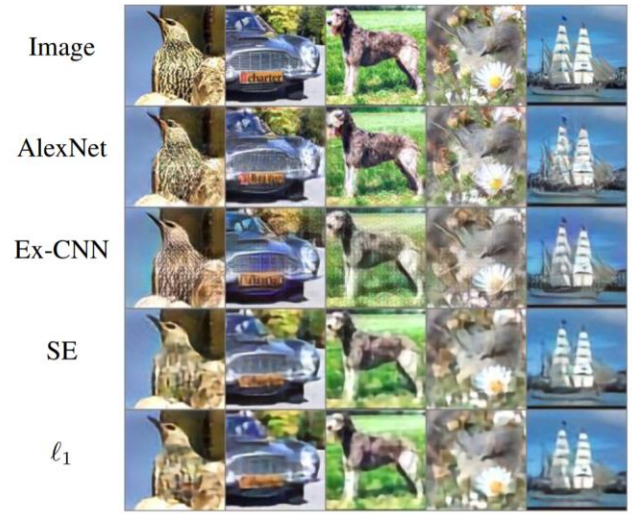



Figure 3: Autoencoder qualitative results. Best viewed on screen.

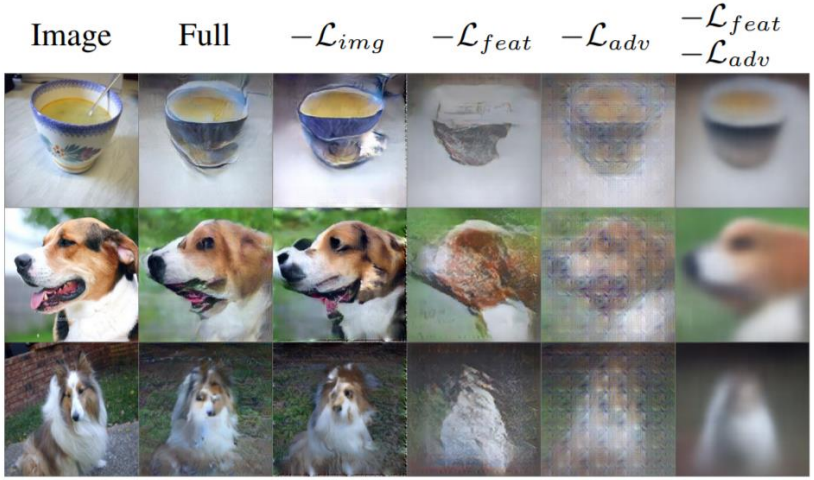



Figure 7: Reconstructions from FC6 with some components of the loss removed.

# Deep CG2Real: Synthetic-to-Real Translation via Image Disentanglement

Bi et al.

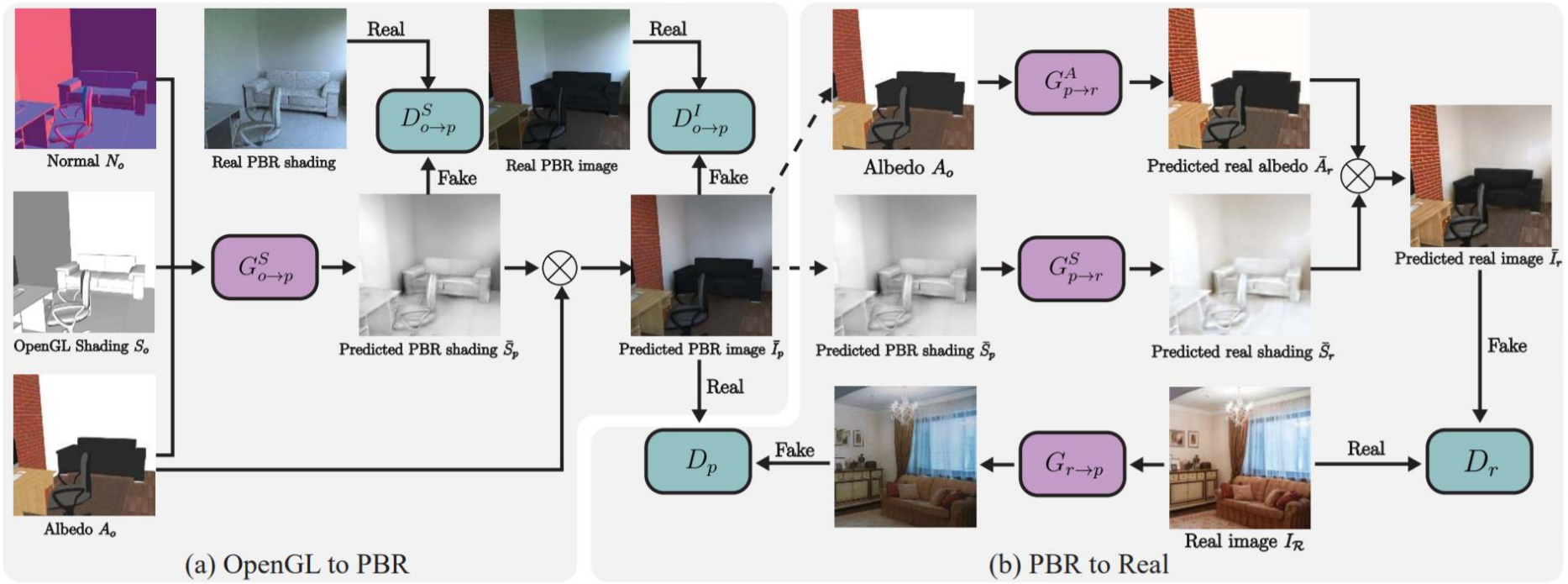



Figure 2: The framework of our two-stage OpenGL to real translations.

# Neural Rerendering in the Wild (CVPR 2019)

- https://www.youtube.com/watch?v=E1crWQn_kmY

# Image-guided Neural Object Rendering


- https://niessnerlab.org/projects/thies2020ignor.html

# Neural Point-Based Graphics


- https://dmitryulyanov.github.io/neural_point_based_graphics
- https://arxiv.org/pdf/1906.08240.pdf

# Deferred Neural Rendering: Image Synthesis using Neural Textures (2019)


- https://www.youtube.com/watch?v=z-pVip6WeyY

# DeepView: View Synthesis with Learned Gradient Descent (CVPR 2019)

- https://www.youtube.com/watch?v=CQ0kdR3c4Ec

# Content Generation


- Texture Synthesis
  - Make/replicate small fragments
- Procedural Modeling
  - Make 3D models
- Neural Rendering
  - Render using neural networks (first wave)
- **Generative Modeling**
  - **Generate from many examples (GANs)…**

# Generative Adversarial Nets

Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozairy, Courville, Bengio


- A two player min-max game to generate data hopefully indistinguishable from real data

# Generative Adversarial Networks (GANs)

[Goodfellow et al. 2014]

$z$ → **Player 1: generator** Scores if discriminator can't distinguish output from real image

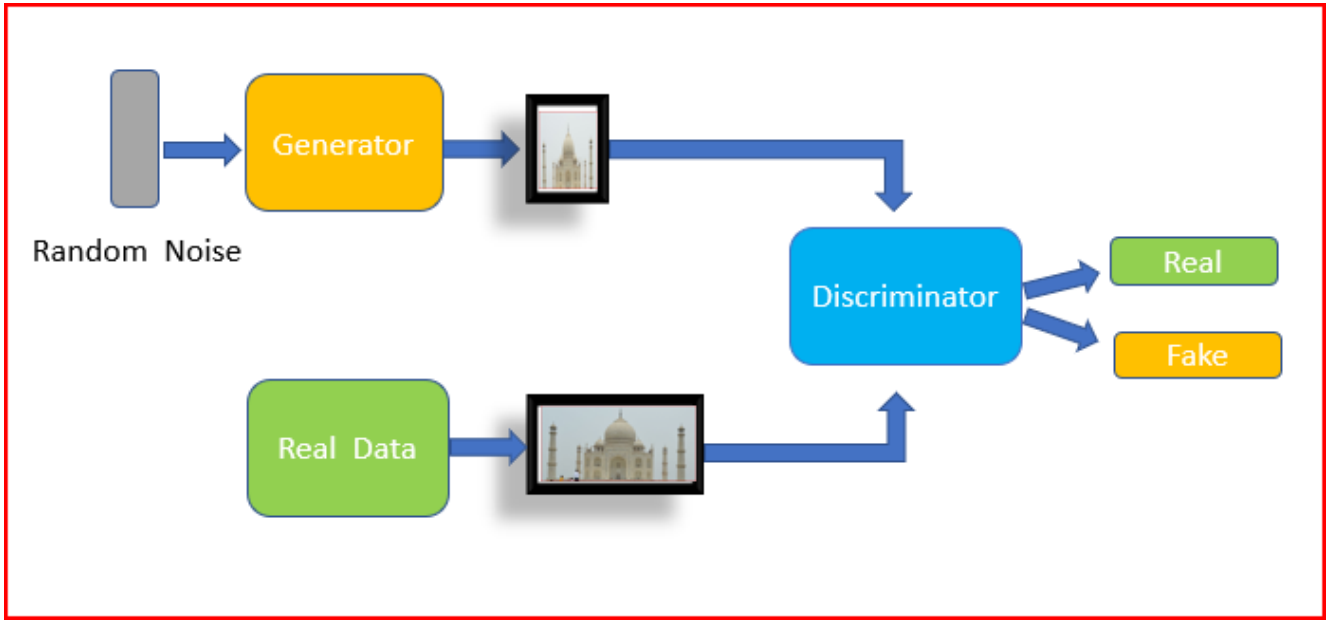
from dataset

**Player 2: discriminator** Scores if it can distinguish between real and fake → real/fake

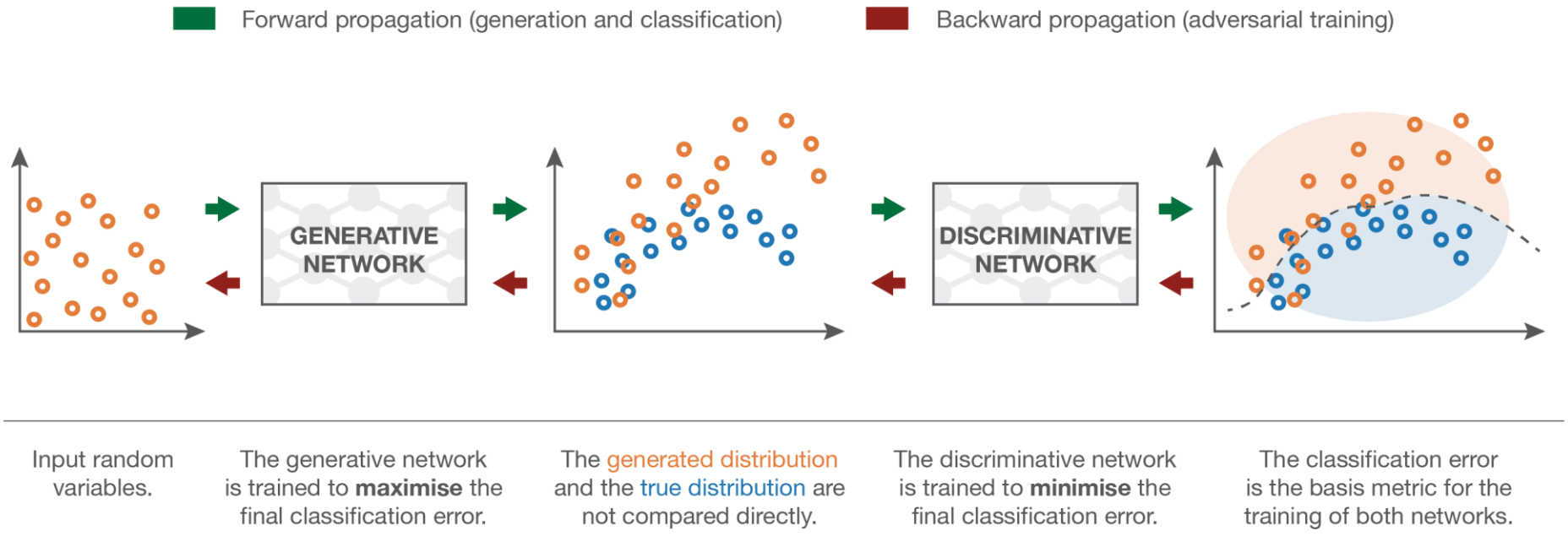
[CreativeAI – SIGGRAPH Course]

Image credit: *A Style-Based Generator Architecture for Generative Adversarial Networks,* Karras et al.

# Generative Adversarial Networks (GANs)



# GAN Information Flow




[CreativeAI – SIGGRAPH Course]

https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

# Examples (2014)


- Example of the Progression in the Capabilities of GANs From 2014 to 2017. Taken from The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation, 2018.

# Artbreeder Demo


- https://www.artbreeder.com/

# Conditional GAN: Pix2Pix

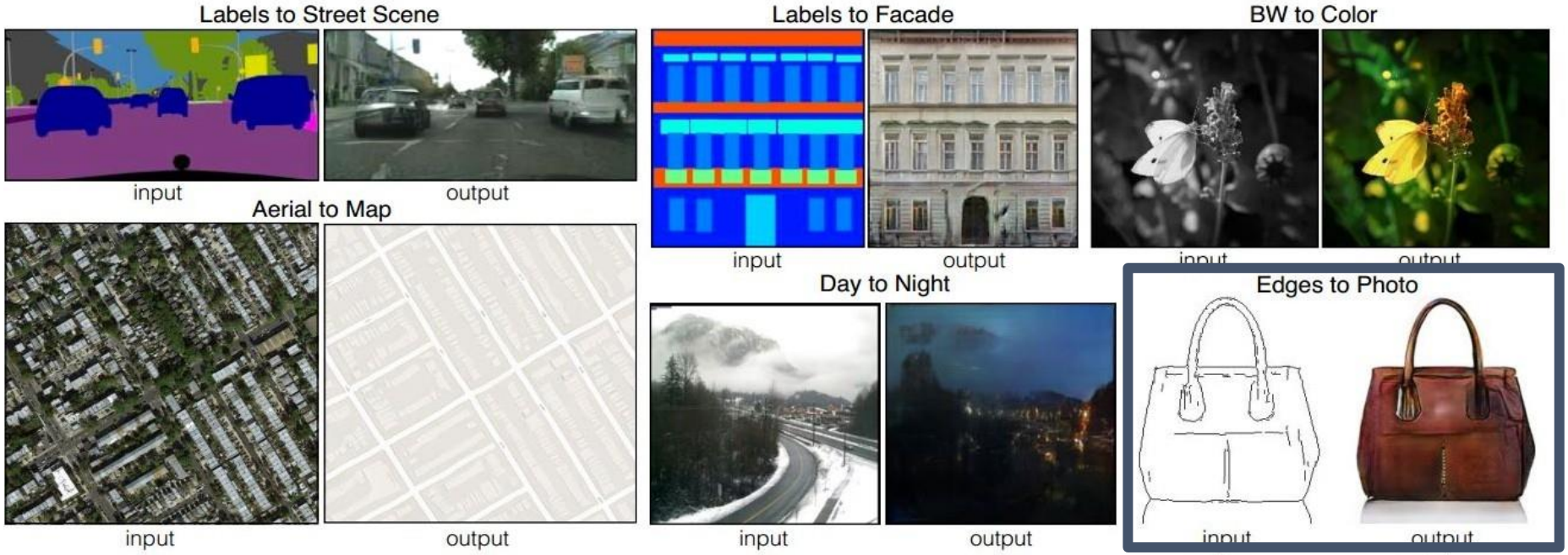



Image-to-image Translation with Conditional Adversarial Nets
Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. CVPR 2017

# Conditional GAN

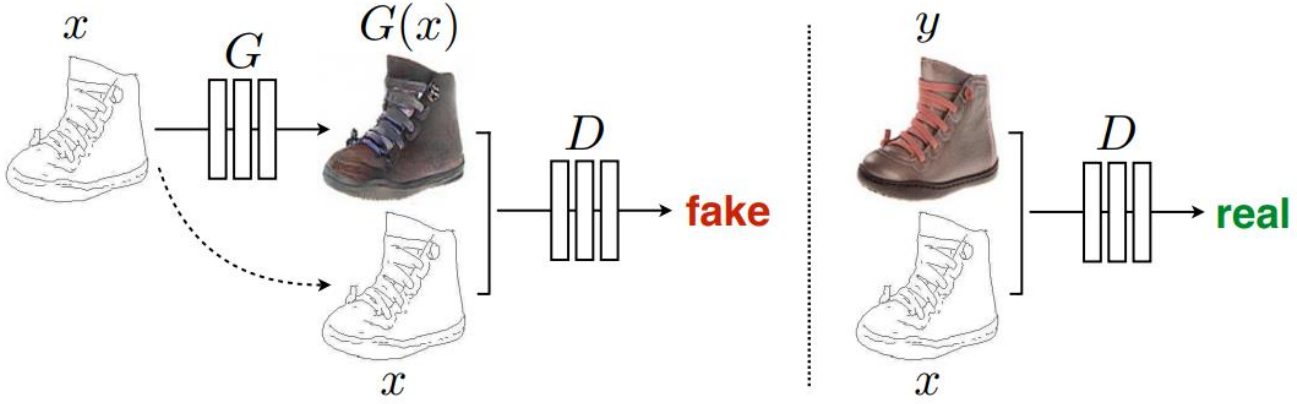
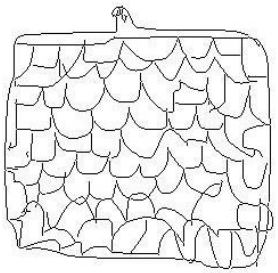
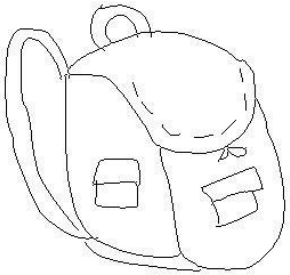
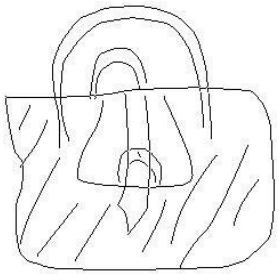
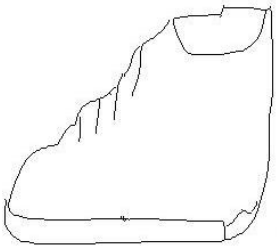
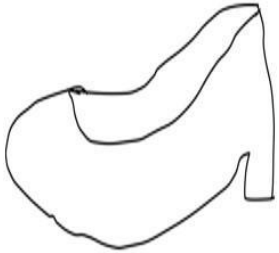



Figure 2: Training a conditional GAN to map edges→photo. The discriminator, $D$, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator, $G$, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

# Edges → Images

# *Sketches* → Images

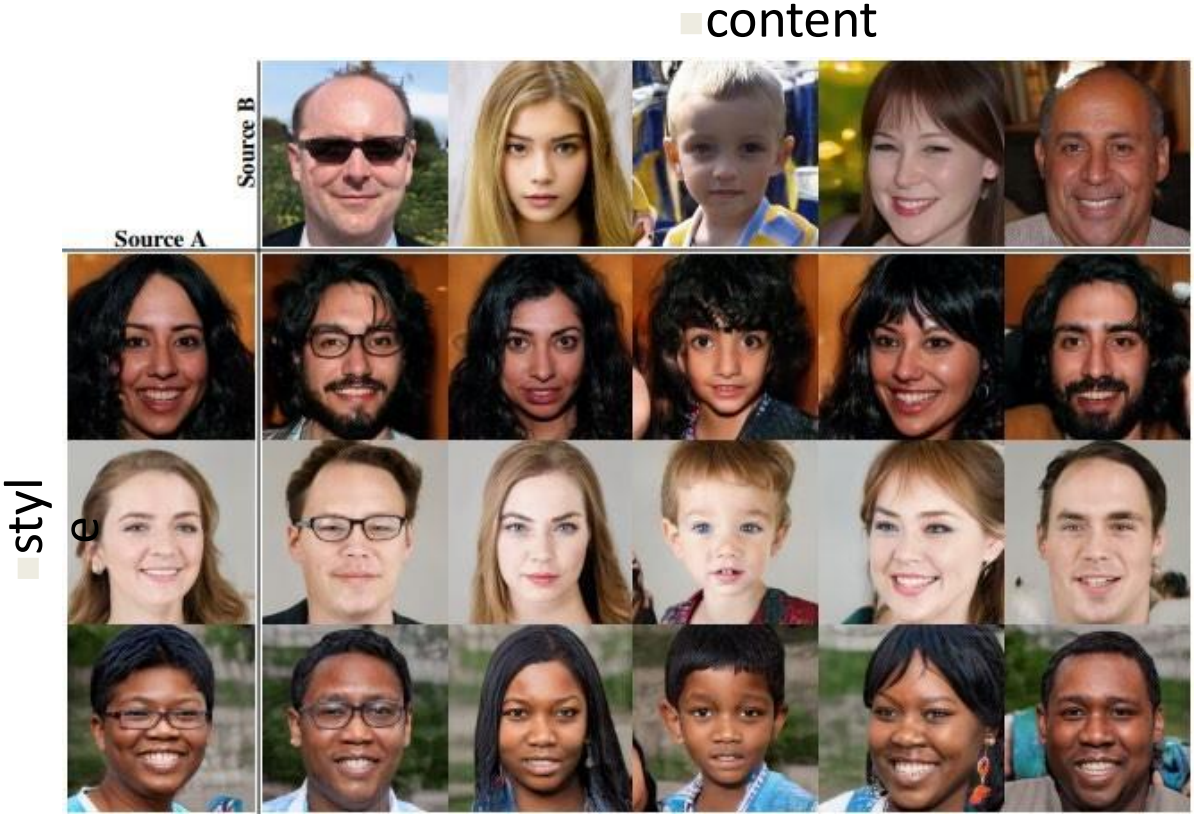
| Input | Output | Input | Output | Input | Output |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Pix2Pix Demo


- https://phillipi.github.io/pix2pix/

# StyleGAN

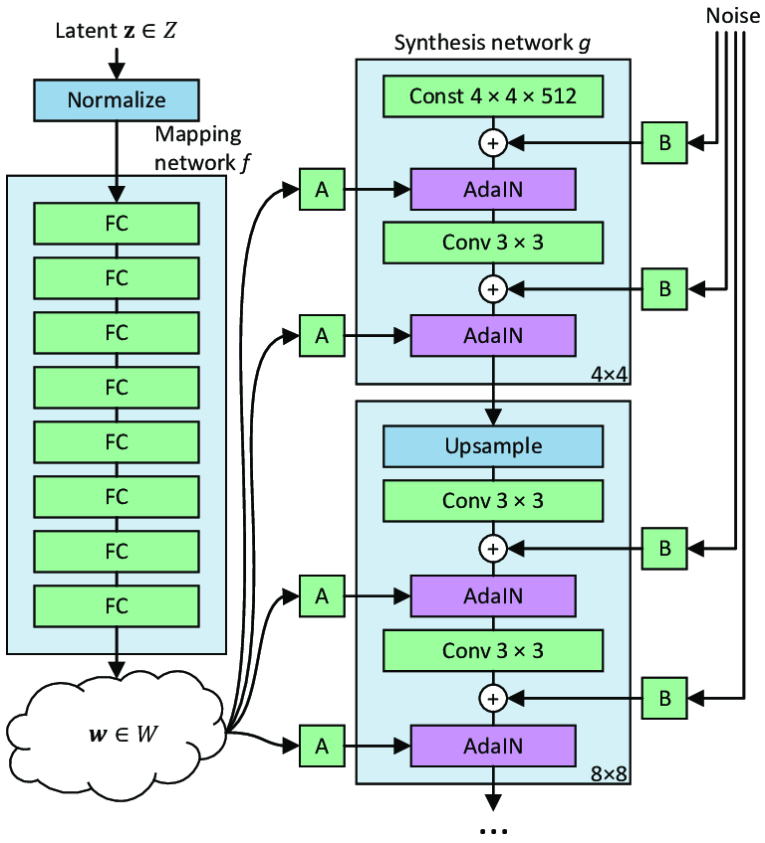
- Additional Tricks:
  - Coarse-to-fine training
  - Transformation of
- to a more complex distr.
  - ...




[CreativeAI – SIGGRAPH Course]

Image credit: *A Style-Based Generator Architecture for Generative Adversarial Networks,* Karras et al.

# StyleGAN


Latent $\mathbf{z} \in Z$
Normalize
Mapping network $f$
FC
FC
FC
FC
FC
FC
FC
FC
$\boldsymbol{w} \in W$
Synthesis network $g$
Noise
Const 4 × 4 × 512
B
AdaIN
Conv 3 × 3
B
A
AdaIN
4×4
Upsample
Conv 3 × 3
B
A
AdaIN
Conv 3 × 3
B
A
AdaIN
8×8
...

# StyleGAN Demo

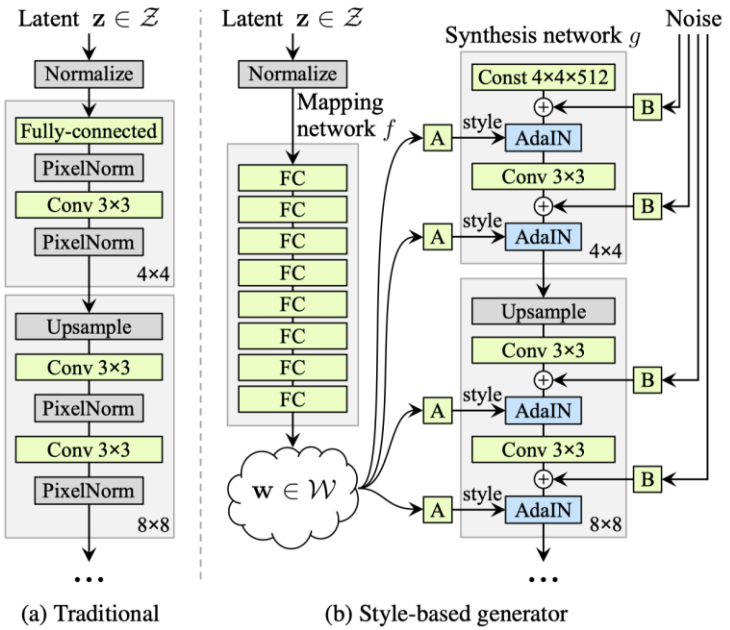

- https://thispersondoesnotexist.com/

# StyleGANs

- Extension to the GAN architecture with proposed large changes including
  - use of a mapping network to map points in latent space to an intermediate latent space
  - use of the intermediate latent space to control style at each point in the generator model
  - introduction to noise as a source of variation at each point in the generator model
- The discriminator or the loss function is not modified in the original implementation of StyleGan so the same discussion about GAN loss functions, regularization, and hyperparameters can be applied here as well

# Style-based generator

- Typically latent code for a generator is given as a Input layer (first layer of feed-forward network)
- Instead we switch to a learned constant
  - Latent code: $z \in Z$
  - Non-linear mapping: $f: Z \rightarrow W$ produces $w \in W$ – implemented using a 8-layer MLP
  - Dimensionality of both spaces to 512
- Learned affine transformations specialize $w$ to styles $y = (y_s, y_b)$ which is used to control adaptive instance normalization (AdaIN) operations after each convolution layer of the synthesis network $g$
  - $AdaIN(x_i, y) = y_{s,i} \frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$
  - AdaIN operation takes a feature map $x_i$, which is normalized separately, and then scaled and biased using corresponding scalar components from style $y$
  - That means dimensionality of $y$ is twice the number of feature maps on that layer



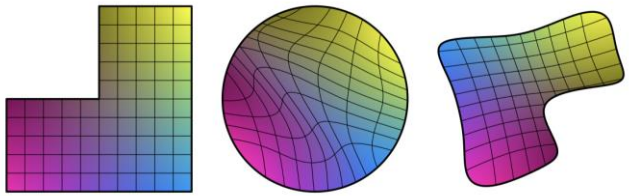

[Karras et. al., 2019]

# Style Mixing

- *Mixing regularization:* encourages style to localize.
  - given percentage of images are generated using two random latent codes instead of one during training
  - generating such an image simply requires from one latent code to another
- Two latent codes $z_1, z_2$ runs though the mapping network and have corresponding $w_1, w_2$ control the style so that $w_1$ applies before the crossover point and $w_2$ after it.
- This technique prevents the network from assuming that adjacent styles are correlated

# Stochastic Variation

- In human portraits, the following can be considered stochastic
  - Exact placement of hairs
  - Stubble
  - Freckles
  - Skin pores
- Noise tends to only affect the stochastic aspects of the generation process.



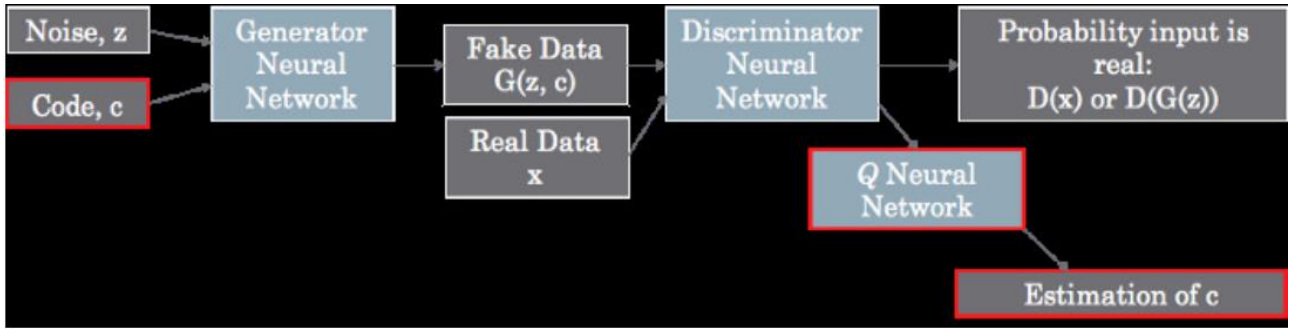
(a) Distribution of features in training set (b) Mapping from $\mathcal{Z}$ to features (c) Mapping from $\mathcal{W}$ to features

- An example training set where some combination (e.g., long haired males) is missing.
- This forces the mapping from Z to image features to become curved so that the forbidden combination disappears in Z to prevent the sampling of invalid combinations.
- The learned mapping from Z to W is able to “undo” much of the warping.

# InfoGAN

Chen et al. 2016

- Unsupervised Disentanglement Learning
- Extends GAN by
  - Split the incoming noise vector z into two parts - noise and code. The goal is to learn meaningful codes for the dataset.
  - In addition to the discriminator, it adds another prediction head to the network that tries to predict the code from the generated sample. The loss is a combination of the normal GAN loss and the prediction loss.
  - This new loss term can be interpreted as a lower bound on the mutual information between the generated samples and the code.
- The output of the mapping network is called *conditional styles.* This will modulate each block in the synthesis network using AdaIN.



# CycleGAN

by Zhu et al. 2017

- Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

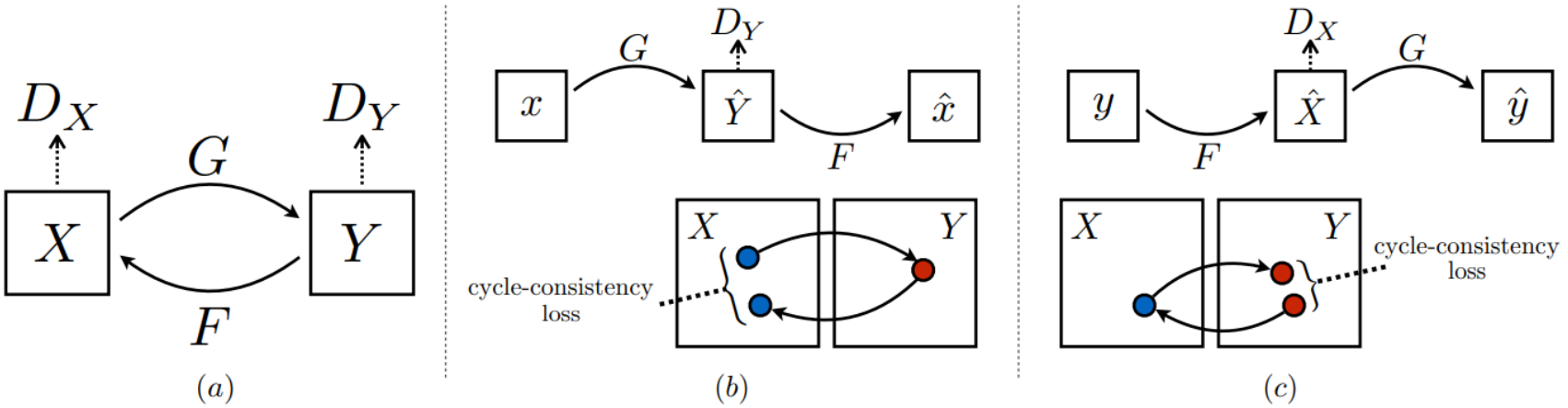



Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators $D_Y$ and $D_X$. $D_Y$ encourages $G$ to translate $X$ into outputs indistinguishable from domain $Y$, and vice versa for $D_X$ and $F$. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

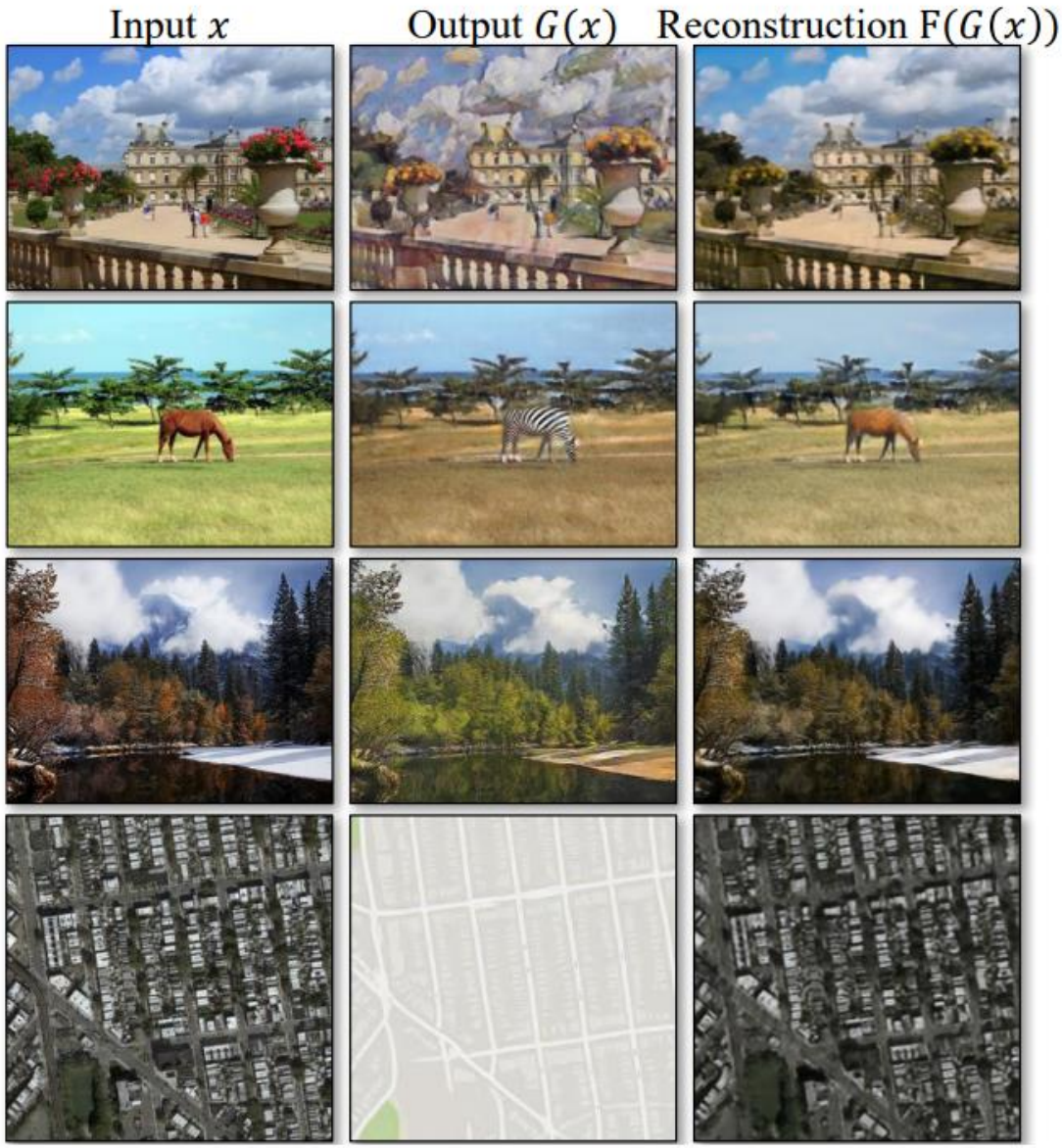


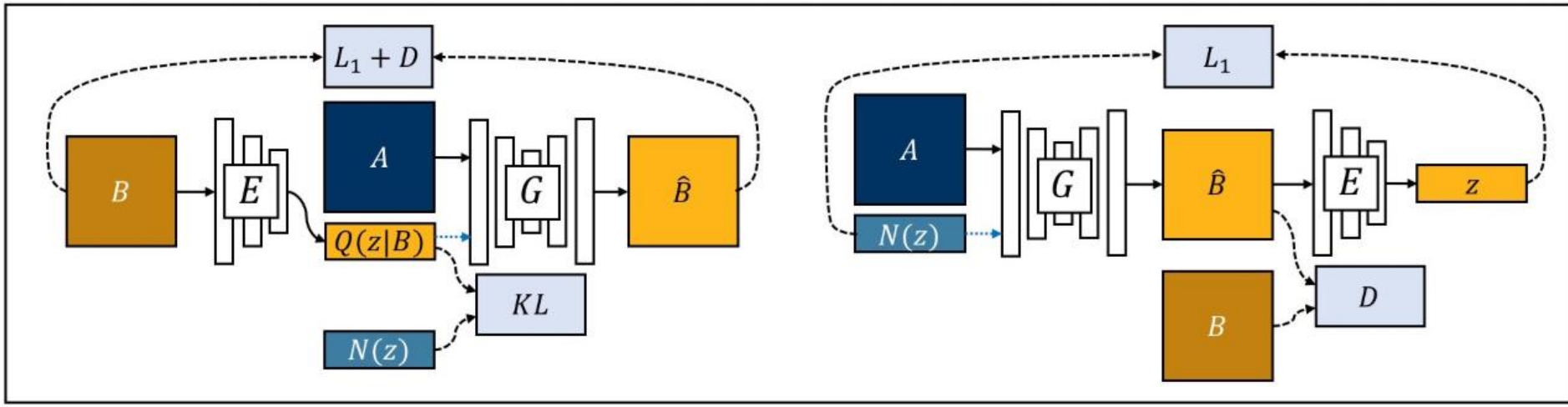

Figure 4: The input images $x$, output images $G(x)$ and the reconstructed images $F(G(x))$ from various experiments. From top to bottom: photo↔Cezanne, horses↔zebras, winter→summer Yosemite, aerial photos↔Google maps.


Input
Monet
Van Gogh
Cezanne
Ukiyo-e

# BiCycleGAN

Zhu et al. 2017

- Toward Multimodal Image-to-Image Translation



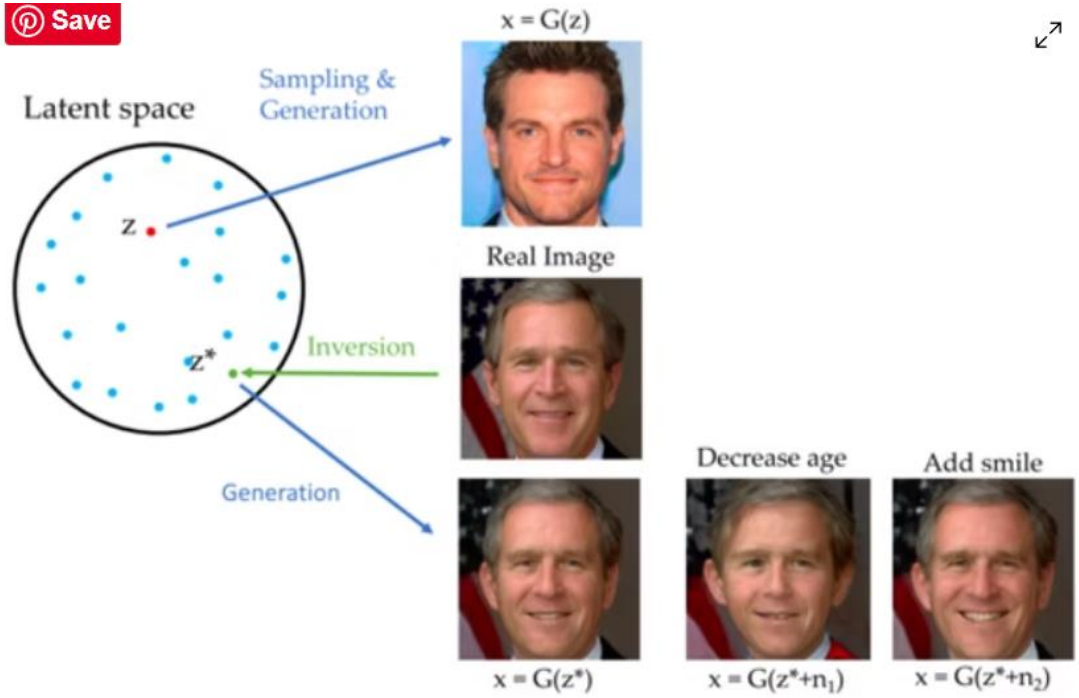
# BiCycleGAN

Zhu et al. 2017


- Video: https://junyanz.github.io/BicycleGAN/

# GAN Inversion

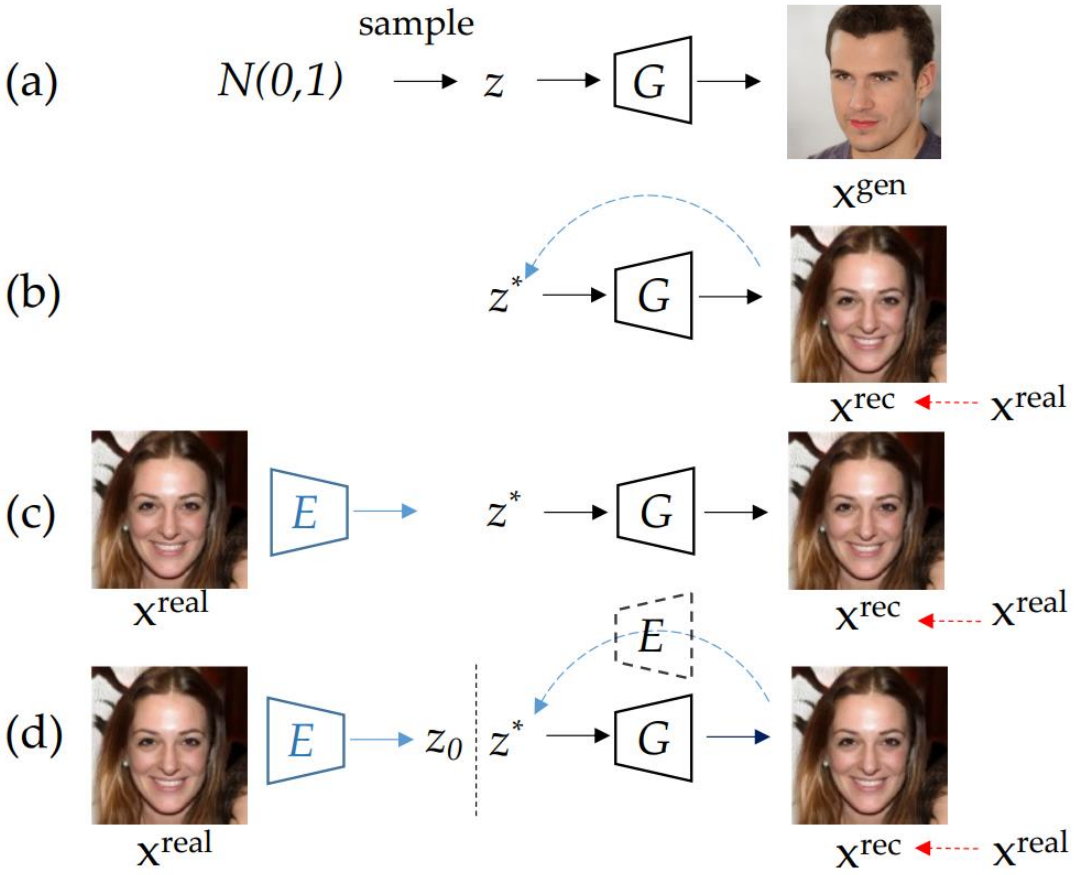
- Aims to invert an image into the latent space to obtain its latent code; then it could be re-generated in a different way




Survey:

Xia et al. 2022 (TPAMI)

# GAN Inversion




Forward GAN

Inversion by optimization

Initialize with a trained encoder; then inversion by optimization

Like above but add a trained regularizer as well

https://github.com/weihaox/GAN-Inversion