Texture Synthesis, Video Textures

BIL721: Computational Photography  Aykut Erdem  
Spring 2015, Lecture 4  Hacettepe University  
Computer Vision Lab (HUCVL)
Today

• Texture synthesis
• Image analogies
• Video textures
Today

- Texture synthesis
- Image analogies
- Video textures
Texture

• What is texture?
  – Easy to recognize, hard to define
  – Deterministic/regular textures (“thing-like”)
  – Stochastic textures (“stuff-like”)

• Tasks
  – Discrimination / Segmentation
  – Classification
  – Texture synthesis
  – Shape from texture
  – Texture transfer
  – Video textures
Modeling Texture

• What is texture?
  – An image obeying some statistical properties
  – Similar structures ("textons") repeated over and over again according to some placement rule
  – Must separate what repeats and what stays the same
  – Often has some degree of randomness
  – Often modeled as repeated trials of a random process
Texture Synthesis

- **Goal of Texture Synthesis**: create new samples of a given texture
- Many applications: virtual environments, hole-filling, texturing surfaces
The Challenge

- Need to model the whole spectrum: from repeated to stochastic texture
Method 1: Copy Block(s) of Pixels

Visual artifacts at block boundaries!

pattern repeated
One idea: Build Probability Distributions

Basic idea
1. Compute statistics of input texture (e.g., histogram of edge filter responses)
2. Generate a new texture that keeps those same statistics

One idea: Build Probability Distributions

But it (usually) doesn’t work

- Probability distributions are hard to model well
Statistical Modeling of Texture

- Assume stochastic model of texture (Markov Random Field)
- Stationarity: the stochastic model is the same regardless of position

stationary texture

non-stationary texture
Markov Chain

- Markov Chain
  - a sequence of random variables $x_1, x_2, \ldots, x_n$
  - $x_t$ is the state of the model at time $t$

\[
\begin{array}{cccc}
  x_1 & \rightarrow & x_2 & \rightarrow & x_3 & \rightarrow & x_4 & \rightarrow & x_5 \\
\end{array}
\]

- Markov assumption: Each state is dependent only on the previous one
  - dependency given by a conditional probability:
    \[
p(x_t | x_{t-1})
    \]
- The above is actually a first-order Markov chain
- An $n^{th}$-order Markov chain:
  \[
p(x_t | x_{t-1}, \ldots, x_{t-N})
  \]
Markov Random Field

A Markov Random Field (MRF)

- Generalization of Markov chains to two or more dimensions

**First-order MRF:**

- Probability that pixel $X$ takes a certain value given the values of neighbors $A$, $B$, $C$, and $D$:

$$P(X|A, B, C, D)$$

- Higher-order MRFs have larger neighborhoods
Statistical Modeling of Texture

• Assume stochastic model of texture (Markov Random Field)

• Stationarity: the stochastic model is the same regardless of position

• Markov (i.e., local) property:
  \[ p(\text{pixel} \mid \text{rest of image}) = p(\text{pixel} \mid \text{neighborhood}) \]
Motivation from Language

• Shannon (1948) proposed a way to generate English-looking text using \( n \)-grams
  – Assume a Markov model
  – Use a large text corpus to compute probability distributions of each letter given \( N-1 \) previous letters
  – Starting from a seed, repeatedly sample the conditional probabilities to generate new letters
  – Can use whole words instead of letters too
• 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!
  – Now let’s try this in 2D using pixels
Efros & Leung Algorithm

- Assuming Markov property, compute $P(p|N(p))$
  - Building explicit probability tables infeasible
  - Instead, we search the input image for all similar neighborhoods — that’s our pdf for $p$
  - To sample from this pdf, just pick one match at random
Details

• How to match patches?
  – Gaussian-weighted SSD (more emphasis on nearby pixels)

• What order to fill in new pixels?
  – “Onion skin” order: pixels with most neighbors are synthesized first
  – To synthesize from scratch, start with a randomly selected small patch from the source texture

• How big should the patches be?
Finding matches

- Sum of squared differences (SSD)

\[ \| A - B \|^2 \]
Finding matches

- Sum of squared differences (SSD)
  - Gaussian-weighted to make sure closer neighbors are in better agreement

\[ \| \ast (\text{image} - \text{image}) \|^2 \]
Neighborhood Window

input
Varying Window Size

Increasing window size
Texture synthesis algorithm

• While image not filled
  1. Get unfilled pixels with filled neighbors, sorted by number of filled neighbors
  2. For each pixel, get top $N$ matches based on visible neighbors
     - Patch Distance: Gaussian-weighted SSD
  3. Randomly select one of the matches and copy pixel from it
Synthesis Results

french canvas

rafi weave

slide by Alyosha Efros
More Results

- white bread
- brick wall

slide by Alyosha Efros
Homage to Shannon

...
Summary

• The Efros & Leung algorithm
  – Very simple
  – Surprisingly good results
  – Synthesis is easier than analysis!
  – …but very slow

• How can you improve it? Any ideas?
Image Quilting [Efros & Freeman]

- **Observation:** neighbor pixels are highly correlated

**Idea:** unit of synthesis = block

- Exactly the same but now we want $P(B|N(B))$
- Much faster: synthesize all pixels in a block at once
- Not the same as multi-scale!
Input texture

Random placement of blocks

Neighboring blocks constrained by overlap

Minimal error boundary cut
Minimal error boundary

overlapping blocks

vertical boundary

\[
\begin{bmatrix}
\text{overlapping blocks} \\
\end{bmatrix}^2 = \text{min. error boundary}
\]
Texture synthesis with graph cuts

- **Seam Boundaries**
- **Sample Texture**

**Synthesized Texture (Initialization)**

**Steps:**
1. **Step 2**
2. **Step 3**
3. **Step 4**
4. **Step 5**

**Seam Costs**

**Synthesized Texture**

(After 5 steps of Refinement)

[Kwatra et al. 2005]

- **Entire patch matching:**
  - We use the cost of existing seams to quantify the error in output. Once we pick such an error-region, we force the patch selection process to attempt to place a new patch over a region where the input and output images respectively, and overlap the error-region. When the texture is being initialized, we first find a region in the current texture that needs a lot of improvement. We use the cost of existing seams to quantify the error in output, we normalize the sum-of-squared-differences (SSD) cost with the area of the overlapping region. We compute this cost for output images. To account for partial overlaps between the input and the output, we normalize the sum-of-squared-differences (SSD) cost with the area of the overlapping region. We compute this cost for output, we normalize the sum-of-squared-differences (SSD) cost with the area of the overlapping region.

**3.2 Surrounded Regions**

Now we describe several algorithms for picking candidate patches.

- **(1)**
- **(2)**
- **(3)**

These selection methods are used in some detail. The same three placement algorithms are used in some detail. The same three placement algorithms are used in some detail. The same three placement algorithms are used in some detail.
There is a potential for visible seams along the border between old and new patches in a region where multiple patches already meet. These seams can be problematic as they may disrupt the visual coherence of the final texture. To address this issue, we can use graph cuts to find the optimal placement of seams.

The example shown in Figure 2 demonstrates how graph cuts can be used to find the minimum cut between two patches. In this case, the red line represents the minimum cut, which separates the old patch from the new patch. The path determined by this cut is from the top to the bottom of the region, indicating the direction of the seam.

In the example, pixels 1, 2, and 3 are constrained to come from the old patch, while pixels 4, 5, and 6 are constrained to come from the new patch. This constraint ensures that the seam is placed in a way that minimizes the visual disruption.

Accounting for existing seams is crucial as well. In Figure 3, we see an old seam between pixels 1 and 3, which we wish to lay down in the output texture. To incorporate these existing seams, we can introduce seam nodes into the graph and connect them with arcs.

For each existing seam, we introduce a seam node and connect it to the old patch node with an arc. This new seam cost is added to the graph cut problem. If no arc is cut, the seam is removed, and the seam cost is not counted in the final cost. If one of the arcs between a seam node and another node is cut, the seam is removed.

In the example, there is an old seam between pixels 1 and 3. This old seam affects the seam placement decisions. We incorporate the old seam cost into the new graph cut problem. The graph formulation with old seams is shown in Figure 3(right).

The graph formulation with old seams shows how we can incorporate existing seams into the graph cut problem. This allows us to lay down new patches in a way that respects existing seams, thereby reducing the likelihood of visible discontinuities.

In summary, accounting for existing seams is essential to achieve a seamless texture synthesis. By considering existing seams and incorporating their costs into the graph cut problem, we can ensure that the new patches fit smoothly into the existing texture, resulting in a more visually appealing output.
slide by Alyosha Efros
input image

Portilla & Simoncelli

Xu, Guo & Shum

Wei & Levoy

Efros & Freeman
input image

Portilla & Simoncelli

Wei & Levoy

Xu, Guo & Shum

Efros & Freeman
Failures
(Chernobyl Harvest)

excessive repetition

mismatched or distorted boundaries
Self Tuning Texture Optimization

Image Quilting / Our Result
Large-scale Structures

Resynthesizer / Our Result
Near-regular Structures

Graphcut Textures / Our Result
Repetitions

Image Melding / Our Result
Smoothing

http://w-x.ch/publications/self-tuning-texture-optimization/
  supplementary/results/

Self Tuning Texture Optimization

Political Texture Synthesis!

Bush campaign digitally altered TV ad

President Bush’s campaign acknowledged Thursday that it had digitally altered a photo that appeared in a national cable television commercial. In the photo, a handful of soldiers were multiplied many times.

This section shows a sampling of the duplication of soldiers.

Original photograph
Fill Order

• In what order should we fill the pixels?
Fill Order

- In what order should we fill the pixels?
  - choose pixels that have more neighbors filled
  - choose pixels that are continuations of lines/curves/edges

Exemplar-based Inpainting demo

Application: Texture Transfer

- Try to explain one object with bits and pieces of another object:
Texture Transfer

Constraint

Texture sample
Texture Transfer

- Take the texture from one image and “paint” it onto another object

Same as texture synthesis, except an additional constraint:
- Consistency of texture
- Patches from texture should correspond to patches from constraint in some way. Typical example: blur luminance, use SSD for distance
Texture Transfer

source texture

target image

correspondence maps

texture transfer result
Texture Transfer

slide by Alyosha Efros
Today

- Texture synthesis
- Image analogies
- Video textures
Image Analogies
Image Analogies

Unfiltered source

Filtered source

Unfiltered target

Filtered target
Image Analogies

• Define a similarity between A and B
• For each patch in B:
  – Find a matching patch in A, whose corresponding A’ also fits in well with existing patches in B’
  – Copy the patch in A’ to B’
• Algorithm is done iteratively, coarse-to-fine
The Approach

Unfiltered source

Filtered source

Unfiltered target

Filtered target
Image Analogy Algorithm

function CREATEIMAGEANALOGY(A, A', B):
    Compute Gaussian pyramids for A, A', and B
    Compute features for A, A', and B
    Initialize the search structures (e.g., for ANN)
    for each level \( \ell \), from coarsest to finest, do:
        for each pixel \( q \in B'_\ell \), in scan-line order, do:
            \( p \leftarrow \text{BESTMATCH}(A, A', B, B', s, \ell, q) \)
            \( B'_\ell(q) \leftarrow A'_\ell(p) \)
            \( s_\ell(q) \leftarrow p \)
    return \( B'_L \)
The Approach

Unfiltered source

Filtered source

Unfiltered target

Filtered target

slide by Steve Seitz
The Approach

**function** CREATEIMAGEANALOGY\((A, A', B)\):

- Compute Gaussian pyramids for \(A, A'\), and \(B\)
- Compute features for \(A, A'\), and \(B\)
- Initialize the search structures (e.g., for ANN)

**for** each level \(\ell\), from coarsest to finest, **do**:

- **for** each pixel \(q \in B'_\ell\), in scan-line order, **do**:
  - \(p \leftarrow \text{BESTMATCH}(A, A', B, B', s, \ell, q)\)
  - \(B'_\ell(q) \leftarrow A'_\ell(p)\)
  - \(s_\ell(q) \leftarrow p\)

**return** \(B'_L\)

**function** BESTMATCH\((A, A', B, B', s, \ell, q)\):

- \(p_{\text{app}} \leftarrow \text{BESTAPPROXIMATEMATCH}(A, A', B, B', \ell, q)\)
- \(p_{\text{coh}} \leftarrow \text{BESTCOHESIEMATCH}(A, A', B, B', s, \ell, q)\)
- \(d_{\text{app}} \leftarrow \| F_\ell(p_{\text{app}}) - F_\ell(q) \|^2\)
- \(d_{\text{coh}} \leftarrow \| F_\ell(p_{\text{coh}}) - F_\ell(q) \|^2\)
- **if** \(d_{\text{coh}} \leq d_{\text{app}} (1 + 2^{\ell-L} \kappa)\) **then**
  - **return** \(p_{\text{coh}}\)
- **else**
  - **return** \(p_{\text{app}}\)

---

slide by Steve Seitz
Implementation Details

- Use approximate nearest neighbor search and Ashikhmin’s coherence search heuristic
- Use feature vectors instead of pixel values
  - Feature vector can consist of RGB values plus additional “channels” such as luminance, outputs of derivative filters
- Luminance remapping to align color histograms of source and target images
Blur Filter

Unfiltered source \((A)\)  
Filtered source \((A')\)

Unfiltered target \((B)\)  
Filtered target \((B')\)
Edge Filter

Unfiltered source (A)  Filtered source (A')

Unfiltered target (B)  Filtered target (B')
Artistic Filters

A

A’

B

B’
Artistic Filters

A

A'

B

B'
Colorization

Unfiltered source (A)  Filtered source (A')

Unfiltered target (B)  Filtered target (B')
Texture-by-numbers
Super-resolution
Super-resolution (result!)

B

B'
Super-resolution

Figure 5

Super-resolution. For each example, the training pairs (above) contain low- and high-resolution versions of a portion of an image. This training data was obtained from the MIT VisTex web page, Copyright 1995 MIT. All rights reserved.

Improved texture synthesis. The textures synthesized with Wei and Levoy's algorithm [49] give blurry results because the perceptual similarity. Ashikhmin's algorithm [2] gives high-quality coherent patches, but creates horizontal edges when patches reach the end of the weave. Our algorithm combines the advantages of these two previous methods. We used a 

Figure 6

Unfiltered target (A), filtered target (B), input Wei-Levoy (W), and Ashikhmin Ours (A' are perpendicular to the coordinate axes, with the exception of horizontal edges when patches reach the end of the weave. Additionally, Ashikhmin's algorithm does not capture appearance at multiple scales, such as in the upper right corner of the flower texture.
Super-resolution result

Figure 5
Improved texture synthesis. The textures synthesized with Wei and Levoy’s algorithm [49] give blurry results because the $L^2$ norm is a poor measure of perceptual similarity. Ashikhmin’s algorithm [2] gives high-quality coherent patches, but creates horizontal edges when patches reach the end of the source image, such as in the upper right corner of the flower texture. Additionally, Ashikhmin’s algorithm does not capture appearance at multiple scales, such as the regular pattern of the weave. Our algorithm combines the advantages of these two previous methods. We used $\kappa = 5$ for both textures in this figure. (The input textures were obtained from the MIT VisTex web page, Copyright © 1995 MIT. All rights reserved).

Figure 6
Super-resolution. For each example, the training pairs (above) contain low- and high-resolution versions of a portion of an image. This training data is used to specify a “super-resolution” filter that is applied to a blurred version of the full image (below, left) to recover an approximation to the higher-resolution original (below, right). (Maple trees image courtesy Philip Greenspun, http://philip.greenspun.com).
Today

• Texture synthesis
• Image analogies
• Video textures
Still photos
Video clips
Video textures
Problem statement

video clip

video texture
Video Textures Approach

• How do we find good transitions?
Finding good transitions

- Compute $L_2$ distance $D_{i, j}$ between all frames

Similar frames make good transitions
Markov chain representation

Similar frames make good transitions
Transition costs

• Transition from i to j if successor of i is similar to j
• Cost function: \( C_{i \rightarrow j} = D_{i+1, j} \)
Transition probabilities

Probability for transition $P_{i \rightarrow j}$ inversely related to cost:

$$P_{i \rightarrow j} \sim \exp \left( - \frac{C_{i \rightarrow j}}{\sigma^2} \right)$$

high $\sigma$  low $\sigma$
Transition probabilities

$D_{ij}$

$P_{ij}$
Preserving dynamics
Preserving dynamics
Preserving dynamics

- Cost for transition $i \rightarrow j$: $C_{i,j}$

\[ N \begin{bmatrix} 1 \\ \vdots \\ N \end{bmatrix} w_k \begin{bmatrix} D_{i,k} \\ D_{i+1,k} \\ \vdots \end{bmatrix} 1_k \]
Preserving dynamics – effect

• Cost for transition $i \rightarrow j$: $C_{i,j} = w_k D_{i,k}^N$
Preserving dynamics

- Filter with diagonal kernel, weights $w$.

$$D_{ij}' = \sum_{k=-m}^{m-1} w_k D_{i+k,j+k}$$

- Filter with diagonal kernel, weights $w$. 

slide by Gabriel Brostow and Tim Weyrich
Dead ends

• No good transition at the end of sequence
Future cost

- Propagate future transition costs backward
- Iteratively compute new cost

\[ F_{i \rightarrow j} = C_{i \rightarrow j} + \alpha \min_k F_{j \rightarrow k} \]
Future cost

- Propagate future transition costs backward
- Iteratively compute new cost

\[ F_{i \rightarrow j} = C_{i \rightarrow j} + \alpha \min_k F_{j \rightarrow k} \]
Future cost

- Propagate future transition costs backward
- Iteratively compute new cost

\[ F_{i\rightarrow j} = C_{i\rightarrow j} + \alpha \min_k F_{j\rightarrow k} \]
Future cost

- Propagate future transition costs backward
- Iteratively compute new cost

\[ F_{i \rightarrow j} = C_{i \rightarrow j} + \alpha \min_k F_{j \rightarrow k} \]
Future cost

• Propagate future transition costs backward
• Iteratively compute new cost

\[ F_{i \rightarrow j} = C_{i \rightarrow j} + \alpha \min_k F_{j \rightarrow k} \]

• Q-learning
Future cost – effect
Finding good loops

- Alternative to random transitions
- Precompute set of loops up front using Dynamic Programming
Visual discontinuities

• Problem: Visible “Jumps”
Crossfading

• Solution: Crossfade from one sequence to the other.

\[
\begin{align*}
A_{i-2} & \quad \frac{3}{4} \quad A_{i-1} & \quad \frac{2}{4} \quad A_i & \quad \frac{1}{4} \quad A_{i+1} \\
& \quad + \quad \frac{1}{4} \quad B_{j-2} & \quad + \quad \frac{2}{4} \quad B_{j-1} & \quad + \quad \frac{3}{4} \quad B_j & \quad B_{j+1} \\
A_{i-2} & \quad A_{i-1}/B_{j-2} & \quad A_{i-1}/B_{j-2} & \quad A_{i-1}/B_{j-2} & \quad B_{j+1} \\
& \quad & \quad & \quad & \quad \\
\end{align*}
\]
Morphing

• Interpolation task:

\[
\frac{2}{5}A + \frac{2}{5}B + \frac{1}{5}C
\]
Morphing

- Interpolation task:

\[ \frac{2}{5} A + \frac{2}{5} B + \frac{1}{5} C \]

- Compute correspondence between pixels of all frames
Morphing

• Interpolation task:

\[
\frac{2}{5} A + \frac{2}{5} B + \frac{1}{5} C
\]

• Compute correspondence between pixels of all frames
• Interpolate pixel position and color in morphed frame
• based on [Shum 2000]
Results – crossfading/morphing
Results – crossfading/morphing

- Jump Cut
- Crossfade
- Morph

slide by Arno Schödl
A final example
The Chemical Brothers - Star Guitar
(Director: Michel Gondry)

https://www.youtube.com/watch?v=0S43IwBF0uM