Factoring Repeated Content Within and Among Images

Han-Wook Park

10 March 2011

Korea University
Computer Graphics Lab.
Abstract

• We reduce transmission bandwidth and memory for images by factoring their repeated content
  ▪ Transform map + epitome
    • Affine deformation and color scaling
  ▪ Allows efficient random-access
    • Can be used for real-time texture mapping
  ▪ Orthogonal to traditional image compression
    • Further compression such as DXT
  ▪ Also effective across a collection of images
    • in the context of image-based rendering
Introduction

• Realistic rendering of outdoor scenes requires detailed photographic textures
  ▪ Reasonable bandwidth to transmit & memory space

• Textures often contain repeated patterns
  ▪ Bricks, tiles, windows, etc.
  ▪ Traditional image compression schemes do not detect correlation of high-frequency features across nonlocal neighborhoods
  ▪ Schemes like JPEG require decompression prior to rendering

• Our goal is to exploit the significant repetition of content in the images to reduce bandwidth and memory
Introduction

- Our technique does not assume regular or fixed-frequency repetition
- Factor a given image $I$ into an epitome $E$ and a transform map $\phi$
Introduction
Introduction

• Access to the epitome $E$ uses filtered, access to the transform map $\phi$ uses nearest sampling

• Extracted blocks by using $\phi_t$ can overlap arbitrarily in the epitome

![Image of image blocks and overlapping locations in epitome chart]
Introduction

• Benefits
  ▪ Supports efficient random access
  ▪ Image compression can still be applied
  ▪ The epitome can have nested structure to offer progressive detail level of textures

• Limitations
  ▪ Transform map introduces a memory indirection that can add small access latency
  ▪ The reconstructed image blocks may not match exactly along their boundaries
  ▪ Filtered minification using an epitome mipmap may introduce color bleeding between epitome charts
Representation

• We redefine the transform map to encode local affine deformations

  ▪ Define for each block a $2 \times 3$ matrix $D$
    
    $$I'[p] = E[\phi_D[p/s] p].$$

  ▪ $\phi_D$ is piecewise constant over each image block
  ▪ Matrix $D$ could be a full $3 \times 3$ perspective deformation, but affine deformations form a sufficiently accurate local approximation
Representation

- Repeated image elements may also differ due to low-frequency lighting variations over the image
  - $3 \times 3$ diagonal matrix $L$ to factor out lighting variation

Representation

• For storage efficiency we quantize the coefficients stored in $\phi$

  ▪ 16-bit fixed point numbers for two transition coefficients, 4 additional 8-bit integer for affine deformation
    • 3 fractional bits – max $8K^2$, precision of 0.125,
    • 64 bits/block or 0.25 bits / pixel for $s^2 = 16^2$

  ▪ 8bits/channel for encoding $\phi_L$
Construction

• Input image $I$ is square with size $n \times n$
  - $\phi$ has size $\lceil n/s \rceil \times \lceil n/s \rceil$

• We seek to minimize the size of the two stored textures, $|E| + |\phi|$, as well as the image reconstruction error $\|I' - I\|^2$

$$\min_{s,E,\phi} \lambda (|E| + |\phi|) + \sum_{p \in I} \|E[\phi_D[p/s] p] \phi_L[p/s] - I[p]\|^2$$

  - $\lambda$ provides a tradeoff between accuracy and conciseness
Construction

• For a given block size \( s \) and maximum error \( \epsilon \)
  - Error \( e(B) \) for block \( B \) is
    \[
    e(B) = \frac{\sum_{p \in B} \| I'[p] - I[p] \|^2}{\sigma(I_B)^\alpha + \beta}.
    \]
  - Variance \( \sigma(I_B) \) as a perceptual factor to better preserve low-contrast features
    • With \( 0 \leq \alpha \leq 2 \) and small \( \beta \)
  - Seek for
    \[
    \min_{E, \phi} |E| \text{ such that } \forall B \in I, e(B) \leq \epsilon.
    \]
Construction

- Approximate using a greedy, deterministic, iterative construction process
  - Each epitome chart is a connected set of $4 \times 4$-pixel blocks
  - Maximize the number of new image blocks while minimizing the epitome growth
    - Find self-similarities in $I$
    - Create an epitome chart for each repeated content, to satisfy a maximum norm on the image reconstruction error
    - Optimize the transform map $\phi$, to minimize the reconstruction error given the epitome content
    - Assemble all epitome charts into an epitome atlas $E$
Finding self-similarities

- For each block in the input image
  - Finding self-similarities within tolerance $\epsilon$
Finding self-similarities

- We perform match search using the KLT feature tracker [Lucas and Kanade 1981; Shi and Tomasi 1994]
  - Optimizes affine alignment of two windows
  - Designed for small affine transformations
    - Need a good starting state
Finding self-similarities

• We initialize separate KLT searches at a grid of seed points spaced every $s/4$ pixels
  - Prune the search using color histograms
  - Compute color scaling $L_{ij}$ by dividing the mean colors
    • Color scaling must not exceed 1.25
  - Guess initial rotation using orientation histograms
    $$\theta_{\text{guess}} = \arg \min_{\theta} \sum_{\theta=0^\circ}^{360^\circ} \left( H_{\text{orient}}(\theta, B_i) - H_{\text{orient}}(\theta + \theta', S_j) \right)^2$$
    • 36 buckets over 0-360 degrees using luminance gradient strength
  - Perform separate searches in pre-built image pyramid
    • Only search from minified levels to avoid blurring
  - We also consider both mirror reflections
Finding self-similarities

Histogram $H_{\text{orient}}(\theta, B_j)$

Histogram $H_{\text{orient}}(\theta, S_j)$

Image Pyramid

Block $B_j$

$S_j^0$

$S_j^1$

$S_j^2$

$I_0$: scale = 1.0

$I_1$: scale = 1.5

$I_2$: scale = $1.5^2$

$I_3$: scale = $1.5^3$
Finding self-similarities

• Some image blocks may have an excessive number of matches
  ▪ Ex. sky in a photograph
  ▪ We define a separate relationship of equivalent blocks

• During the search for Match($B_i$)
  ▪ Find another block that is nearly identical up to color scaling $B_j$
    • Low tolerance, no deformation
  ▪ $B_j$ shares the same match list as $B_i$
Creating epitome charts
Creating epitome charts

• When $I^E$ is a set of successfully reconstructed blocks

\[ I^E = \{ B \in I \mid e(B) \leq \epsilon \} \]

- $E$: epitome
- We seek to add the region $\Delta E$ that maximizes

\[ \text{Benefit}(\Delta E) = |I^E + \Delta E \setminus I^E| - |\Delta E| \]

• We want a region that is able to contain the transformed patches from many Match lists
- Find such a candidate region $C_j$ for each $s \times s$ epitome block $B_j$
Creating epitome charts

\[
\text{Cover}(B_j) = \{ M_{i,k} \mid M_{i,k}(B_i) \cap B_j \neq \emptyset \}.
\]
\[
C_j = \{ B \mid B \cap M_{i,k}(B_i) \neq \emptyset, M_{i,k} \in \text{Cover}(B_j) \}.
\]
Creating epitome charts

• The chart growth candidates are
  - \((\Delta E)_j = C_j \setminus E\) for \(B_j\) inside or adjacent to the current chart
  - \((\Delta E)_j = C_j\) for starting a new chart

• Grow until we cannot find any addition for which Benefit\((\Delta E') \geq 0\)
  - Restart growing process at a new location

• Terminate when the whole image is reconstructed
  - \(I^E = I\)
Creating epitome charts
Optimizing the transform map

• After the epitome construction is completed, we iterate through all image blocks $B_i$, determine the location in the epitome that offers the best reconstruction of $B_i$, and update the transform map $\phi$ accordingly

$$\phi[B_i] = \arg \min_{M \in \text{Match}(B_i), M(B_i) \subseteq E} ||B_i - M(B_i)||.$$ 

- This process may create unused epitome areas
  • Remove the unused blocks
Optimizing the transform map

- The quality of the reconstructed image can improve significantly
Assembling charts into an epitome atlas

• We use the heuristic algorithm of [Freivalds et al. 2002]
  - Polyomino packing problem
  - Larger to smaller
  - Small charts are more likely to fit into the gaps left between the larger charts

Chart List = \{ \begin{array}{c} \text{chart} \end{array} \}

\begin{array}{c}
\begin{array}{c}
\text{chart} \\
\text{chart} \\
\text{chart} \\
\text{chart} \\
\text{chart} \\
\end{array}
\end{array}
Hierarchical construction

• For large images, the matching search becomes expensive
  ▪ We have explored a hierarchical construction algorithm

• Partition the image into sub-images \{i\}
  ▪ Obtain its epitome \(E_i\), and then form their union \(E' = \bigcup_i E_i\)
  ▪ Reconstruct \(\phi\) using \(E'\) to trim blocks away
  ▪ Splitting into \(K\) sub-images can potentially provide \(K\) times speed up
Texture mapping

- Mipmapping
  - Color bleeding occurs between different charts
  - Add padding (e.g. 4 pixels)
  - For extreme minification, use original mipmap of image I
Texture mapping

- Chart padding by itself does not guarantee continuous inter-block reconstruction
  - Perform explicit bilinear interpolation
    - 4 closest samples
    - Using pixel shader
    - Fast - 800M pixels/second on an NVIDIA GeForce 8800 GTX
Compression

• $\phi$ compresses well due to its local coherence

- Applying lossless PNG compression to the offset map $\phi_t$ reduces it from 7.06 KB to 4.34 KB
Compression

- The epitome $E$ also can be compressed
  - DXT compression is used for our real-time rendering scenario
    - Supports random access
  - 4 x 4 epitome blocks can be directly copied from compressed DXT blocks
  - Also can be compressed using PNG or JPEG
Compression

Close-up of the 1792×944 input image in Figure 28

(1) Compression after factoring (35KB epitome + 35KB transform map)

(2) Compression of input image with JPEG 2000 (70KB)
Progressive representation

- We can create a nested epitome structure
  - Scalable level-of-detail representation
  - Small epitome $E_1$, larger $E_2$, much larger $E_3$...
    - Store difference only

- Transform map is different, but predictable
  - Many blocks still refer to $E_1$
    - Allows effective compression
Progressive representation

- We first construct \((\phi_2, E_2)\) using a small error threshold \(\epsilon_2\). Next, we construct a coarser representation \((\phi_1, E_1)\) using a large error threshold \(\epsilon_1\)
  - Content of \(E_1\) is constrained to be a subset of \(E_2\)
    - By adaptively removing blocks from \(E_2\)

- Generate remap function \(\psi_2\)
  - \(E_1\) and \(E_2\) are packed differently, need a remapping function

- Overall progressive stream will be

\[
E_1, \phi_1, E_2 \setminus E_1, \psi_2, \text{diff}(\phi_2, \phi_1), \ E_3 \setminus E_2, \psi_3, \text{diff}(\phi_3, \phi_2), \ldots
\]
Factoring image collections

- We apply the hierarchical construction algorithm described before.
Results
Results

Figure 21: Memory size as a function of the image block size $s$ for the example in Figure 1, with fixed error tolerance $\epsilon=0.002$.

Figure 22: Epitome memory size as a function of error tolerance $\epsilon$ for the example in Figure 1, with fixed block size $s=12$. The size of the original image (744KB) is indicated by the red square.
Results

**Detail Transfer**

- Raw image
- Input $I$
- Reconstruction $I'$

*Figure 23: Example of intra-image detail transfer.*

**Detail Removal**

- Input $I$
- Epitome $E$
- Reconstruction $I'$

*Figure 24: Example of image element “generification.”*
Future work

• Allow editing of the epitome to update shared image elements
• Exploit image factoring for better inpainting
• Speed up the epitome construction
• Improve matching of content across image collections
• Increase the reconstruction quality by using a perceptual metric