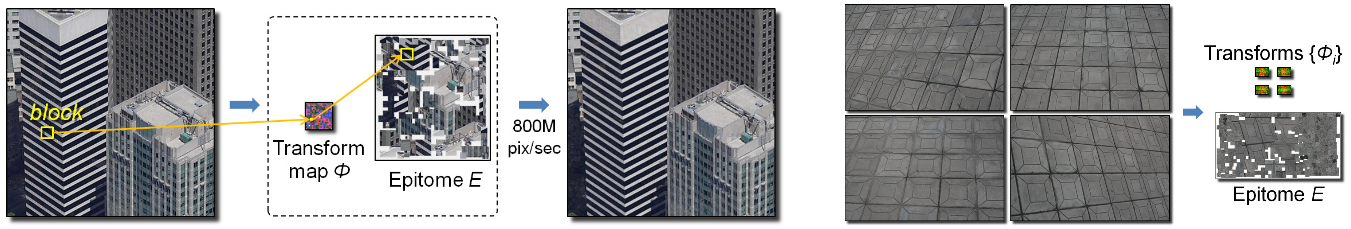
Factoring Repeated Content Within and Among Images

Huamin Wang, Georgia Tech
Yonatan Wexler, Microsoft Corporation
Eyal Ofek, Microsoft Corporation
Hugues Hoppe, Microsoft Research



(a) Input image I Factored representation Reconstructed image I' (b) Factoring a collection of images $\{I_i\}$

Figure 1: Factoring image content into a block-based transform map and a condensed epitome still allows fast random-access evaluation.

Abstract

We reduce transmission bandwidth and memory space for images by factoring their repeated content. A transform map and a condensed epitome are created such that all image blocks can be reconstructed from transformed epitome patches. The transforms may include affine deformation and color scaling to account for perspective and tonal variations across the image. The factored representation allows efficient random-access through a simple indirection, and can therefore be used for real-time texture mapping without expansion in memory. Our scheme is orthogonal to traditional image compression, in the sense that the epitome is amenable to further compression such as DXT. Moreover it allows a new mode of progressivity, whereby generic features appear before unique detail. Factoring is also effective across a collection of images, particularly in the context of image-based rendering. Eliminating redundant content lets us include textures that are several times as large in the same memory space.

Keywords: image compression, image epitomes, progressive images.

1. Introduction

Realistic rendering of outdoor scenes (e.g. with Google Earth or Microsoft Virtual Earth) requires detailed photographic textures, especially for dense city centers with high depth complexity. Two key implementation challenges are: (1) bandwidth to transmit the texture images over the Internet, and (2) memory space to store these textures for rendering on the client device.

Our goal is to exploit the significant repetition of content in the images to reduce bandwidth and memory. For instance, textures often contain repeated patterns such as bricks, tiles, windows, etc. Traditional image compression schemes are not designed to take advantage of such repetitions. Most schemes analyze local image structure, for instance by encoding coefficients of DCT, wavelet, or other transforms. Such methods do not detect correlation of high-frequency features across *nonlocal* neighborhoods. Some codebook techniques like vector quantization can efficiently encode duplicate image blocks, but such duplication arises only if the 2D period of the repeating image content aligns precisely with the block size – a rare circumstance in practice (Section 2).

Another aspect is that schemes like JPEG require decompression *prior* to rendering, and thus do not address the issue of memory space. Often there is simply not enough memory to uncompress the texture content for real-time graphics rendering.

Our idea We introduce a new representation that factors large-scale repeated image content while retaining efficient random access. Our technique does not assume regular or fixed-frequency repetition, so it can handle the natural repetitions (and mixtures of them) that we see in real-world photographs such as in Figure 1.

The basic approach is to factor a given image I into an *epitome* E and a *transform map* ϕ. The epitome contains a set of *charts* that are copied contiguously from the input image, and the transform map encodes how to reconstruct the image by selecting transformed regions of these epitome charts. Specifically, the image is divided into a regular grid of *blocks*, and each block ($s{\times}s$ pixels) is recovered from an epitome *patch*, as illustrated in Figure 2.

In the simplest scheme, the patch is determined by a translation vector t stored as ϕ_t, and the reconstructed image is

$$I'[p] = E\big[p + \phi_t[\lfloor p/s \rfloor]\big].$$

The access to the epitome E uses filtered (e.g. bilinear) sampling, while the access to the transform map ϕ uses nearest sampling.

Note that the translation vectors ϕ_t have finer resolution than block granularity; in fact they may have subpixel precision. Thus, our technique is not forming a simple dictionary of blocks as in vector quantization schemes. Rather, the extracted blocks can overlap arbitrarily in the epitome as shown in Figure 3.

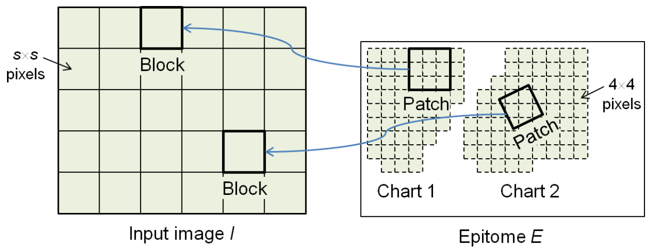


Figure 2: The epitome contains a set of polyomino charts; each image block is reconstructed from a transformed patch in a chart.

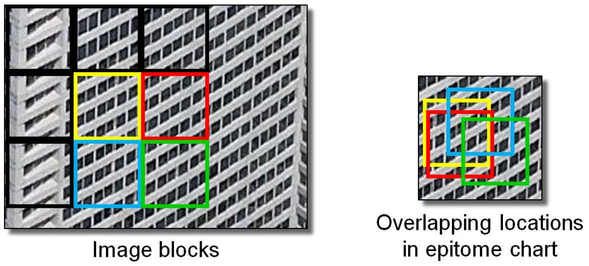


Figure 3: Blocks of the input image often overlap in the epitome.

Intuitively, the goal is to introduce sufficient content into the epitome E such that all blocks of the original image are well approximated by some transformed versions from within E.

Our representation builds on prior work on image epitomes [Jojic et al. 2003; Kannan et al. 2006; Wei et al. 2008]. These earlier techniques develop epitome images as a powerful *analysis* structure, for segmentation, denoising, recognition, indexing, and texture synthesis. Our focus is instead on efficient image *reconstruction*, and we specialize the epitome construction accordingly.

One distinction is that our epitome needs only capture the blocks of the original image, rather than the neighborhoods of all pixels. A more important distinction is that we create a compact transform map $\phi: I \rightarrow E$ from the image to the epitome to allow (lossy) reconstruction of the original image. Some earlier epitome schemes also explored such a map, but at the fine resolution of image pixels, and thus did not enable a concise factoring. Finally, our reconstruction is extremely fast; it provides random access at over 800M pixels/second in a current GPU pixel shader.

If the original image is an exact tiling with period $\tau \times \tau$, then all its blocks can be reconstructed from an epitome chart whose size is only $s\lceil \tau/s \rceil \times s\lceil \tau/s \rceil$ pixels. In addition, several repeated patterns (like multiple building facades) with *different* periodicities can be encoded as separate charts within the same epitome factoring. Representing multiple repeating patterns efficiently within the same texture is advantageous as it can eliminate costly changes in runtime rendering state.

In Section 3 we generalize the transform map ϕ to include geometric deformations, for robustness to perspective view distortion. And optionally, we introduce a low-resolution color scaling map to improve factoring in the presence of the low-frequency lighting variations commonly found in real-world photographs.

Benefits

- The factored representation (ϕ, E) supports efficient random access – color can be evaluated at any point without expanding any data to a temporary memory buffer.
- Image compression can still be applied to the epitome to exploit its fine-scale correlation. Thus our scheme can be viewed as an orthogonal front-end that effectively reduces image size (the number of pixels) by discovering repeating larger-scale content.
- The transform map ϕ is itself highly predictable in regions with either unique or repeating content.
- Like earlier block indirection schemes [e.g. Kraus and Ertl 2002], we encode large image regions of constant or undefined color very effectively while preserving random access.
- The epitome can have nested structure to offer a new mode of progressivity at the level of texture features, whereby generic features are transmitted before specific ones. Unlike the coarse-to-fine or bit-slice progressivity provided by many image compression methods, our scheme quickly recovers a full-resolution detailed image, albeit with fewer unique features.

Limitations

- The transform map introduces a memory indirection that can add access latency. Fortunately, its granularity is coarse (e.g. 16^2 pixel blocks), so the indirected accesses have good locality.
- The reconstructed image blocks may not match exactly along their boundaries. We are able to reduce the resulting blocking artifacts using an interpolation technique in the pixel shader, at a small cost in computation and bandwidth.
- Filtered minification using an epitome mipmap may introduce color bleeding between epitome charts, just as in surface texture atlases. We reduce this artifact by padding the charts with gutter regions, at some cost in storage.

Finally, we explore how factoring can be applied to image collections (Figure 1b). This capability is especially exciting for image-based rendering approaches that store multiple photos of the same scene from different viewpoints. While some content is unique to each image, for instance due to occlusion, large portions of the scene appear in two or more images and can therefore be factored.

2. Related work

Vector quantization Images can be compressed using a codebook of image blocks [Gersho and Gray 1992; Beers et al. 1996]. Levoy and Hanrahan [1996] apply VQ to collections of images within light fields. All these schemes use small blocks (e.g. 4×4 pixels), because with larger blocks the codebook cannot accurately capture the wide variety of block content. The difficulty is that even if the image content is highly repetitive, the rigid placement of the blocks implies that they will most often be unique. By operating on small blocks, VQ is effectively exploiting local data correlation like traditional compression schemes.

Our insight is to introduce fine-grain transformations to enable content reuse using much larger blocks. In turn, larger blocks allow a more compact transform map, and enable more flexible instancing with affine deformations and color scaling. Our approach achieves factoring of content significantly larger than the local analysis performed in traditional compression techniques.

Glyph instancing The JBIG2 compression standard identifies and factors repeated text characters in a segmented image.

Near-regular textures Liu et al. [2004] and Hays et al. [2006] analyze near-regular textures to infer lattice structures and local deformations. Our scheme treats general images with interspersed repeated patterns, not necessarily with lattice structure. Also we focus on a concise representation for real-time rendering.

Procedural modeling Müller et al. [2007] automatically infer a non-uniform lattice from a building facade image, and condense it to an irreducible facade using symmetry simplifications. Such a procedural description allows accurate and precise modeling of repeated grid elements, but with a finely tessellated mesh model.

Epitomic analysis Jojic et al. [2003] create an epitome E from an input image I by iteratively optimizing both E and a mapping $\mathcal{T}: E \rightarrow I$, within a Bayesian framework. The mapping $\mathcal{T}$ contains overlapping patches of three different sizes (4^2, 12^2, 24^2). The inferred $(E, \mathcal{T})$ makes a powerful generative model, but unlike our block-based transform map $\phi: I \rightarrow E$, it is not designed for (and does not permit) random-access reconstruction of the image.

Kannan et al. [2006] form an epitome by automatically learning irregularly shaped jigsaw regions shared within a set of training images. They create a generative model using an offset map $I \rightarrow E$, but this map is stored at the same resolution as I and therefore is not a concise factoring. Wei et al. [2008] resynthesize textures from optimized exemplars using a control map at the same or slightly lower resolution than the original image.

Fractal compression Iterated function systems can approximate an image as the attractor of a set of recursive transformations on both geometry and color [Fisher 1995]. In this challenging setting there is no codebook or epitome, only transformations.

Synthesis magnification In the context of texture synthesis, Lefebvre and Hoppe [2005] sample a high-res. texture using a synthesized low-res. offset map similar to our translation map ϕ_t. They also use blending to hide seams, but perform blending over the whole image rather than just at block boundaries.

Video compression Schemes like MPEG use block-based flow vectors between successive frames as an image predictor. Our epitome combines content from many images at once, factors repetitions within individual images, and supports random access.

Image denoising. The computation of matching windows within an image has also been used for denoising [Dabov et al. 2007].

Shape from textures In computer vision, some schemes identify repeating texture elements in an image to recover 3D scene geometry [Lobay and Forsyth 2006] or to infer the 2D structure of partially occluding objects [Ahuja and Todorovic 2007].

3. Representation

The basic factored representation introduced in Section 1 assumes simple translation transforms. We now describe some generalizations and discuss the encoding of the transform map.

Affine deformations In many cases, the repeated elements of the input image I are not identical translated copies of each other. For example, in an oblique picture of a tiled floor, the tiles undergo perspective foreshortening as they approach the horizon (Figure 4). To account for this, we redefine the transform map to encode local affine deformations. We define for each block a 2×3 matrix D that maps any image point $p = (x\ y\ 1)^T$ to position Dp in the epitome. The reconstructed image is therefore $I'[p] = E[\phi_D[p/s]\,p]$. As before, this affine deformation map ϕ_D is piecewise constant over each image block.

The epitome will typically contain the large-scale versions of the features, while the foreshortened versions are minified instances. Thus an image block near a foreshortened feature is mapped by ϕ_D to some larger quadrilateral region in E. Matrix D could be a full 3×3 perspective deformation, but we found that affine deformations form a sufficiently accurate local approximation.

Color scaling Repeated image elements may also differ due to low-frequency lighting variations over the image. In the tiled floor example, there may be smooth color variations across the floor due to nonuniform illumination. Also, photos in image collections may vary due to different exposure or white-balance parameters. We factor out these lighting variations by introducing a color scaling function, denoted by a 3×3 diagonal matrix L. The transform map becomes the tuple $\phi = (\phi_D, \phi_L)$, such that

$$I'[p] = E[\phi_D[p/s]\,p]\,\phi_L[p/s]\,.$$

Although the 3-channel color scaling map ϕ_L requires additional storage, typically there is a greater benefit in the improved factoring of the lighting-normalized image $I[p]\phi_L^{-1}[p]$. We store ϕ_L as a per-block constant, accessed with nearest sampling like ϕ_D.

The image *function* I' can be interpreted as a composition of functions, $I' = \phi_L \circ E \circ \phi_D$, and therefore (ϕ_L, E, ϕ_D) can be interpreted as a *factoring* of the original image function I.

Encoding of the transform map For storage efficiency we quantize the coefficients stored in ϕ. We find that 16-bit fixed-point numbers are sufficient for the two translation coefficients. With 3 fractional bits, this provides 0.125 subpixel positioning of the transformed blocks while allowing access to an epitome E up to size $(8\text{K})^2$. If the transform includes affine deformations, we store the 4 additional vector coefficients as 8-bit integers. In total these coefficients require 64 bits/block or only 0.25 bits/pixel with a block size of $s^2 = 16^2$. Similarly, we quantize the color scaling map ϕ_L to 8 bits/channel, thus using less than 0.1 bits/pixel.

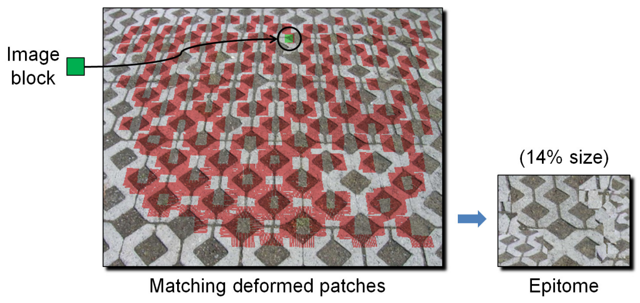


Figure 4: Image blocks match many affinely deformed instances.

4. Construction

We now turn to the problem of constructing a factored image representation. For simplicity let us assume that the input image I is square with size $n\times n$. With a block size $s\times s$, the transform map ϕ has size $\lceil n/s\rceil\times\lceil n/s\rceil$.

Ideal goal We desire the factored representation to be both concise and accurate. Thus we seek to minimize the size of the two stored textures, $|E| + |\phi|$, as well as the image reconstruction error $\|I' - I\|^2$. Mathematically, this can be expressed as

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

where the parameter λ provides a tradeoff between accuracy and conciseness. Let us briefly consider the two extremes.

If conciseness is ignored ($\lambda = 0$), we can achieve a lossless representation by letting the epitome E equal the input image, and letting the transform map ϕ be a 1×1 image containing an identity transform. Thus, lossless representation is possible with negligible storage overhead.

For maximum conciseness ($\lambda \to \infty$), E will contain a single block of size $s = \sqrt{n}$, to reach total storage of only $O(\sqrt{n}\cdot\sqrt{n}) = O(n)$. Thus extremely aggressive compression is also achievable.

Of course, the more interesting case is that of intermediate values of λ, where the representation can hope to factor some repeated image content to form a smaller epitome, but usually at the cost of some reconstruction error.

Our approach To make the problem more tractable, we assume a given block size s (discussed in Section 7) so that $|\phi|$ is fixed. And, rather than minimizing the functional with parameter λ, we instead specify a maximum reconstruction error ϵ that must be satisfied for each image block, and seek the most concise representation that achieves that error threshold.

Let $e(B)$ denote the reconstruction error of an image block B:

$$e(B) = \frac{\sum_{p\in B}\|I'[p] - I[p]\|^2}{\sigma(I_B)^\alpha + \beta}. \qquad (1)$$

Note that the reconstruction I' includes color scaling, which is set such that $\sum_{p\in B} I'[p] = \sum_{p\in B} I[p]$. The variance $\sigma(I_B)$ of the source block is introduced in the denominator (with an exponent $0 \le \alpha \le 2$ and small β) as a perceptual factor to better preserve low-contrast features in relatively smooth regions.

We then seek

$$\min_{E,\phi} |E| \quad \text{such that } \forall B \in I,\ e(B) \leq \epsilon\,.$$

We approximate this minimization using a greedy, deterministic construction process that iteratively grows epitome charts copied from the input image. Each epitome chart is a connected set of 4×4-pixel blocks, and thus has the shape of a polyomino. The general strategy is to maximize the number of new image blocks $\{B\} \subset I$ that can be accurately reconstructed from the growing epitome, while minimizing the epitome growth.

Overview The construction procedure has the following steps:

- 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 ϕ, to minimize the reconstruction error given the epitome content.
- Assemble all epitome charts into an epitome atlas E.

4.1 Finding self-similarities

For each block in the input image, we compute the set of all transformed regions (patches) in the image with similar content, as shown in Figure 5. That is, for block $B_i \in I$ we find the set $\text{Match}(B_i) = \{M_{i,0}, M_{i,1}, \ldots\}$ of transforms identifying patches of image I that reconstruct B_i within tolerance ϵ. Each transform $M_{i,j} = (D_{i,j}, L_{i,j})$ has an affine deformation and color scaling. Block B_i is compared with $M_{i,j}(B_i)$ using Equation (1).

We perform match search using the Kanade-Lucas-Tomasi (KLT) feature tracker, which optimizes affine alignment of two windows [Lucas and Kanade 1981; Shi and Tomasi 1994]. Because KLT is designed for small translations, rotations, and scalings, it must be initialized with a good starting state.

Translation We initialize separate KLT searches at a grid of seed points spaced every $s/4$ pixels (see Figure 6). We prune the search by only considering seeds whose (precomputed) neighborhood color histograms are sufficiently similar to the queried block. For each candidate position S_j we compute the color scaling $L_{i,j}$ by dividing the mean colors of the two neighborhoods. However, we constrain the color scaling coefficients to not exceed 1.25, to give preference to brighter image content and thereby avoid quantization errors in reconstruction.

Rotation We obtain a starting rotation angle θ_{guess} by comparing orientation histograms of the two neighborhoods (Figure 7):

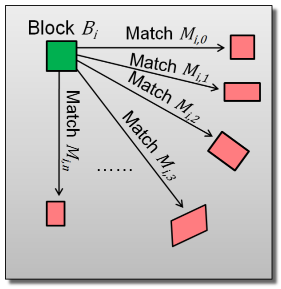
$$\theta_{guess} = \arg\min_{\theta'} \sum_{\theta=0°}^{360°} \left(H_{orient}(\theta, B_i) - H_{orient}(\theta+\theta', S_j)\right)^2.$$

Each histogram contains 36 buckets over the range 0-360 degrees. The value in each bucket is the luminance gradient strength in that orientation integrated over the block. The orientation histograms are precomputed for all block and seed positions.

Scaling We build an image pyramid with sub-octave resolution, and perform separate searches in each pyramid level (Figure 8). We only consider pyramid levels that are minified, i.e. corresponding to matches that are magnified in the original image, to avoid blurring in the reconstruction.

Reflection Finally, we consider both mirror reflections since these are representable by the affine deformations.

Some image blocks may have an excessive number of matches. For example, the sky in a photograph often contains blocks that all match each other, resulting in clique of $O(n^4)$ complexity. To overcome this problem we define a separate relationship of *equivalent blocks*. If during the search for $\text{Match}(B_i)$ we find another block B_j that is nearly identical up to color scaling (with a tight tolerance and without deformation), we tag B_j to share the same match list as B_i.



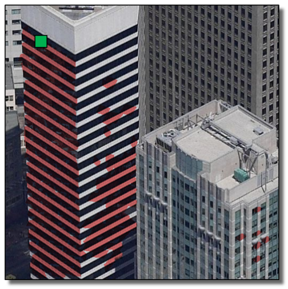


Figure 5: A given image block shown in green accurately matches the transformed patches highlighted in red.

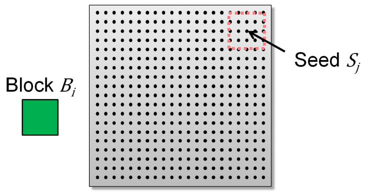


Figure 6: Translation is seeded at a set of grid points.

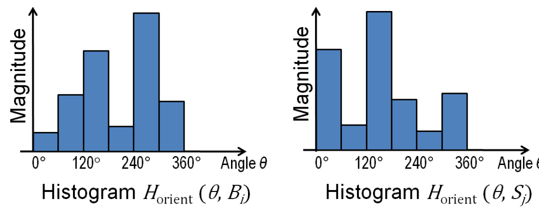


Figure 7: Rotation is initialized using orientation histograms.

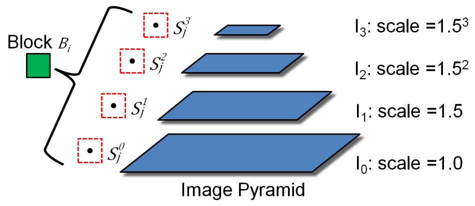


Figure 8: Scaling is handled by searching in pyramid levels.

4.2 Creating epitome charts

We seek to copy charts from the input image into the epitome E, such that the charts can reconstruct other image regions. We grow each chart greedily, trying to account for as many image blocks as possible. Each epitome growth step adds a region $\Delta E \subset I$. Let $I^E \subseteq I$ denote the subset of the image that is accurately reconstructed by epitome E:

$$I^E = \{B \in I \mid e(B) \leq \epsilon\}.$$

Thus we seek to add the region ΔE that maximizes

$$\text{Benefit}(\Delta E) = \left|I^{E+\Delta E} \setminus I^E\right| - |\Delta E|\,.$$

Letting the increment ΔE be a single image block would only match other image blocks that are strictly equivalent. Instead we desire a somewhat larger region that is able to contain the transformed patches from many Match lists. We find such a candidate region C_j for each s×s epitome block B_j as follows (see Figure 9).

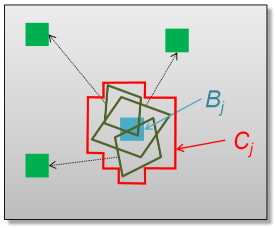


Figure 9: Candidate region C_j for epitome growth, formed as the union of transformed patches that overlap the block B_j.

First we construct an inverse mapping $\mathrm{Cover}(B_j)$ that contains the set of image blocks B_i whose matched patches overlap with B_j:

$$\mathrm{Cover}(B_j) = \{ M_{i,k} \mid M_{i,k}(B_i) \cap B_j \neq \emptyset \}.$$

We then define C_j as the set of epitome blocks necessary to reconstruct all transformed blocks in $\mathrm{Cover}(B_j)$:

$$C_j = \{ B \mid B \cap M_{i,k}(B_i) \neq \emptyset, M_{i,k} \in \mathrm{Cover}(B_j) \}.$$

The chart growth candidates are then $(\Delta E)_j = C_j \backslash E$ for all blocks B_j inside or adjacent to the current chart, or $(\Delta E)_j = C_j$ for all blocks $B_j \notin E$ in the whole image if starting a new chart. Given a current existing chart, if we cannot find any addition ΔE for which $\mathrm{Benefit}(\Delta E) \geq 0$, then we restart the chart growing process at a new location in the image. The process terminates when the whole image is accurately reconstructed, i.e. $I^E = I$.

Figure 10 illustrates the process. The first addition to the epitome is the set C_j that can account for the most image content for its given size (first row). The epitome chart is shown in red, and the matched image content is revealed. We then iteratively grow this epitome chart, resulting in additional matched image content (second row). When incremental growth to the chart is no longer beneficial, a new chart is started at the next most useful location (third row). Figure 11 shows the final result.

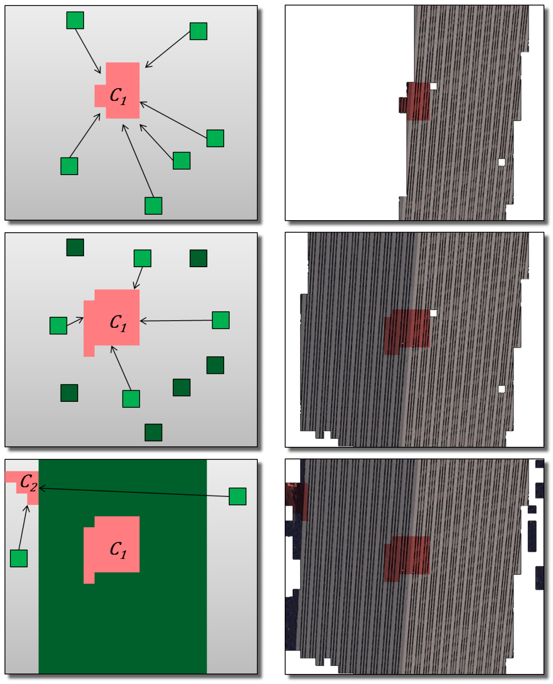


Figure 10: Illustrated example of growing the epitome (in red). The right column reveals the image subset I^E that is accurately reconstructed at each step of the process.

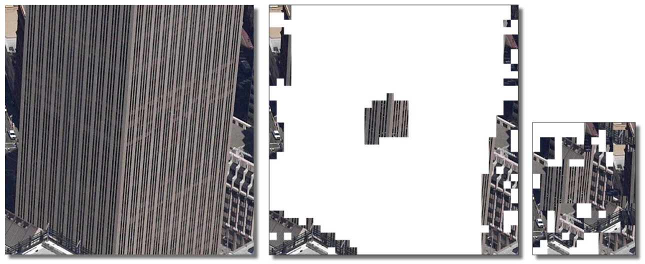


Figure 11: Construction example: input image, epitome charts, and charts packed into an epitome atlas.

4.3 Optimizing the transform map

During the incremental growth of the epitome, each image block B_i is assigned to the first epitome location that reconstructs it sufficiently well, i.e. $M_{i,k}(B_i)$ for some $M_{i,k} \in \mathrm{Match}(B_i)$. However, content subsequently added to the epitome may provide a better reconstruction of block B_i. Therefore, 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 ϕ accordingly:

$$\phi[B_i] = \underset{M \in \mathrm{Match}(B_i),\ M(B_i) \subset E}{\arg\min} \|B_i - M(B_i)\|.$$

The quality of the reconstructed image can improve significantly as shown in Figure 12. This optimization changes which content of the epitome is used during reconstruction, so we remove any unused content by appropriately trimming blocks from the charts.

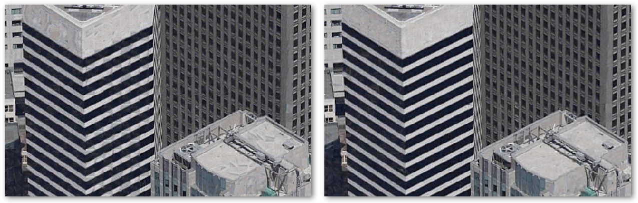


Figure 12: Comparison of image reconstruction before and after optimization of the transform map.

4.4 Assembling charts into an epitome atlas

We pack the charts together into an epitome atlas. This packing problem is related to surface texture atlas packing [Sander et al. 2001; Lévy et al. 2002]. In our case, the charts are polyominoes, so packing is an NP-hard discrete problem. We use the heuristic algorithm of [Freivalds et al. 2002]. The strategy is to consider charts in order of decreasing size, and determine for each chart the optimal placement (including rotation and mirroring) that minimizes the growth in area of the bounding rectangle, as illustrated in Figure 13. The motivation for the heuristic is that small charts are more likely to fit into the gaps left between the larger charts.

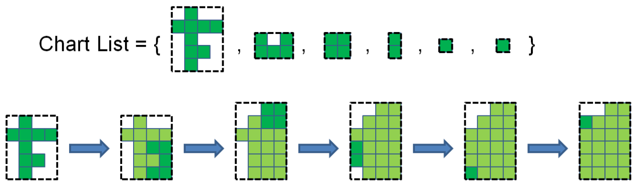


Figure 13: Example of chart packing for a collection of 6 charts.

4.5 Hierarchical construction

For large images, the matching search becomes expensive. As a speedup we have explored a hierarchical construction algorithm. We partition the image into sub-images $\{I_i\}$, factor each sub-image separately to obtain its epitome E_i, and then form their union $\bar{E} = \cup_i E_i$. We then run the construction process with the full image I as input, but restrict the match search to the smaller image $\bar{E}$. Epitome charts that are redundant across the images are trimmed away during optimization (Section 4.3), so we obtain a final epitome E that is more compact than $\bar{E}$. Splitting the image into K sub-images can potentially provide a K times speedup (n^4 versus $K(n^2/K)^2 = n^4/K$). We show this process for the case of image collections in Section 6.

5. Applications

5.1 Texture mapping

To use our factored representation in the context of 3D rendering, we must address two issues:

(1) Enabling texture minification to prevent aliasing.

(2) Obtaining continuous reconstruction across block boundaries.

Mipmapping We allow minification using an ordinary mipmap structure over the epitome texture. However, just as in a surface texture atlas, the epitome consists of irregular charts, so the mipmap pyramid will inevitably contain coarser-level samples whose bilinear basis functions span different charts, and this leads to color bleeding in the reconstructed image. As is commonly done for texture atlases, we reduce this problem by adding a padding gutter (e.g. 4 pixels) between the charts.

At very coarse minification, the access to the transform map ϕ will itself suffer from aliasing. The solution is to store a mipmap of a coarse version of the input image. Fortunately, this coarse mipmap occupies little space. Figure 14 shows a 1D visualization of the overall data structure for minification and magnification.

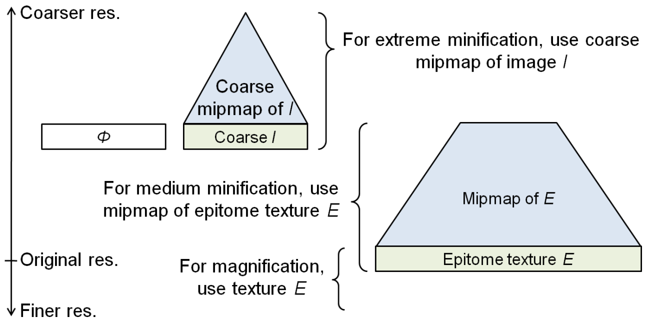

Coarser res.
Original res.
Finer res.
Φ
Coarse mipmap of I
Coarse I
For extreme minification, use coarse mipmap of image I
For medium minification, use mipmap of epitome texture E
Mipmap of E
Epitome texture E
For magnification, use texture E

Figure 14: Strategy for image minification and magnification.

Continuous reconstruction Chart padding by itself does not guarantee continuous inter-block reconstruction, for two reasons:

(1) Due to epitome instancing, the padded samples cannot match all the blocks that may be adjacent in the reconstructed image;

(2) In the presence of affine deformations, the sample positions do not align geometrically at the block boundaries (much like in surface texture atlases).

We guarantee continuous reconstruction by performing explicit bilinear interpolation in the pixel shader. We access the 4 closest samples separately through the transform map (possibly mapping to non-adjacent blocks), and bilinearly blend these sample values. Thus, sampling near the block corners may access up to 4 separate epitome charts. The evaluation is fast: 800M pixels/second on an NVIDIA GeForce 8800 GTX.

Resampling. Many blocks of the reconstructed image I' contain affinely transformed epitome content. The affine warping involves (bilinear) sampling, and therefore introduces a slight amount of blurring. However, images are most often used in texture mapping where similar interpolation also occurs. It is important to note that the final rendering of I' does not introduce additional resampling, because we render with bilinear filtering directly from the epitome which contains original un-resampled content. To make visual comparisons more fair (and in our favor), it might be reasonable to blur the original image I by evaluating it at one-quarter pixel offset in both x and y (an average level of blurring), but we did not do so in the results.

5.2 Compression

The **transform map** ϕ compresses well due to its local coherence (Figure 15). Indeed, if adjacent image blocks access adjacent content in the epitome, their associated translation vectors ϕ_t are identical. With repeating content, adjacent translation vectors often differ by a small multiple (typically 0 or −1) of the tiling period. For the example in Figure 15, applying lossless PNG compression to the offset map ϕ_t reduces it from 7.06 KB to 4.34 KB, or less than 0.14 bits per pixel of I.

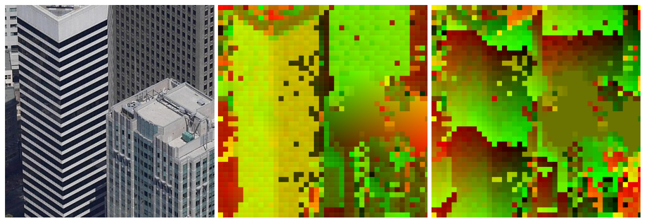
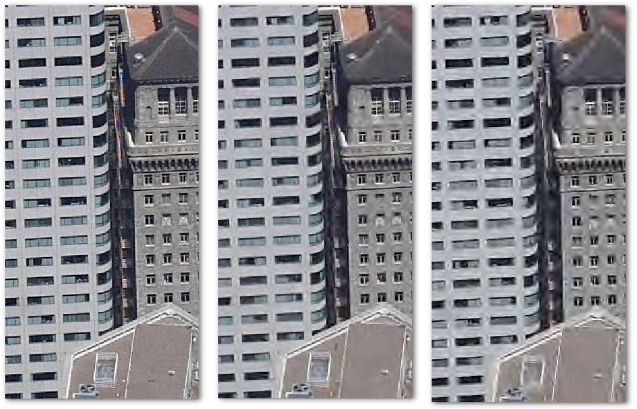


Figure 15: Visualization of the coherence in the transform map. Middle image shows the epitome locations (red=x,green=y) of the reconstruction, and right image shows translation vectors ϕ_t.

The **epitome** E can be compressed with a variety of techniques. For our main scenario of real-time rendering, DXT compression is most appropriate because it supports random access [McCabe and Brothers 1998]. In fact, because the epitome is constructed as a subset of 4x4 blocks in the input image, if the input image is already DXT compressed, we can simply copy those compressed blocks unchanged. Thus our factoring scheme is readily cascaded with DXT compression. For example, when our scheme produces a 4X compression, DXT will provide an additional 4X compression resulting a 16X overall reduction in required memory.

For persistent storage, the epitome can be entropy-compressed using schemes like PNG and JPEG. Ideally, these schemes could benefit from knowledge of the undefined gutter regions between charts. Figure 16 shows some examples that compare (1) our factored representation with its epitome compressed with JPEG 2000, and (2) the original image compressed with JPEG 2000 to have the same overall compressed size. The results show that at high compression rates, (1) is clearly superior to (2).


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)

Figure 16: Image quality when compressing the factored epitome with JPEG 2000, and when compressing the original image with JPEG 2000 to achieve the same overall size. (Because JPEG requires decompression prior to rendering, this comparison is not completely appropriate for our scenario of real-time rendering.)

5.3 Progressive representation

We can create a nested epitome structure to represent increasingly accurate approximations of a given image, in other words a scalable level-of-detail representation.

For example, a rough reconstruction I_1 of image I is obtained using a small epitome image E_1 and an initial transform map ϕ_1. Then, a more accurate reconstruction I_2 of the same image is obtained by adding more image content to form a larger epitome image E_2, together with a new transform map ϕ_2. We enforce that content in E_2 be a superset of the content in image E_1, so that we need only store or transmit the difference $E_2 \backslash E_1$. Although the transform maps ϕ_1 and ϕ_2 differ, many of the blocks in I_2 still refer to content in E_1 and therefore their block transforms in ϕ_2 can be predicted from those in ϕ_1 to allow effective compression.

We have explored two schemes for progressivity of the epitome content, as shown in Figure 17.

The first approach is to organize E_1 and $E_2 \backslash E_1$ as separate sub-images that are **concatenated** together to form E_2. However, a major drawback of this approach is that the incremental content in $E_2 \backslash E_1$ cannot spatially extend the existing epitome charts in E_1 since these are already tightly packed. Consequently, we obtain many new charts in $E_2 \backslash E_1$ whose content overlaps significantly with that already in E_1.

Our preferred approach lets content in E_1 be spatially **remapped** when forming E_2 so that existing charts can be augmented (or partitioned) as needed. We first construct (ϕ_2, E_2) using a small error threshold ϵ_2. Next, we construct a coarser representation (ϕ_1, E_1) using a large error threshold ϵ_1, where epitome content is constrained to be a subset of E_2. This constraint is achieved by adaptively removing unnecessary blocks from E_2.

While all blocks of E_1 also exist in E_2, they generally appear in different locations because E_1 and E_2 are packed independently. Therefore we form E_2 with the help of a remap ψ_2 that records the destination addresses of the blocks from (1) the previous epitome E_1 and (2) the stream of new image blocks $E_2 \backslash E_1$. Fortunately, there is significant spatial locality, so the remap should ψ_2 compress well.

The overall progressive stream of data contains:

$$E_1, \phi_1,\ E_2 \backslash E_1, \psi_2, \text{diff}(\phi_2, \phi_1),\ E_3 \backslash E_2, \psi_3, \text{diff}(\phi_3, \phi_2),\ \dots\ .$$

This progressive representation can be used for transmission, or to select a particular content complexity at load time. An example is shown in Figure 18.

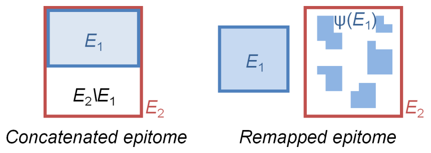


Figure 17 Two progressive schemes; remapped is our preferred.

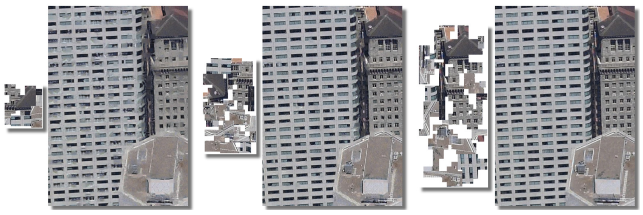

ϵ=0.004; 16x ϵ=0.003; 5.4x ϵ=0.002; 2.5x

Figure 18: Example result of progressivity, for different error tolerances ϵ, showing memory savings factor. This image is challenging due to the variation in windows.

6. Factoring image collections

Image-based rendering techniques allow navigation within a scene using a set of photos taken from several viewpoints [e.g. Chen 1995; Buehler et al. 2001; Snavely et al. 2006]. If a scene region is diffuse, locally planar, and unoccluded, its image in one view may be well reconstructed in a nearby view using local affine deformation and color scaling. Note that our factoring approach operates without scene geometry such as in [Wood et al. 2000].

Given a collection of images $\{I_i\}$, we seek a common epitome E and a set of transform maps $\{\phi_i\}$. This problem can be addressed using our basic construction procedure by simply considering as input the concatenation of all the images. Because the input is large, we apply the hierarchical construction algorithm described in Section 4.5. Figure 19 shows an example result.

The epitome size typically grows sublinearly with the number of input images. In this example, the epitomes have 53K, 69K, 94K, 107K, and 122K pixels for 1..5 input images respectively. The larger delta (94K-69K) between the 2nd and 3rd images is due to the fact that the 3rd image has more glossy reflection.

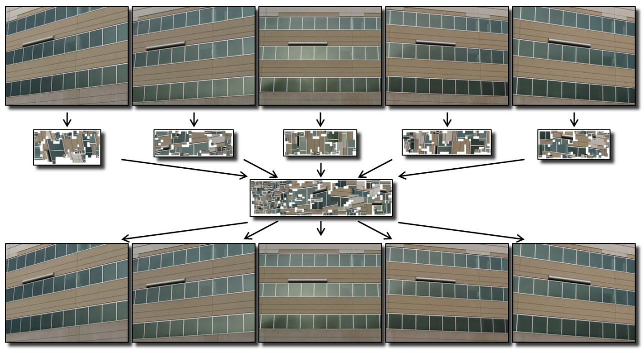
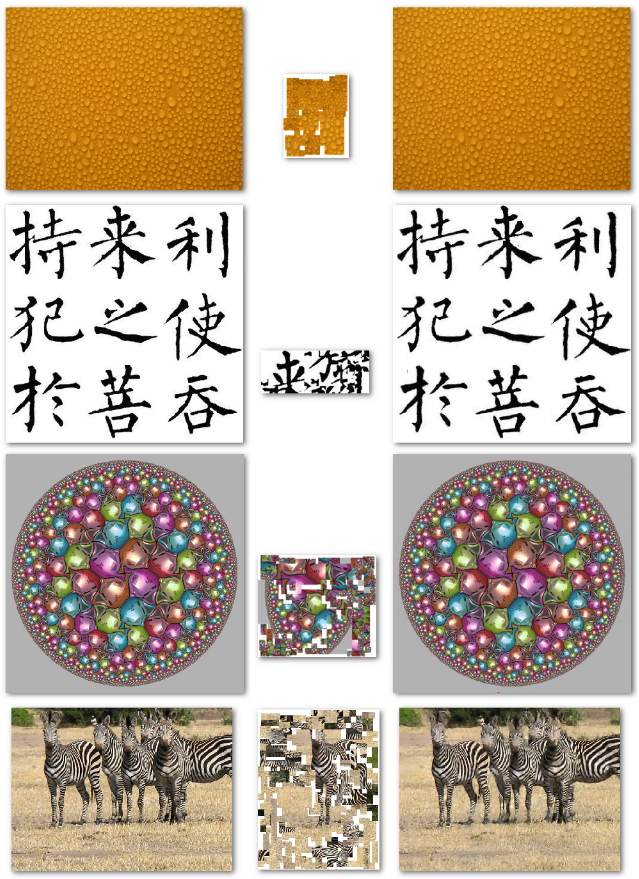


Figure 19: Example of factoring an image collection.

7. Results and discussion

Figure 20 shows additional examples, and Table 1 summarizes quantitative results. The water drops in the first row of Figure 20 are from an actual photograph, and are therefore not perfectly repeating; nonetheless a useful factoring was possible. The characters in the second row are also all distinct; here it is the fact that individual strokes are similar at the scale of a block that allows a condensed epitome. The third row shows a hyperbolic tiling of colored teapots; the factoring is good in the interior, but it appears that too much content is retained near the periphery where the teapot tiles are small. We believe that the KLT-based matching procedure is converging to poor local minima in these high-frequency regions, and this is an area for improvement. The zebras in the last row have a high-frequency pattern that is not repeating at the scale of the blocks, and thus does not factor well.

Construction is an offline preprocess, so we did not invest much in its optimization. Speeding up this process is an interesting area for future research, especially for the case of image collections.



Input I Epitome E Reconstruction I'

Figure 20: Additional image factoring results.

Block size The graph in Figure 21 explores how the memory size (for E, ϕ, and total) varies as a function of the block size s. As the block size increases, the epitome also grows because there are fewer repeating elements of such large size. On the other hand, for very small block sizes, the epitome reaches a minimal size, but the transform map occupies a lot of memory. For this testcase, total memory size has a wide valley for block sizes ranging from 8 to 20, with a minimum at s=12.

Accuracy vs. space Figure 22 graphs epitome memory size as a function the error tolerance ϵ. As expected, the epitome shrinks monotonically as the tolerance is increased. Because the input image (Figure 1) is an actual photograph and does not contain any uniform region such as sky, no content is perfectly repeating, so the tolerance ϵ must reach some nonzero threshold before the epitome size begins to reduce.

Example	Input I	Block size s	Epitome E	Memory savings	RMS error	Time (mins)
Figure 1a	504×504	12	328×232	3.1	2.7%	71
Figure 1b	5×600×396	12	472×252	8.0	2.5%	1100
Figure 3	432×372	12	152×132	6.7	3.1%	65
Figure 4	492×372	12	192×132	6.1	3.0%	77
Figure 11	432×432	12	132×136	8.2	3.3%	70
Figure 18	360×516	12	348×148	3.3	2.6%	43
Figure 19	5×600×480	12	548×196	10.0	2.9%	1420
Figure 20a	624×480	12	260×184	5.4	1.6%	262
Figure 20b	396×396	12	180×72	9.3	1.4%	21
Figure 20c	800×800	16	396×340	4.5	2.3%	146
Figure 20d	552×408	12	304×400	1.8	5.4%	89
Figure 24	396×396	12	152×116	7.3	4.5%	28
Figure 25	592×448	16	140×72	19.1	2.5%	112
Figure 28	1792×944	16	372×1144	3.8	3.9%	1950

Table 1: Quantitative results of image factoring.

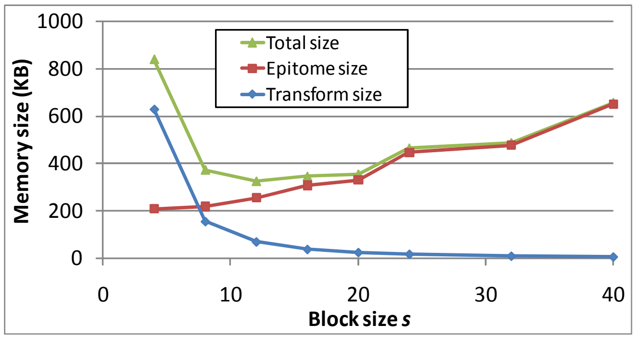


Figure 21: Memory size as function of the image block size s for the example in Figure 1, with fixed error tolerance ϵ=0.002.

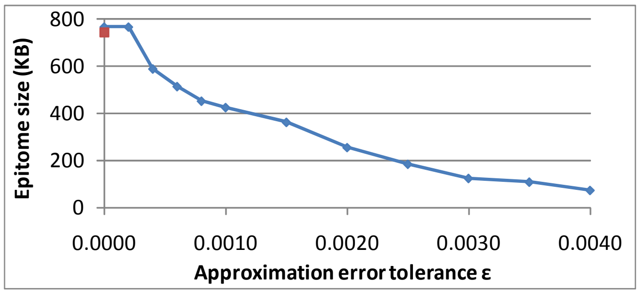


Figure 22: Epitome memory size as function of error tolerance ϵ for the example in Figure 1, with fixed block size s=12. The size of the original image (744KB) is indicated by the red square.

Detail transfer We can use image factoring to transfer detail from finely sampled image regions to coarsely sampled regions as shown in Figure 23. To achieve this result, we use a large block size s=40 and constrain the source of the epitome content to the high-resolution region in the lower third of the image.

Detail removal Some image elements like the tiles in Figure 24 are structurally similar but not exact duplicates due to variations in material and wear. The effect of aggressive image factoring is to carefully preserve the layout of the elements while removing their unique texture features.

Illumination factoring The draping of the cloth in Figure 25 causes both image-space warping and nonuniform shading of the regular texture pattern. These are factored efficiently within the concise transform map. For the user-selected error tolerance, the large appearance variation resulted in two epitome charts. Omitting the color scaling map results in a much larger reconstruction error as shown in Figure 26.

Figure 27 shows more visualizations of the epitome construction process. In the $|\mathrm{Cover}(B_i)|$ image, the bright vertical stripe at the front edge of building is present because those image neighborhoods are able to match content on both adjacent facades. Indeed a single epitome chart started along this edge will maximize the efficiency of the epitome.

General images like the cityscape in Figure 28 are challenging because the many surface occlusions create boundaries with mixtures of patterns, and hence the image blocks along these boundaries are less likely to be repeating. The bottom row reveals its progressive representation. It shows the original image in grayscale, with colors indicating the set of nested epitome charts that are formed for different reconstruction tolerances, from red for the smallest epitome to green for the largest epitome.

8. Summary and future work

Most image compression schemes are designed to exploit local structure in the data. We present an orthogonal technique that exploits the repeated instancing of larger-scale elements, either within a single image or across a collection of images. The factored image representation supports random-access rendering directly from its condensed form.

Some areas of future work include:

- Allow editing of the epitome to update shared image elements, similar to [Brooks and Dodgson 2002].
- Exploit image factoring for better inpainting.
- Speed up the epitome construction.
- Improve matching of content across image collections, perhaps with the help of interest points as in [Brown and Lowe 2003].
- Increase the reconstruction quality by using a perceptual metric.

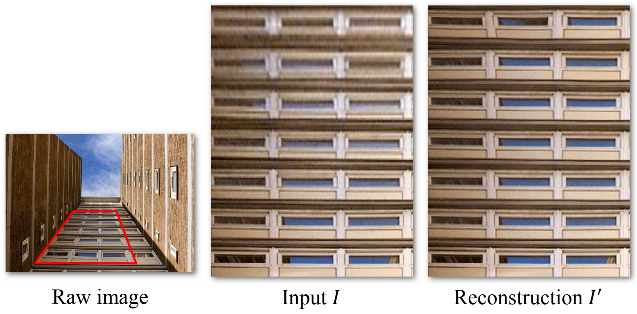


Figure 23: Example of intra-image detail transfer.

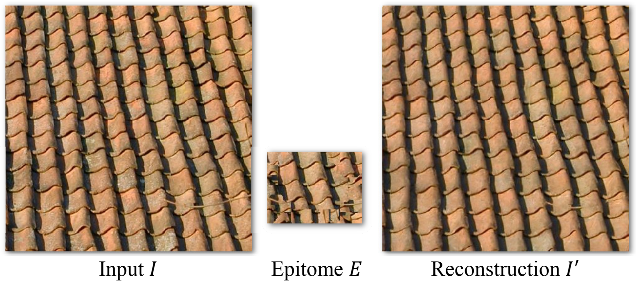


Figure 24: Example of image element "generification".

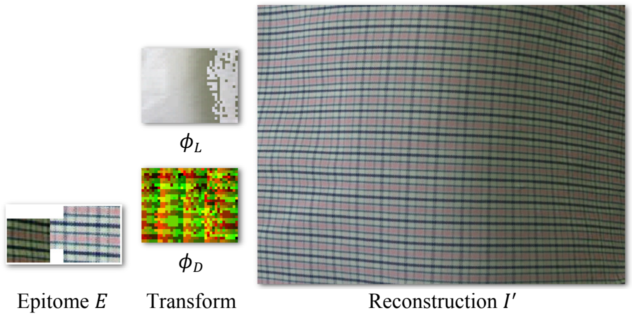


Figure 25: Factoring of both warping and lighting.

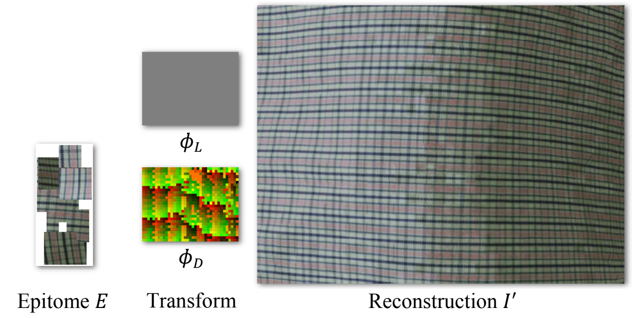


Figure 26: Factoring without color scaling is much less effective.

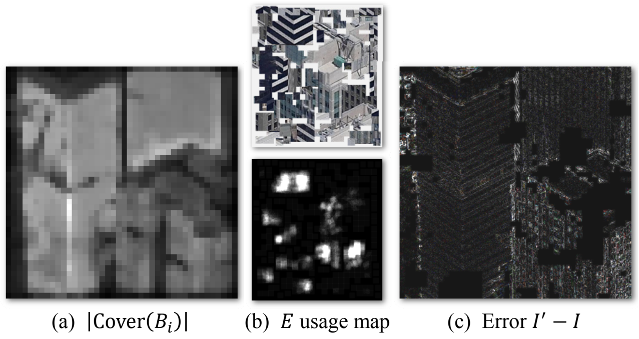
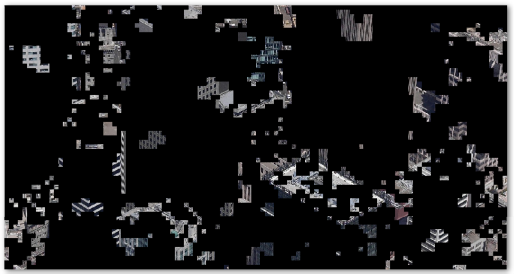
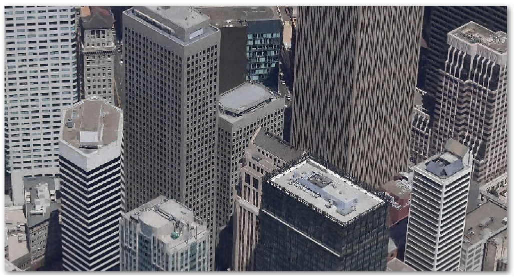


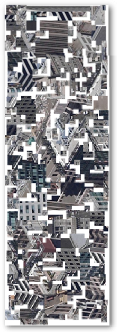
Figure 27: Additional results for the example data in Figure 1: (a) visualization of the sizes of the cover sets (number of blocks that match a given region), (b) frequency of usage of the epitome content, and (c) emphasized reconstruction error.



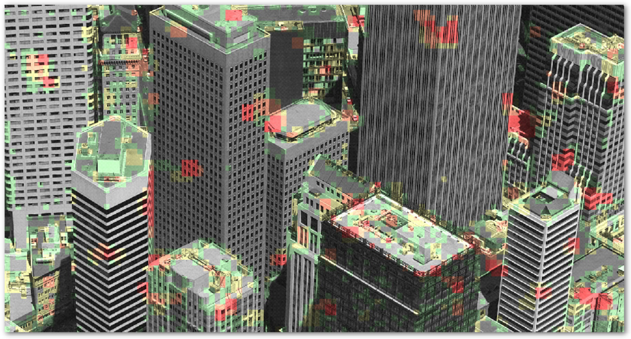
Epitome charts with $\epsilon = 0.002$



Reconstructed image I' (1792×944)



(372×1144)
Epitome E



Locations of progressive epitome charts (for ϵ=0.004,0.003,0.002,0.001)

Figure 28: Example of factoring a large image.

Acknowledgments

We thank Howard Zhou for inspiring us to look at feature trackers to detect self correspondences. We also thank Craig Kaplan for the hyperbolic teapot tiling.

References

AHUJA N., AND TODOROVIC S. 2007. Extracting texels in 2.1D natural textures. *ICCV*.

BEERS A., AGRAWALA M., AND CHADDHA N. 1996. Rendering from compressed textures. *ACM SIGGRAPH*.

BROOKS S., AND DODGSON N. 2002. Self-similarity based texture editing. *ACM SIGGRAPH*.

BROWN M., AND LOWE D. 2003. Recognizing panoramas. *ICCV*.

BUEHLER C., BOSSE M., MCMILLAN L., GORTLER S., AND COHEN M. 2001. Unstructured lumigraph rendering. *SIGGRAPH*.

CHEN S. 1995. QuickTime VR: An image-based approach to virtual environment navigation. *ACM SIGGRAPH*.

DABOV K., FOI A., KATKOVNIK V., AND EGIAZARIAN K. 2007. Image denoising by sparse 3D transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16, 8.

FISHER Y. 1995. *Fractal Image Compression, Theory and Application*. Springer-Verlag, New York.

FREIVALDS K., DOGRUSOZ U., AND KIKUSTS P. 2002. Disconnected graph layout and the polyomino packing approach. *Lecture Notes in Computer Science*, 2265, 378-391.

GERSHO A., AND GRAY R. 1992. *Vector quantization and signal compression*. Kluwer Academic Publishers, Boston.

HAYS J., LEORDEANU M., EFROS A., AND LIU Y. 2006. Discovering texture regularity as a higher-order correspondence problem. *ECCV*.

JOJIC N., FREY B., AND KANNAN A. 2003. Epitomic analysis of appearance and shape. *ICCV*.

KANNAN A., WINN J., AND ROTHER C. 2006. Clustering appearance and shape by learning jigsaws. *NIPS*.

KRAUS M., AND ERTL T. 2002. Adaptive texture maps. *Graphics Hardware*.

LEFEBVRE S., AND HOPPE H. 2005. Parallel controllable texture synthesis. *ACM SIGGRAPH*.

LEVOY M., AND HANRAHAN P. 1996. Light field rendering. *ACM SIGGRAPH*.

LÉVY B., PETITJEAN S., RAY N., AND MAILLOT J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM SIGGRAPH*.

LOBAY A., AND FORSYTH D. 2006. Shape from texture without boundaries. *Int. J. Computer Vision*, 67, 1, 71-91.

LIU Y., LIN W.-C., AND HAYS J. 2004. Near-regular texture analysis and manipulation. *ACM SIGGRAPH*.

LUCAS B., AND KANADE T. 1981. An iterative image registration technique with an application to stereo vision. *Proceedings of Imaging Understanding Workshop*.

MCCABE D., AND BROTHERS J. 1998. DirectX 6 texture map compression. *Game Developer*, 42-46.

MÜLLER P., ZENG G., WONKA P., AND VAN GOOL L. 2007. Image-based procedural modeling of facades. *ACM SIGGRAPH*.

SANDER P., SNYDER J., GORTLER S., AND HOPPE H. 2001. Texture mapping progressive meshes. *ACM SIGGRAPH*.

SHI J., AND TOMASI C. 1994. Good features to track. *CVPR*.

SNAVELY N., SEITZ S., AND SZELISKI R. 2006. Photo Tourism: Exploring photo collections in 3D. *ACM SIGGRAPH*.

WEI L.-Y., HAN J., ZHOU K., HUJUN B., GUO B., AND SHUM H.-Y. 2008. Inverse texture synthesis. *ACM SIGGRAPH*.

WOOD D., AZUMA D., ALDINGER K., CURLESS B., DUCHAMP T., SALESIN D., AND STUETZLE W. 2000. Surface light fields for 3D photography. *ACM SIGGRAPH*.