Texture Synthesis

Tiantian Liu
Definition

• Texture

- Texture refers to the properties held and sensations caused by the external surface of objects received through the sense of touch. Texture is sometimes used to describe the feel of non-tactile sensations. Texture can also be termed as a pattern that has been scaled down (especially in case of two dimensional non-tactile textures) where the individual elements that go on to make the pattern not distinguishable.
Definition

- Texture
  - Local
    - Each pixel of a texture image is characterized by a small set of spatially neighboring pixels
  - Stationary
    - characterization is the same for all pixels
Definition

• Texture
Definition

• Texture

- Animal Skin
- Stone
- Water
- Hair
- Fabric
- Bricks
Definition

• Texture Synthesis
  – Texture synthesis is the process of algorithmically constructing a large digital image from a small digital sample image by taking advantage of its structural content. It is object of research to computer graphics and is used in many fields, amongst others digital image editing, 3D computer graphics and post-production of films.
Definition

• Texture Synthesis
Application

- Surface Texture producing
Application

• Image Editing

Texture Synthesis
Application

- Image Editing
Application

- Flow-controlled 2D texture
Papers

• Fast Texture Synthesis using Tree-structured Vector Quantization

• Texture Optimization for Example-based Synthesis
  – Kwatra V, Essa I, Bobick A, Kwatra N. 2005

• Accelerated Parallel Texture Optimization
  – Huang HD, Tong X, Wang WC. 2007
Papers

• Fast Texture Synthesis using Tree-structured Vector Quantization

• Texture Optimization for Example-based Synthesis
  – Kwatra V, Essa I, Bobick A, Kwatra N. 2005

• Accelerated Parallel Texture Optimization
  – Huang HD, Tong X, Wang WC. 2007
Fast Texture Synthesis using Tree-structured Vector Quantization

Li-Yi Wei  
Marc Levoy  
Stanford University

Figure 1: Our texture generation process takes an example texture patch (left) and a random noise (middle) as input, and modifies this random noise to make it look like the given example texture. The synthesized texture (right) can be of arbitrary size, and is perceived as very similar to the given example. Using our algorithm, textures can be generated within seconds, and the synthesized results are always tileable.
Fast Texture Synthesis

Abstract

Texture synthesis is important for many applications in computer graphics, vision, and image processing. However, it remains difficult to design an algorithm that is both efficient and capable of generating high quality results. In this paper, we present an efficient algorithm for realistic texture synthesis. The algorithm is easy to use and requires only a sample texture as input. It generates textures with perceived quality equal to or better than those produced by previous techniques, but runs two orders of magnitude faster. This permits us to apply texture synthesis to problems where it has traditionally been considered impractical. In particular, we have applied it to constrained synthesis for image editing and temporal texture generation. Our algorithm is derived from Markov Random Field texture models and generates textures through a deterministic searching process. We accelerate this synthesis process using tree-structured vector quantization.

Keywords: Texture Synthesis, Compression Algorithms, Image Processing

1 Introduction

Texture is a ubiquitous visual experience. It can describe a wide variety of surface characteristics such as terrain, plants, minerals, fur and skin. Since reproducing the visual realism of the physical world is a major goal for computer graphics, textures are commonly employed when rendering synthetic images. These textures can be obtained from a variety of sources such as hand-drawn pictures or scanned photographs. Hand-drawn pictures can be aesthetically pleasing, but it is hard to make them photo-realistic. Most scanned images, however, are of inadequate size and can lead to visible seams or repetition if they are directly used for texture mapping.

Texture synthesis is an alternative way to create textures. Because synthetic textures can be made any size, visual repetition is avoided. Texture synthesis can also produce tileable images by properly handling the boundary conditions. Potential applications of texture synthesis are also broad; some examples are image denoising, occlusion fill-in, and compression.

The goal of texture synthesis can be stated as follows: Given a texture sample, synthesize a new texture that, when perceived by a human observer, appears to be generated by the same underlying stochastic process. The major challenges are 1) modeling- how to estimate the stochastic process from a given finite texture sample and 2) sampling- how to develop an efficient sampling procedure to produce new textures from a given model. Both the modeling and sampling parts are essential for the success of texture synthesis: the visual fidelity of generated textures will depend primarily on
Fast Texture Synthesis

- Basic idea
Fast Texture Synthesis

• Basic idea

How large should the neighbor be?
Fast Texture Synthesis

• Basic idea

What if the neighbor is too large or too small?
Fast Texture Synthesis

• Multiple resolutions
Fast Texture Synthesis

- Multi-resolution allows the use of smaller neighborhoods

1 level  
5×5

1 level  
11×11

3 levels  
5×5
Fast Texture Synthesis

• A major issue with pixel-based algorithms introduced so far – they are very slow

• For input with N pixels and output with M pixels
  – $O(MN)$ time complexity
  – Make it $O(cMN)$ for neighborhood size $c$

• A fixed neighborhood allows us to do it faster
  – $O( c \log(N) M )$ – TSVQ
Fast Texture Synthesis

- Acceleration - TSVQ

Search per output pixel

Search tree
Fast Texture Synthesis

• Pseudo Code

```plaintext
function I_s ← TextureSynthesis(I_a, outputSize)
1   I_s ← Initialize(outputSize);
2   G_a ← BuildPyramid(I_a);
3   G_s ← BuildPyramid(I_s);
4   foreach level L from lower to higher resolutions of G_s
5       loop through all pixels (x_s, y_s) of G_s(L)
6           C ← FindBestMatch(G_a, G_s, L, x_s, y_s);
7           G_s(L, x_s, y_s) ← C;
8       I_s ← ReconPyramid(G_s);
9   return I_s;

function C ← FindBestMatch(G_a, G_s, L, x_s, y_s)
1   N_s ← BuildNeighborhood(G_s, L, x_s, y_s);
2   N_a^{best} ← null;   C ← null;
3   loop through all pixels (x_a, y_a) of G_a(L)
4       N_a ← BuildNeighborhood(G_a, L, x_a, y_a);
5           if Match(N_a, N_s) > Match(N_a^{best}, N_s)
6              N_a^{best} ← N_a;   C ← G_a(L, x_a, y_a);
7   return C;
```
Fast Texture Synthesis

• Results

Random

Oriented

Regular

Semi-regular
Fast Texture Synthesis

• Results
Fast Texture Synthesis

• Problems
  – Possibly get stuck on details.
  – Still slow.
Fast Texture Synthesis

• Problems
  – Possibly get stuck on details.
  – Still slow.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Training Time</th>
<th>Synthesis Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efros and Leung</td>
<td>none</td>
<td>1941 seconds</td>
</tr>
<tr>
<td>Exhaustive Searching</td>
<td>none</td>
<td>503 seconds</td>
</tr>
<tr>
<td>TSVQ acceleration</td>
<td>12 seconds</td>
<td>12 seconds</td>
</tr>
</tbody>
</table>

Table 3: A breakdown of running time for the textures shown in Figure 9. The first row shows the timing of Efros and Leung's algorithm. The second and third rows show the timing of our algorithm, using exhaustive searching and TSVQ acceleration, respectively. All the timings were measured using a 195 MHz R10000 processor.
Papers

• Fast Texture Synthesis using Tree-structured Vector Quantization

• Texture Optimization for Example-based Synthesis
  – Kwatra V, Essa I, Bobick A, Kwatra N. 2005

• Accelerated Parallel Texture Optimization
  – Huang HD, Tong X, Wang WC. 2007
Texture Optimization

Texture Optimization for Example-based Synthesis

Vivek Kwatra        Irfan Essa        Aaron Bobick        Nipun Kwatra

GVU Center / College of Computing
Georgia Institute of Technology
{kawatra, irfan, afb, nipun}@cc.gatech.edu

FLOW FIELD: SINK    FRAME 1    FRAME 17    FRAME 1    FRAME 17

Figure 1: Animating texture using a flow field. Shown are keyframes from texture sequences following a sink.
Texture Optimization

Abstract

We present a novel technique for texture synthesis using optimization. We define a Markov Random Field (MRF)-based similarity metric for measuring the quality of synthesized texture with respect to a given input sample. This allows us to formulate the synthesis problem as minimization of an energy function, which is optimized using an Expectation Maximization (EM)-like algorithm. In contrast to most example-based techniques that do region-growing, ours is a joint optimization approach that progressively refines the entire texture. Additionally, our approach is ideally suited to allow for controllable synthesis of textures. Specifically, we demonstrate controllability by animating image textures using flow fields. We allow for general two-dimensional flow fields that may dynamically change over time. Applications of this technique include dynamic texturing of fluid animations and texture-based flow visualization.


Keywords: Texture Synthesis, Energy Minimization, Flow Visualization, Texture Animation, Image-based Rendering.
Texture Optimization

• Texture Similarity and Synthesis
  • Shape, size, orientation of texture elements similar to source
Texture Optimization

• Texture Energy
  – Tricky!
    • Pixel-by-pixel comparison of source and target not possible
  – Compare texture “elements”
    • Local pixel neighborhoods
  – Want each target neighborhood to be similar to some source neighborhood
Texture Optimization

- Texture Energy

\[ Z \quad \text{(source)} \]

\[ X \quad \text{(target frame)} \]
Texture Optimization

- Texture Energy

$Z$ (source)  
$X$ (target frame)
Texture Optimization

• Texture Energy

$Z$ (source)

$X_p$ (neighborhood)

$X$ (target frame)
Texture Optimization

- Texture Energy

\[ Z \] (source)

\[ X_p \) (neighborhood)

\[ X \] (target frame)
Texture Optimization

• Texture Energy

(nearest neighbor) $Z_p$

$Z$ (source)

$X_p$ (neighborhood)

$X$ (target frame)
Texture Optimization

• Texture Energy

$E_t(X_p) = \| X_p - Z_p \|^2$
Texture Optimization

• Texture Energy
Texture Optimization

• Texture Energy

\[ E_t(X) = \sum_{p} \|X_p - Z_p\|^2 \]

\[ = \sum \text{individual neighborhood energy} \]
Texture Optimization

• Flow-guided Texture Animation
  – Flow consistency
    • Perceived motion similar to flow
  – Texture similarity
    • Shape, size, orientation of texture elements similar to source

Texture + Flow Field = Output
Texture Optimization

- Naïve way:
  - Ignore Texture Similarity
  - Warp each frame via flow field

Source Texture → Texture Similarity → Flowing Target → Flow Consistency → Target Flow

Target Frames $X_0$, $X_1$, ..., $X_n$
Texture Optimization

- Warp

Target Frames: $X_0$, $Warp$, $X_1$, ..., $X_n$
Texture Optimization

• Simply Warp
  – Problem: Texture structure not maintained
Texture Optimization

• Warp + Correct
  – Problem: Texture structure not maintained
Texture Optimization

- Link to original goals
Texture Optimization

Energy = \textbf{Flow Energy + Texture Energy}
Texture Optimization

Energy = **Flow Energy + Texture Energy**

\[ E_f(X) = \| X - W \|^2 \]
Texture Optimization

- Optimize Total Energy of target frame:

\[
E(X) = \text{Flow Energy} + \text{Texture Energy}
\]

\[
= \|X - W\|^2 + \sum_p \|X_p - Z_p\|^2
\]

- Initialize: \( X \leftarrow W \)
  (target frame \( \leftarrow \) warped frame)

- Iteratively improve target frame (E-M algorithm!)
Texture Optimization

- **Two variables** we need to optimize over

\[
E(X) = \| X - W \|^2 + \sum_p \| X_p - Z_p \|^2
\]
Texture Optimization

- **Two variables** we need to optimize over

\[
E(X) = \|X - W\|^2 + \sum_p \|X_p - Z_p\|^2
\]

- Target Frame
Texture Optimization

- Two variables we need to optimize over:

\[ E(X) = \|X - W\|^2 + \sum P \|X_p - Z_p\|^2 \]
Texture Optimization

• Iterative Algorithm
  – Step 1
  – Minimize $E(X)$ w.r.t. $\{Z_p\}$

$$E(X) = \|X - W\|^2 + \sum_p \|X_p - Z_p\|^2$$

  – Find Nearest Source Neighborhoods
Texture Optimization

- Iterative Algorithm
  - Step 2
  - Minimize $E(X)$ w.r.t $X$

$$E(X) = \|X - W\|^2 + \sum_p \|X_p - Z_p\|^2$$

- Set $\frac{\partial E}{\partial X} = 0 \iff$ Solve Linear System
Texture Optimization

- Pseudo code

---

Algorithm 1 Texture Synthesis

\[ z^0_p \leftarrow \text{random neighborhood in } Z \quad \forall p \in X^\dagger \]

for iteration \( n = 0 : N \) do

\[ x^{n+1} \leftarrow \arg \min_x E_t(x; \{z^n_p\}) \]

\[ z^{n+1}_p \leftarrow \text{nearest neighbor of } x^{n+1}_p \text{ in } Z \quad \forall p \in X^\dagger \]

if \( z^{n+1}_p = z^n_p \quad \forall p \in X^\dagger \) then

\[ x \leftarrow x^{n+1} \]

break

end if

end for
Texture Optimization

- Multi-level synthesis
  - Random Initialization
  - Multiple
    - Resolution Levels
    - Neighborhood Sizes
  - Progressively refined output

[KEBK05]
Texture Optimization

- Multi-level synthesis
  - Random Initialization
  - Multiple
    - Resolution Levels
    - Neighborhood Sizes
  - Progressively refined output

[KEBK05]
Texture Optimization

• Results

[KEBK05]
Texture Optimization

• Results
Texture Optimization

• Results

[KEBK05]
Texture Optimization

• Results

[KEBK05]
Texture Optimization

- Problems
  - Linear Solve may cause blurriness
    - Solution: Discrete Optimization
  - Can get stuck in local minima
    - Several solutions, but not work well
  - Very very very slow (especially on surfaces)
    - Solution: Exploit parallel hardware
    - Solution: Nearest neighbor queries can be made in parallel
Papers

• Fast Texture Synthesis using Tree-structured Vector Quantization

• Texture Optimization for Example-based Synthesis
  — Kwatra V, Essa I, Bobick A, Kwatra N. 2005

• Accelerated Parallel Texture Optimization
  — Huang HD, Tong X, Wang WC. 2007
Accelerated Parallel Texture Optimization

Hao-Da Huang\textsuperscript{1,3} (黄浩达), Xin Tong\textsuperscript{2} (童 欣), and Wen-Cheng Wang\textsuperscript{1} (王文成)

\textsuperscript{1} State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China
\textsuperscript{2} Microsoft Research Asia, Beijing 100080, China
\textsuperscript{3} Graduate University of Chinese Academy of Sciences, Beijing 100080, China

E-mail: haoda.huang@gmail.com; xtong@microsoft.com; whn@ios.ac.cn

Received November 7, 2006; revised June 19, 2007.

Abstract Texture optimization is a texture synthesis method that can efficiently reproduce various features of exemplar textures. However, its slow synthesis speed limits its usage in many interactive or real time applications. In this paper, we propose a parallel texture optimization algorithm to run on GPUs. In our algorithm, $k$-coherence search and principle component analysis (PCA) are used for hardware acceleration, and two acceleration techniques are further developed to speed up our GPU-based texture optimization. With a reasonable precomputation cost, the online synthesis speed of our algorithm is 4000+ times faster than that of the original texture optimization algorithm and thus our algorithm is capable of interactive applications. The advantages of the new scheme are demonstrated by applying it to interactive editing of flow-guided synthesis.

Keywords texture synthesis, energy minimization, parallel, GPU, flow visualization
Accelerated Parallel Texture Optimization

• GPU-Based Texture Optimization
  – K-coherence search
  – Principal Components Analysis (PCA)
Accelerated Parallel Texture Optimization

- K-coherence search on GPU

Parallel Controllable Texture Synthesis
Sylvain Lefebvre  Hugues Hoppe
Microsoft Research

Figure 1: Given a small exemplar image, our parallel synthesis algorithm computes windows of spatially deterministic texture from an infinite landscape in real-time. Synthesis variation is obtained using a novel jittering technique that enables several intuitive controls.
Accelerated Parallel Texture Optimization

• K-coherence search on GPU

Abstract

We present a texture synthesis scheme based on neighborhood matching, with contributions in two areas: parallelism and control. Our scheme defines an infinite, deterministic, aperiodic texture, from which windows can be computed in real-time on a GPU. We attain high-quality synthesis using a new analysis structure called the Gaussian stack, together with a coordinate upsampling step and a subpass correction approach. Texture variation is achieved by multiresolution jittering of exemplar coordinates. Combined with the local support of parallel synthesis, the jitter enables intuitive user controls including multiscale randomness, spatial modulation over both exemplar and output, feature drag-and-drop, and periodicity constraints. We also introduce synthesis magnification, a fast method for amplifying coarse synthesis results to higher resolution.

Keywords: runtime content synthesis, data amplification, Gaussian stack, neighborhood matching, coordinate jitter, synthesis magnification.
Accelerated Parallel Texture Optimization

• **K-coherence search on GPU (pre-computing)**
  • For each input pixel, find k others with similar neighborhoods
Accelerated Parallel Texture Optimization

- K-coherence search on GPU (searching)
Accelerated Parallel Texture Optimization

• PCA – reducing the dimension
  • mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.
  • theoretically the optimum transform for given data in least square terms
Accelerated Parallel Texture Optimization

• Pseudo Code

Algorithm. Parallel Texture Synthesis
\[ z^0_p \leftarrow \text{random neighborhood in } Z \forall p \in X^+ \]
\[ \text{for iteration } n = 0 : N \text{ do} \]
\[ x^{n+1} \leftarrow \arg \min_{x, x(p) \in D(p)} [E_t(x; \{z^n_p\}) + \lambda E_c(x; u)] \]
\[ // E-step \]
\[ z^{n+1}_p \leftarrow \arg \min_{v, v \in C(p)} [\| \tilde{x}_p - \tilde{v} \|^2 + \lambda E_c(y; u)] \]
\[ // M-step \]
\[ \text{if } z^{n+1}_p = z^n_p \forall p \in X^+ \text{ then} \]
\[ x \leftarrow x^{n+1} \]
\[ \text{break} \]
\[ \text{end if} \]
\[ \text{end for} \]
Accelerated Parallel Texture Optimization

- GPU implementation details:
  - Divide parallel scheme into 3 fragment programs.
    - E-step program
    - M-step program
    - Up sampling program
Accelerated Parallel Texture Optimization

• GPU implementation details:
  • Divide parallel scheme into 3 fragment programs.
    – E-step program
      » Input: neighborhood texture $Y(\text{storing } \{z_p\})$  
      » Output: pixel texture $X(\text{storing } g x)$  
    – M-step program
    – Up sampling program
Accelerated Parallel Texture Optimization

• GPU implementation details:
  • Divide parallel scheme into 3 fragment programs.
    – E-step program
    – M-step program
      » Input: X and related pre-computed data
      » Output: new neighborhood Y
    – Up sampling program
Accelerated Parallel Texture Optimization

• GPU implementation details:
  • Divide parallel scheme into 3 fragment programs.
    – E-step program
    – M-step program
    – Up sampling program
      » Needed in multi-level synthesis.
Accelerated Parallel Texture Optimization

- Results

[HTW 2007]
Accelerated Parallel Texture Optimization

• Comparison

<table>
<thead>
<tr>
<th>Target</th>
<th>Texture Synthesis  $(64^2 \rightarrow 256^2)$</th>
<th>Flow-Guided Synthesis  $(64^2 \rightarrow 256^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Texture Optimization</td>
<td>420~600s</td>
<td>20~60s</td>
</tr>
<tr>
<td>Discrete Texture Optimization</td>
<td>$&gt; 3.6s$</td>
<td>$&gt; 1.2s$</td>
</tr>
<tr>
<td>Basic Parallel Scheme</td>
<td>170~190ms</td>
<td>140~150ms</td>
</tr>
<tr>
<td>Accelerate Parallel Scheme</td>
<td>77~87ms</td>
<td>55~65ms</td>
</tr>
</tbody>
</table>

[HTW 2007]
Sum up

• Texture synthesis has many applications:
  – Surface texture procedure, texture editing, 2D flow controlled texture, spatio-temporal textures, lossy compression etc.

• Current CPU implementation methods are mature:
  – Pixel based, patch based, texturing optimization, user controlled, flow controlled etc.
Sum up (cont.)

• Time consuming is still a problem of texture synthesis.
• Parallel algorithms that can be run on GPUs are few.
Thank You ...