Multiscale Texture Synthesis

Charles Han Eric Risser Ravi Ramamoorthi Eitan Grinspun

Columbia University

Abstract

Example-based texture synthesis algorithms have gained widespread popularity for their ability to take a single input image and create a perceptually similar non-periodic texture. However, previous methods rely on single input exemplars that can capture only a limited band of spatial scales. For example, synthesizing a continent-like appearance at a variety of zoom levels would require an impractically high input resolution. In this paper, we develop a multiscale texture synthesis algorithm. We propose a novel example-based representation, which we call an exemplar graph, that simply requires a few low-resolution input exemplars at different scales. Moreover, by allowing loops in the graph, we can create infinite zooms and infinitely detailed textures that are impossible with current example-based methods. We also introduce a technique that ameliorates inconsistencies in the user's input, and show that the application of this method yields improved interscale coherence and higher visual quality. We demonstrate optimizations for both CPU and GPU implementations of our method, and use them to produce animations with zooming and panning at multiple scales, as well as static gigapixel-sized images with features spanning many spatial scales.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation

1 Introduction

Example-based texture synthesis algorithms generate a novel image from a given input *exemplar* image. These algorithms make it practical to create a large (or even infinite) coherent, non-periodic texture, while designing or acquiring only a small exemplar of the texture. Unfortunately, the exemplar has a finite (and often small) resolution, and therefore conveys texture information for only a limited band of spatial scales. Texture features larger than the exemplar, or smaller than an exemplar's pixel, are missed altogether. This is a severe shortcoming: many real-world (*multiscale*) textures contain features at widely varying spatial scales. Our work addresses this deficiency by developing a simple method for multiscale texture synthesis from a small group of input exemplars.

We propose a general, succinct, example-based representation, which we call an *exemplar graph* (§3.1). Consider for instance Fig. 1 (top), which depicts the exemplar graph for a map at various scales. When viewed from satellite distance, the texture features are on the order of oceans and land masses. As we zoom in, features take on the shape of coastlines, forests, or mountain ranges. At the finest levels we begin to differentiate rivers, valley systems, and individual ridges. In this graph, each exemplar need only be large enough (in resolution) to faithfully capture those features that

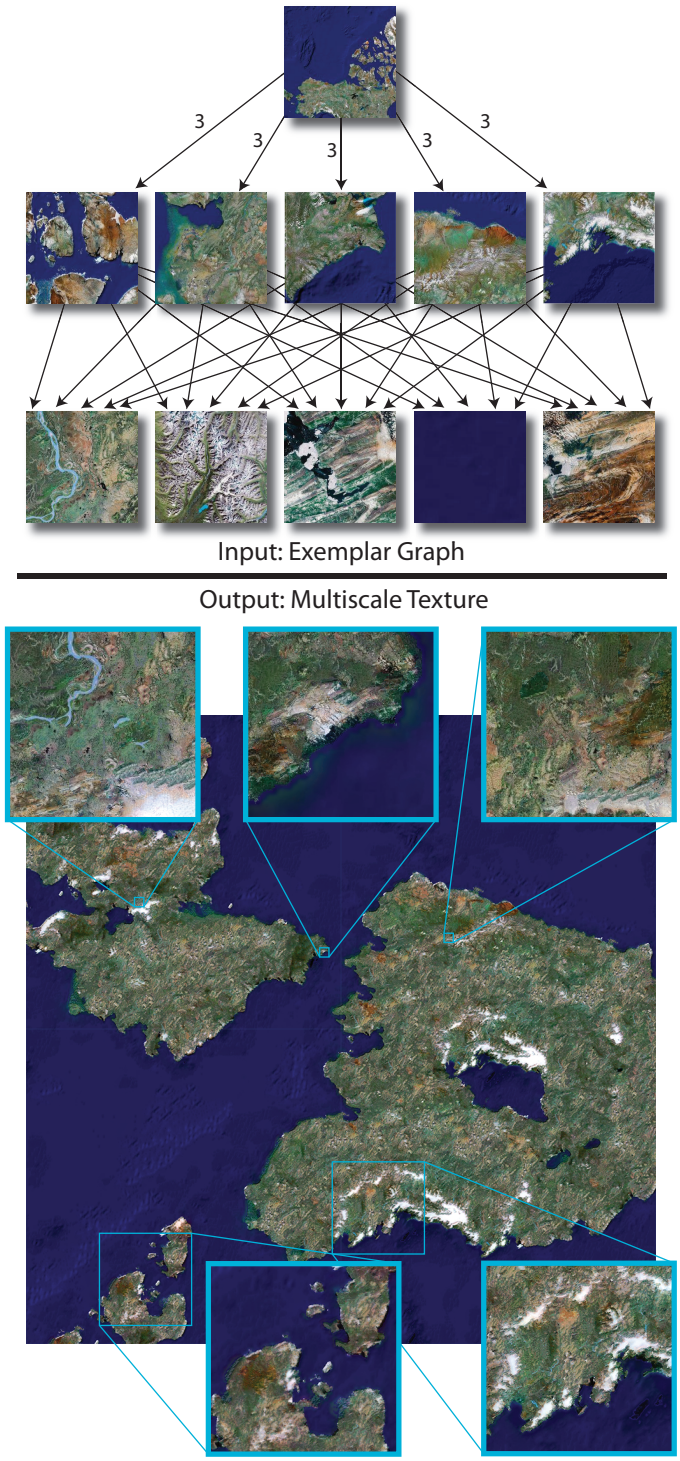

Figure 1: Multiscale texture synthesis. *From a user-defined set of eleven 256 × 256 exemplars and associated scaling relations* (top)*, we synthesize a 16k × 16k texture exhibiting features at all scales* (bottom)*. All figures in this paper have been downsampled to 300dpi; for full resolution images please see the supplemental materials.*

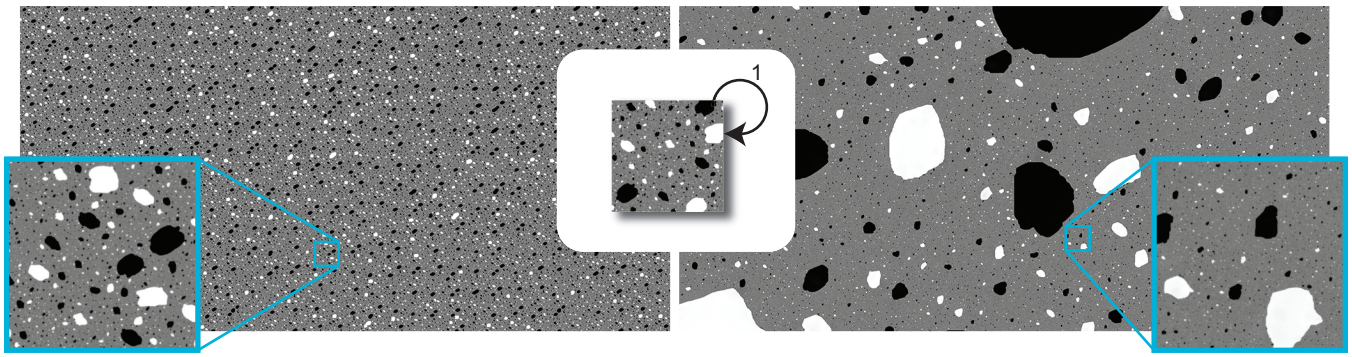

Figure 2: Single- vs. multiscale texture. (left) *A single* 256×256 *exemplar contains information for a limited band of spatial scales, as is evident when a standard synthesis method is used to produce an* $8k\times4k$ *image.* (right) *In contrast, the range of scales synthesized by our method depends on the output (not the exemplar) resolution. Starting with an exemplar graph* (center)*, our synthesis method interprets both the original* 256×256 *texture data together with the self-similarity information in a single reflexive edge to generate an* $8k\times4k$ *image with features spanning multiple scales.*

characterize a particular spatial scale. The graph arrows relate texture structures of differing scale: the head of an arrow points to an upsampled feature present somewhere on its tail, and the label on the arrow gives the *relative scale* between its head and tail.[1] Figure 1 (bottom) depicts a multiscale texture generated from the given exemplar graph. While the multiscale approach produces the entire landscape from eleven 256×256 exemplars, a single-scale method would require a 16k × 16k exemplar.

Loops in the exemplar graph represent an infinitely-detailed, self-similar texture. They enable our method to transform a finite resolution input into an infinite resolution output, that can be navigated by unbounded zooming and panning (see Figs. 2 and 3). Loops make the exemplar graph fundamentally more expressive than a single exemplar, since a single exemplar (of large but finite resolution) cannot allow for infinite levels of detail. By using graphs of exemplars, we take one step toward enjoying the benefits typically associated to *procedural* methods [Perlin 1985; Ebert et al. 2003]—in particular their ability to generate images of arbitrary resolution. At the same time, we allow for synthesis in those settings (*e.g.*, acquired data, artistic design) where a precise mathematical formulation is not readily available.

Example-based texture synthesis is now a relatively mature field with many useful tools and methods. One strength of our framework is that it can directly leverage these existing techniques. In particular, we build on the method of Lefebvre and Hoppe [2005], whose parallel hierarchical synthesis approach provides a natural starting point for our algorithm. We show the insights needed to bridge the gap between conventional and multiscale hierarchical texture synthesis (§4), and furthermore demonstrate optimizations to enable GPU implementation (§6).

With the increased expressive power of exemplar graphs comes an added caveat: the implicit information that the graph gives about the texture function may contain contradictions. In some cases, the user may intentionally provide inconsistent graphs (Figs. 5 and 9, for example), and in other cases, the data acquisition process may not yield consistent data (*e.g.*, changes in lighting). Such contradictions do not exist in single-exemplar setting, where features of all scales are encoded in a single image; our treatment of exemplar graphs must therefore include a discussion of consistency. We present a correction method (§5) that improves interscale coherence in the presence of inconsistencies, and yields higher visual quality.

Our CPU and GPU implementations handle general graphs with arbitrary connectivity, including multiple loops, as evident in numerous examples derived from both user-designed textures and real-world data. Our algorithms can generate gigapixel-sized images exhibiting different features at all scales (*e.g.*, Figs. 1, 7, 8, 9).

[1] All unlabeled edges in Fig. 1 carry a scale relation of $2^3 = 8$.

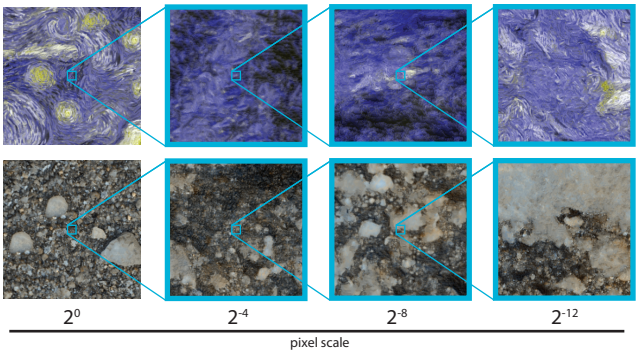

Figure 3: Coherent infinite zooms. *Using a single exemplar with one reflexive edge, we can specify textures with infinite detail. From left to right, each image shows a* 16× *zoom into the previous one. These* self-similar *textures exhibit structure at every scale, all taken from the same exemplar.*

Alternatively, they can render small windows of the multiscale texture at a given spatial position and scale, and even support pans and zooms into infinite resolution textures (Fig. 3).

2 Previous Work

Our work builds on recent literature in texture synthesis, and in particular hierarchical and parallel example-based synthesis.

Texture Synthesis A great deal of recent work synthesizes texture using either parametric [Heeger and Bergen 1995; Portilla and Simoncelli 2000], non-parametric [DeBonet 1997; Efros and Leung 1999; Wei and Levoy 2000], or patch-based [Praun et al. 2000; Efros and Freeman 2001; Liang et al. 2001; Kwatra et al. 2003] approaches. Using only a single exemplar, these methods capture only a limited range of scales.

Hierarchical texture synthesis Hierarchical methods synthesize textures from a single exemplar whose features span varying spatial frequencies [Popat and Picard 1993; Heeger and Bergen 1995; Wei and Levoy 2000]. A hierarchical method synthesizes in a coarse-to-fine manner, establishing the positions of coarse features and refining to add finer ones. This general approach serves as a natural starting point for our work.

Parallel texture synthesis Since multiscale textures are typically very large, our work incorporates ideas from parallel synthesis [Wei and Levoy 2002; Lefebvre and Hoppe 2005] to deterministically synthesize an arbitrary texture window at any scale. This

avoids explicitly rendering to the finest available scale—in fact, recursive exemplar graphs have *no* finest scale!

Multiple exemplars and scales Several existing works employ multiple exemplars, but these methods assume equal scale across all inputs [Heeger and Bergen 1995; Bar-Joseph et al. 2001; Wei 2002; Zalesny et al. 2005; Matusik et al. 2005]. Others take multiple scales into account, either explicitly [Tonietto and Walter 2002] or in the form of local warps [Zhang et al. 2003], but they do not consider scale relationships between exemplars.

3 Key concepts

The input representation for the multiscale synthesis problem is an *exemplar graph*. Given this representation, we present theoretical constructs that will be critical to the multiscale texture synthesis algorithm (§4).

3.1 Exemplar graph

The exemplar graph, $(\mathbb{V},\mathbb{E})$, is a reflexive, directed, weighted graph, whose vertices are the exemplars, $\mathbb{V} = \{E^0, E^1, \ldots\}$, and whose edges, $\mathbb{E}$, denote similarity relations between exemplars. The *root*, E^0, serves as the coarsest-level starting point for synthesis. We fix the spatial units by declaring that root texels have unit diameter. For ease of notation, our exposition assumes that all exemplars have resolution $m \times m$ (where $m = 2^L$), but the formulation can easily be generalized to exemplars of arbitrary size.

Figure 4a shows a simple graph with three exemplars. An edge, $(i, j, r) \in \mathbb{E}$, emanates from a *source* exemplar, E^i, and points to a *destination* exemplar, E^j, and carries an associated *similarity relation* r. In this paper we consider only scaling relations, which we represent by a nonnegative integer r such that 2^r is the spatial scale of the source *relative* to the destination. For example, in Fig. 4a the edge $(0,1,2)$ denotes a transition from E^0 to E^1 along with the interpretation that the diameter of a pixel in E^0 is four (2^2) times the diameter of a pixel in E^1. Likewise, pixels in E^2 are eight times smaller than those of the root. The reflexive edge $(2,2,1)$ indicates that E^2 is similar to a $2\times$ scaling of *itself*. Finally, since exemplars are self-similar, every exemplar has an implicit self-loop (not shown in our figures) with $r = 0$.

We do not restrict the *destination* of an edge; in particular, we permit arbitrary networks including loops (*e.g.*, the self-loop of E^2 in Fig. 4). We do, however, require r to be less than some maximum value r_{max}; this ensures sufficient overlap between source and destination scales, as this is required to reconstruct intermediate scales. In our experience we found that $r_{max} = L - 3$ provided good results.

3.2 Gaussian stacks

We associate to each exemplar, E^i, its Gaussian stack, $E^i_0, E^i_1, \ldots, E^i_L$ [Lefebvre and Hoppe 2005]. Each *stack level*, E^i_k, is an $m \times m$ image obtained by filtering the full-resolution exemplar image with a Gaussian kernel of radius 2^{L-k}. Figure 4b shows the Gaussian stacks associated with the exemplar graph in Fig. 4a, positioned to show their relative scales (E^2 is shown twice to reflect its self-similarity relation). The stacks pictured are eight levels tall, corresponding to an exemplar size of 128 ($L = 7$).

3.3 Admissible candidates

In the single-exemplar setting, neighborhood matching (§4.4) operates naturally on neighborhoods chosen from the same stack level as the source texel. The multiscale setting, however, requires us to consider neighborhoods from multiple candidate stack levels, and–in the presence of loops–possibly even from multiple levels within each exemplar.

The *admissible candidates* for stack level E^i_k,

$$\mathcal{A}(E^i_k) = \{ E^j_l \mid \exists (i, j, k-l) \in \mathbb{E},\ 0 \le l < L \} ,$$

are determined by the exemplar graph edges emanating from E^i, and their associated scaling relations. For example, the admissible candidate sets for three different stack levels are shown with dashed lines in Fig. 4b. The set $\mathcal{A}(E^0_5)$ contains E^0_5, E^1_3, and E^2_2, since links $(0,0,0)$, $(0,1,2)$, and $(0,2,3)$ exist in the exemplar graph. Notice that E^2_1 is *not* admissible, as there is no link $(0,2,4)$. The finest levels of each stack (E^0_7, for example) are not admissible candidates; this is to enforce that correction (see §4.4) will progress to finer scales and not get "stuck" on a given exemplar. Finally, exemplar graph loops (such as the reflexive edge at E^2) can result in stack levels with candidates from the same exemplar, *e.g.*, $E^2_5 \in \mathcal{A}(E^2_6)$.

4 Multiscale texture synthesis

A *graph* of exemplars opens the door to far more expressive, yet economical, design of textures. The question we address below is how to enjoy the benefits of the graph representation with a minimal set of changes to an existing hierarchical approach. Specifically, we extend the parallel, hierarchical approach of Lefebvre and Hoppe [2005], and adopt their notation where applicable.

4.1 Overview

Adopting the traditional hierarchical approach, we build an image pyramid $S_0, S_1, \ldots, S_T$, in a coarse-to-fine order, where T depends on our desired output image size. The images are not represented by color values, but rather store coordinates, $S_t[p] = (i,k,u)$, of some stack level texel, $E^i_k[u]$. Progressing in a coarse-to-fine manner, each level S_t is generated by (1) upsampling the coordinates of S_{t-1}, (2) jittering these coordinates to introduce spatially-deterministic randomness, and then (3) locally correcting pixel neighborhoods to restore a coherent structure.

Multiscale considerations When using Gaussian stacks one must be careful to consider the *physical scale* of a referenced texel relative to the current synthesis level. We use $h_k = 2^{L-k}$ to denote the regular spacing of a texel in level k of a given stack. In our framework, synthesis pixels are not "synchronized"; each synthesized pixel may point to a different exemplar, and to any level of its Gaussian stack. Therefore, whereas Lefebvre and Hoppe [2005] use a single h for each synthesis level, our spacing must be accounted for on a *per-pixel* basis since each pixel can have a unique relative scale. Additionally, our correction step must also take into account the presence of multiple exemplars. When finding a matching neighborhood for a given pixel, we search within *all* admissible candidate levels (§3.3).

The images shown in this paper and the accompanying materials can be on the order of gigapixels; building and maintaining a synthesis pyramid of this size would be cumbersome and impractical. Rather, we exploit the spatial determinism of the parallel approach to generate smaller windows of the overall finest-scale texture and tile them offline. Alternatively, since we can interpret *any* scale as being the output image resolution, we generate our zooming animations (such as Fig. 3 and the accompanying videos) in real time, with finer resolutions being rendered as needed.

4.2 Upsampling

We refine each pixel in S_{t-1} to form a coherent 2×2 patch in S_t by upsampling its coordinates. Intuitively, pixels in the upsampled image will point to the same exemplar as their parent pixels, but

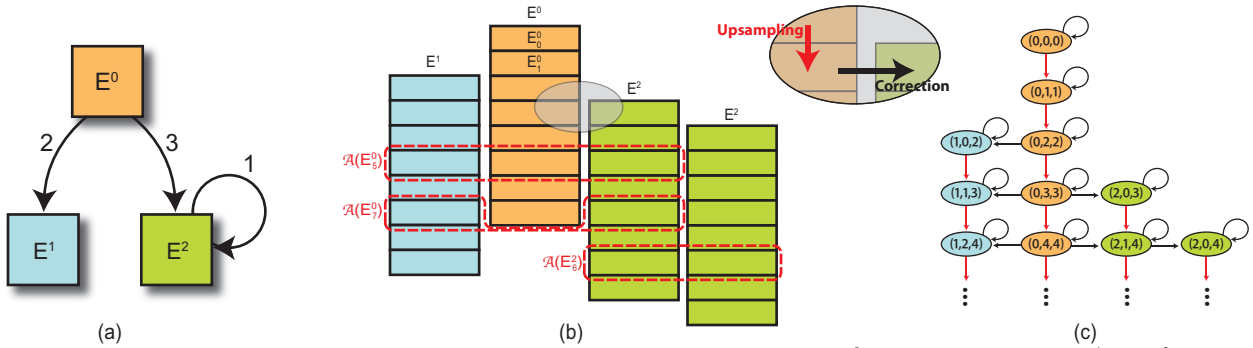

Figure 4: Data structures. (a) *A simple exemplar graph containing three exemplars. The root exemplar, E^0, correlates to exemplars E^1 and E^2 with a scale relation of 4 and 8, respectively.* (b) *Upon computing the Gaussian stacks for each exemplar in the graph, we call those stack levels with equivalent scale* admissible candidates *of one another. To guide the synthesis process towards higher-resolution exemplars, the finest stack levels are considered inadmissible.* (c) *The superexemplar expansion of the graph shown at left. Nodes correspond to stack levels, red edges to upsampling steps, and black edges to correction passes. Node labels give (respectively) exemplar index, stack level, and* red depth*; this last quantity will be used to aid in exemplar graph analysis (§6).*

will move to the next-finer Gaussian stack level. Using $(i,k,u) = S_{t-1}[p]$, the upsampled patch is defined by

$$S_t\left[2p+\Delta+\left(\tfrac{1}{2},\tfrac{1}{2}\right)\right] := \left(i,\, k+1,\, u+\lfloor h_k\Delta \rfloor \pmod m\right) ,$$

where $\Delta \in \left\{\left(\pm\tfrac{1}{2},\pm\tfrac{1}{2}\right)\right\}$.

4.3 Jitter

Next, we jitter the coordinates. Using $(i,k,u) = S_t[p]$, the jittered pixels are

$$S_t[p] := \left(i,k,u+J_t(p) \pmod m\right) , \text{ where } J_t(p) = \left\lfloor h_k \mathcal{H}(p)\rho_t + \left(\tfrac{1}{2},\tfrac{1}{2}\right)\right\rfloor .$$

This jittering step directly follows that of Lefebvre and Hoppe [2005], and we use the hash function, $\mathcal{H}$, and the level-dependent randomness coefficient, $\rho_t \in [0,1]$, defined therein.

4.4 Correction

For each synthesized pixel, $S_t[p] = (i,k,u)$, the correction step seeks among all admissible stack levels, $E_l^j \in \mathcal{A}(E_k^i)$, a texel $E_l^j[v]$, whose local 5×5 exemplar neighborhood best matches the local 5×5 synthesis neighborhood of $S_t[p]$. Formally,

$E_l^j[v]$ is the minimizer of the error functional

$$\sum_{\delta\in\{-2\ldots+2\}^2} \left\| {*S_t[p+\delta]} - E_l^j[v+\delta h_l] \right\|^2 \qquad (1)$$

over $E_l^j \in \mathcal{A}(E_k^i)$ and $v \in \{0\ldots 2^L{-}1\}^2$.

Here $*S_t[p]$ dereferences the texel *pointer*, $S_t[p]$, to get the stored texel color. Following Lefebvre and Hoppe [2005], we perform the computation in parallel, splitting into eight subpasses to aid convergence.

Accelerated matching To accelerate neighborhood matching, we use the *k-coherence* search algorithm [Tong et al. 2002]. Given the exemplar graph, our analysis algorithm identifies for each stack level texel, $E_k^i[u]$, the exemplar texels, $E_l^j[v]$, which minimize the error functional

$$\sum_{\delta\in\{-2\ldots+2\}^2} \left\| E_k^i[u+\delta h_k] - E_l^j[v+\delta h_l] \right\|^2 \qquad (2)$$

over $E_l^j \in \mathcal{A}(E_k^i)$ and $v \in \{0\ldots 2^L{-}1\}^2$. We choose the K best (typically, $K = 2$) spatially dispersed candidates [Zelinka and Garland 2002] to form the candidate set $\tilde{\mathcal{A}}(E_k^i[u])$. We then adopt *coherent synthesis* [Ashikhmin 2001], which seeks the minimum of (1) over the set of precomputed candidates

$$\bigcup_{d\in\{-1\ldots1\}^2} \tilde{\mathcal{A}}(*S_t[p+d]) \qquad (3)$$

drawn from the 3×3 synthesis neighborhood; to ensure that the source and destination neighborhoods are aligned, we replace $E_l^j[v+\delta h_l]$ by $E_l^j[v+(\delta-d)h_l]$ in (1).

5 Inconsistency Correction

We now examine the issue of *consistency* across graph edges. Consider, for example, the exemplar graph in Fig. 5, which prescribes a rainbow-stripe pattern at an 8× coarser scale relative to a black-and-white texture. Clearly such a relation is *inconsistent*, since no combination of downsampled neighborhoods in the greyscale image can reproduce the colorful appearance. This problem unfortunately conflicts with our desire to produce a synthesized image that is coherent across scales.

One could simply restrict the space of allowable inputs to include only strictly consistent exemplar graphs, but this would also restrict many useful and desirable applications. We would often like to use data acquired from different sources (for instance, satellite and aerial imagery), but variations in lighting and exposure make it very hard to enforce consistency in these cases. Inconsistency handling is also desirable in that it allows greater expressive power. For example, the artist-designed exemplar graphs in Figs. 5 and 9 are largely inconsistent, yet generate pleasing outputs; were inconsistency not allowed, the same results would have required much more effort on the part of the artist.

Overview Noting the coarse-to-fine direction of hierarchical synthesis, we introduce the axiom that *the visual appearance of a coarser synthesis level constrains the visual appearance of the next finer level, and by induction, all finer synthesis levels.* Considering that a given exemplar is self-consistent by definition, it follows that inconsistencies arise only as a result of inter-exemplar transitions during the correction step. Our strategy will therefore be to describe each transition with an *appearance transfer function*, $\mathbf{r} : RGB \to RGB$, which captures the overall change in appearance between the source and destination stack level neighborhoods. During synthesis, we will keep a history of all transitions by maintaining a *cumulative transfer function*, $\mathbf{R}_t[p]$, at each synthesis pixel, $S_t[p]$. Specifically, $\mathbf{R}_t[p]$ is the composition of all transfer functions encountered during the synthesis of $S_t[p]$, and the rendered color of pixel $S_t[p]$ is now given by $\mathbf{R}_t[p](*S_t[p])$.

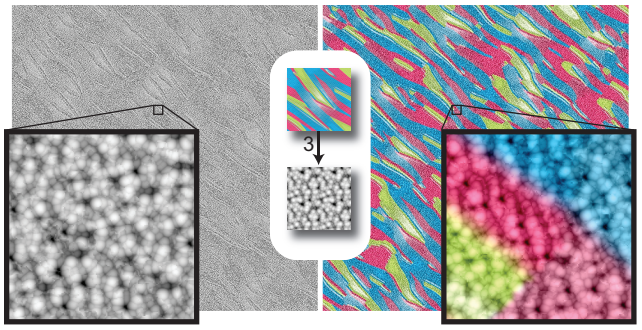

Figure 5: Inconsistency correction. *An exemplar graph* (middle) *may include inconsistent relationships (edge from rainbow-streaked to grey blobby texture). Neighborhoods in the finer (grey) exemplar provide poor matches for those in the coarse (striped) exemplar* (left)*. Inconsistency correction* (right) *repairs this problem by maintaining a color transfer function at each synthesis texel.*

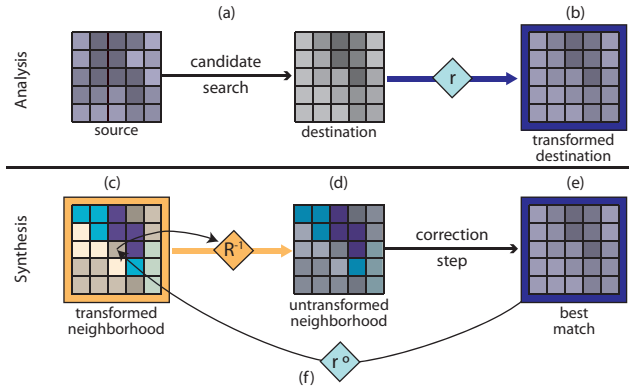

Figure 6: Transfer functions. (a) *For every transition found during analysis,* (b) *we find a transfer function,* $\mathbf{r}$ *that minimizes color error.* (c) *At runtime, we store a cumulative transfer function,* $\mathbf{R}$*, at each synthesis pixel. Since analysis originally took place in the untransformed color space,* (d) *these transfers must be undone before performing the correction step. Finally,* (e) *we arrive at a best-match texel and its associated transfer function, which we* (f) *accumulate into the synthesis pixel by composition.*

To formalize these ideas, consider any transfer function that is linearly composable and invertible. In our implementation, we examined both linear ($\mathbf{r}(\mathbf{c}) = \mathbf{A}\mathbf{c} + \mathbf{b}$) and constant offset ($\mathbf{r}(\mathbf{c}) = \mathbf{c} + \mathbf{b}$) functions, and found that the latter gave good results, a compact and efficiently evaluable representation, and less numerical instability during fitting.

Analysis We will need a transfer function to describe every pixel transition that happens during the correction step. Fortunately, for all source pixels, $E^i_k[u]$, we need only consider a small number of possible destinations, namely the candidate set $E^j_l[v] \in \tilde{\mathcal{A}}(E^i_k[u])$. Consequently, our transfer functions can be computed offline for all precomputed candidates (§4.4).

During the candidate set precomputation (Fig. 6a), we solve for the transfer function that best transforms the destination neighborhood to match the source neighborhood (Fig. 6b), *i.e.*, we optimize $\mathbf{r}$ with respect to the metric

$$\sum_{\delta \in \{-2\ldots+2\}^2} \left\| E^i_k[u+\delta h_k] - \mathbf{r}\left(E^j_l[v+\delta h_l]\right) \right\|^2 . \qquad (4)$$

Given our choice of transfer function, $\mathbf{r}(\mathbf{c}) = \mathbf{c} + \mathbf{b}$, and the use of 5×5 neighborhoods, this yields:

$$\mathbf{b} = \frac{1}{25} \sum_{\delta \in \{-2\ldots+2\}^2} \left(E^i_k[u+\delta h_k] - E^j_l[v+\delta h_l] \right) .$$

By our definition of consistency, $\mathbf{r}$ is the identity map ($\mathbf{b}$=$\mathbf{0}$) for intra-exemplar transitions.

Synthesis Recall that the correction step (§4.4) chooses the transition candidate that best matches the current synthesized neighborhood. We would like to match to the appearance of the *transformed* (*i.e.*, viewer-perceived) neighborhood, $\mathbf{R}_t[p](*S_t[p])$ (Fig. 6c). However, the precomputed transfer function was evaluated with respect to the actual (untransformed) texel values. Therefore, we inverse-transform the synthesis neighborhood back to the original exemplar color space used during analysis (Fig. 6d). For our transfer functions, inversion is simply: $\mathbf{r}^{-1}(\mathbf{c}) = \mathbf{c} - \mathbf{b}$. Composing both the forward and inverse transforms, the error functional in equation 1 becomes

$$\sum_{\delta \in \{-2\ldots+2\}^2} \left\| \mathbf{R}_p^{-1}(\mathbf{R}_{p+\delta}(*S_t[p+\delta])) - \mathbf{r}_v\left(E^j_l[v+\delta h_l]\right) \right\|^2 , \qquad (5)$$

where we adopt the shorthand $\mathbf{R}_p = \mathbf{R}_t[p]$. Upon finding the best-match neighborhood (Fig. 6e), we update the synthesis pixel by composing the associated transfer function onto $\mathbf{R}_t[p]$ (Fig. 6f); for constant offset functions, composition simply amounts to adding offsets, $\mathbf{b}$.

During upsampling, we must propagate the cumulative transfer function to the next-finer synthesis level. We found that letting each pixel inherit its parent's transfer function (*i.e.*, a piecewise constant interpolation of $\mathbf{R}_{t+1}$ from $\mathbf{R}_t$) led to blocking artifacts. Instead, we linearly interpolate the transfer functions of the four nearest parents.

6 GPU optimization

It is often useful to have a real-time visualization of synthesized textures, *e.g.*, for tuning of jitter parameters or for application to games. As in the single-exemplar setting [Lefebvre and Hoppe 2005], we will use principal component analysis (PCA) to make neighborhood matching more tractable on a GPU (or, alternatively, faster on a CPU). However, we first define a construction, called the *superexemplar*, that maps the exemplar graph into a form more readily treatable by existing analysis tools.

Superexemplar For a formal definition of the superexemplar, please refer to the appendix; informally, we (a) unroll exemplar graph loops to transform the graph into a (possibly infinite) tree whose root is E^0, (b) expand each exemplar graph vertex into a chain of vertices (each representing a stack level) connected by *red* edges, and (c) for each exemplar graph edge we link corresponding pairs of stack levels with *black* edges. Figure 4c illustrates the superexemplar expansion of the exemplar graph shown in Fig. 4a. Notice that red edges correspond to synthesis upsampling steps, and black edges correspond to synthesis correction steps.

The *red depth* of a vertex is the number of red edges in the unique path from the superexemplar root, E^0_0, to the vertex. This number directly corresponds to the synthesis level, t, at which the superexemplar vertex plays a role. The set of superexemplar vertices of red depth t gives us the set of stack levels that may appear at synthesis level t. This knowledge will enable us to further optimize our algorithm using PCA projection.

PCA projection We accelerate neighborhood matching (§4.4) by projecting the 5×5 pixel neighborhoods into a truncated 6D principal component analysis (PCA) space. However, we make two additional considerations for multiscale synthesis. First, since pixels may transition across multiple stack levels during correction, we must consider *all* stack levels that can participate at a given synthesis level. Using the superexemplar to find all levels at a given depth, we perform PCA on the set of all neighborhoods found therein to compute a suitable PCA basis.

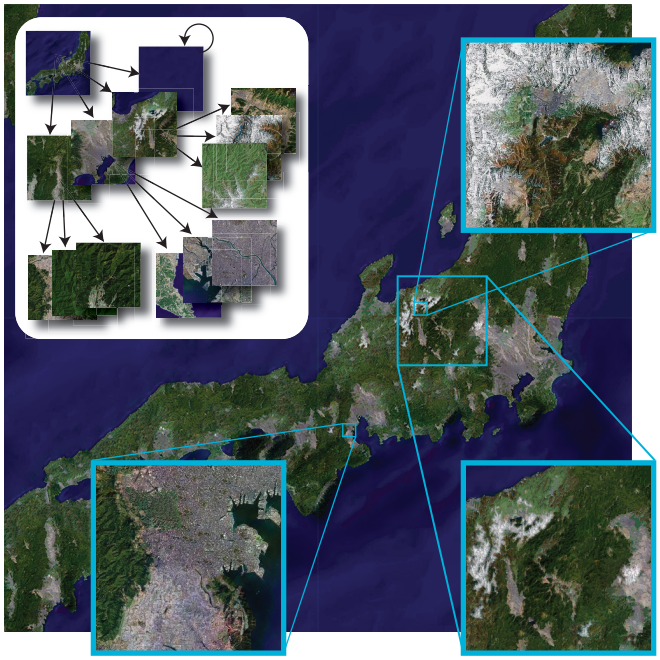

*Figure 7: **Super-resolution.** We use fourteen exemplars and a complex topology (please refer to the supplemental materials to see the full graph) to model a map of Japan. By disabling jitter at the coarsest levels, we "lock in" large features such as mountains and cities; these constrain the proceeding synthesis, which fills in details using the fine-scale exemplars.*

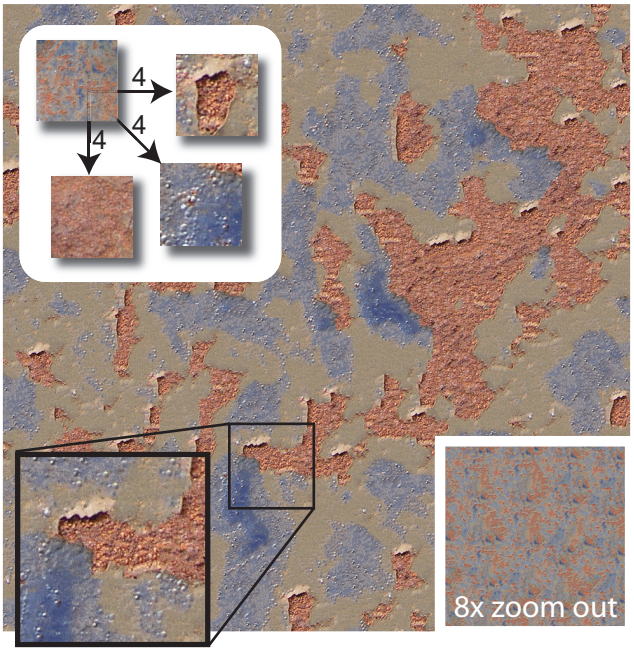

*Figure 8: **Compact representation.** The exemplar graph used here is very small, being comprised of only four 128×128 exemplars; still, we are able to generate a convincing output texture several orders of magnitude larger ($8k \times 8k$, inset).*

To account for the inconsistency correction term in (5), we first transform the target neighborhoods *before* projection into PCA space. Note that a unique transfer function, $\mathbf{r}$, is associated to each candidate destination; we store alongside each candidate its transfer function and its transformed, PCA-projected neighborhood. For the GPU implementation, we also project the RGB color space down to a per-synthesis level 2D PCA space.

Texture packing Since the superexemplar provides all of the w stack levels that participate at level t, it is straightforward to map indices (i,k) at level t to one integer coordinate, $e \in [0 \ldots w-1]$. This allows us to store all needed stack levels in one large $wm \times m$ texture, and to replace the u coordinate universally with $u' = me + u$.

We use one RGB texture for the stack levels $E^e(u)$; three RGBA textures for the two 6D PCA-reduced, inconsistency corrected candidate neighborhoods; and one 16-bit RGBA texture[2] to store each of the candidate links and associated transfer functions, $(\tilde{\mathcal{A}}(E^e(u)), \mathbf{r}_u)$. The synthesis structures $(S[p], \mathbf{R}[p])$ are stored in 16-bit RGBA textures.

7 Results

We now explore the types of results enabled by our multiscale framework. Please note that the figures in this paper have been downsampled to 300dpi; full-resolution versions of all examples are included in the supplementary materials.

Gigapixel textures Figure 1 shows a 16k $\times$ 16k map texture generated using our method. The exemplar graph contains eleven exemplars of size 256×256, with scales spanning over three orders of magnitude. The large resolution of this image is able to capture features at all these scales, and allows us to evaluate the algorithm's success in synthesizing an image with spatial coherence at all scales. We faithfully recreate details at all levels, from the coarse distribution of islands to fine-level terrain details (shown in closeups). Generating such textures using existing single-exemplar methods would require an exemplar on the order of $2^{14} \times 2^{14}$ pixels, or about 400 times more data!

A similar example is shown in Fig. 7, with the key distinction that we have disabled jitter at the coarsest levels. In this light we can interpret our method as a form of super-resolution [Freeman et al. 2001; Hertzmann et al. 2001]. As in previous such approaches, we employ our hierarchical texture synthesis algorithm to fill in high-resolution details on a lower-resolution image—in this case, the root exemplar (a 256 $\times$ 256 map of Japan.) However, we can deal with many more levels of detail beyond the coarse guiding image; the output image shown is again of size 16k $\times$ 16k.

Coherent infinite zooms Figure 3 shows frames from two infinitely zooming animations, with each image containing pixels at $1/16^{\text{th}}$ the scale of the one to its left. Notice that texture characteristics are consistently preserved across all scales. Each sequence was created using a *single* exemplar with a single self-looping edge. What we see here is an example-based approach to creating resolution-independent textures—previously attainable only through procedural methods. Furthermore, our method can utilize both artist-created (van Gogh's *The Starry Night*, top) or captured (a photograph of pebbles, bottom) data. Much longer versions of these zooms and others can be viewed in the accompanying video.

Inconsistency correction In Fig. 5 we demonstrate that our inconsistency correction method is able to compensate for color variations between exemplars. The graph shown prescribes a rainbow stripe pattern at coarse scales, but only links to a black-and-white texture for finer scales. As we see clearly in the left image, the texels in the greyscale exemplar are unable to recreate the desired colorful patterns. In contrast, our inconsistency correction method is able to adjust the colors of fine texels (right inset) to match those encountered at coarser levels in the synthesis.

Artist controllability Finally, we show two examples that demonstrate the compact expressiveness of the exemplar graph rep-

[2] u' will generally exceed the 8-bit limit of 256.

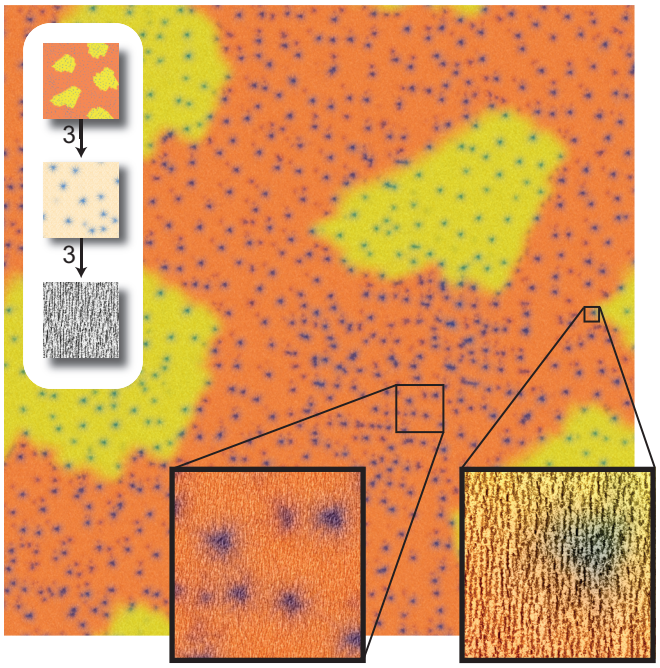

*Figure 9: **A simple chain.** A texture created from a chain of exemplars, exhibiting unique features at three different scales. Crafted in a matter of minutes, this artist-created exemplar graph offers pleasing results that would be much harder to develop using procedural techniques. Note that inconsistencies in the input are repaired by inconsistency correction (§5).*

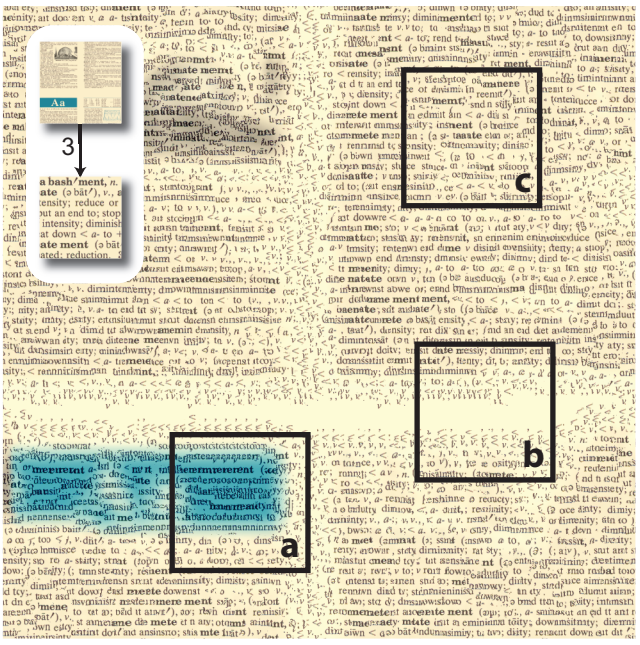

*Figure 10: **Failure modes.** An example illustrating current limitations of our method. For extremely inconsistent inputs,* (a) *the candidate neighborhood search (§4.4) may fail to find coherent matches; such cases cannot be repaired by inconsistency correction.* (b) *Linear interpolation of transfer functions can lead to a smearing effect around hard edges. Additionally,* (c) *semantic structures—such as text—are generally not preserved.*

resentation. The rusted metal surface shown in Fig. 8 was generated using just four 128×128 exemplars all taken from the same high-resolution photograph. For essentially the cost of a single 256×256 exemplar, we can produce large, aperiodic, high-resolution textures (a zoom-out is shown in the inset).

The texture in Fig. 9 was generated from an artist-created chain of exemplars, exhibiting distinct features (yellow splotches, blue dabs of paint, and a grainy surface) at three different scales. Note that the tiny (1–2 pixel) specks in the root exemplar prescribe only the rough placement of the blue dabs, while their wispy details are contributed by the intermediate exemplar. Also notice that we achieve this result despite the largely inconsistent input. The exemplars were made in a matter of minutes, demonstrating the intuitive user control made possible by the exemplar graph; it would be much more difficult and time-consuming to create such effects using procedural methods.

Synthesis performance The zooming animations in Fig. 3 were generated using our GPU implementation, which achieved a synthesis throughput of 9.37M pixels/sec on a GeForce 8800 GTX. In comparison, our implementation of Lefebvre and Hoppe's algorithm [2005] runs at about 15.30M pixels/second on the same machine. A multithreaded implementation of our CPU algorithm runs at about 100k pixels/sec on an 8-core machine.

8 Limitations

Despite inconsistency correction, our method is still sensitive to poorly designed input graphs. It is possible for exemplar regions (or even entire exemplars) to be ignored by the preprocess if there are preferable correspondences present elsewhere in the exemplar graph. Likewise, exemplar regions can be overused if an input contains many identical neighborhoods (*e.g.*, the solid blue region in Fig. 10a). Since we compute our transfer functions *after* finding closest-match neighborhoods, particularly bad cases may not be reparable through simple inconsistency correction alone.

Because we linearly interpolate cumulative transfer functions during upsampling, colors can "bleed" into neighboring areas, especially in regions requiring stronger inconsistency correction. This problem is illustrated in Fig. 10b: at coarser levels, inconsistency correction is able to approximate the text/gutter boundary; however, the hard edge is gradually lost as we continue to refine into finer resolutions.

Another current limitation of our work is that we can often fail to capture semantic structures, particularly if the jitter parameter is set too strongly (Fig. 10c). This is generally an inherent weakness of the pixel-based parallel synthesis approach that we adopt. As with previous such works [Lefebvre and Hoppe 2005], our GPU algorithm can allow for real-time tuning of synthesis parameters. Furthermore, we anticipate that it will be useful to extend other synthesis paradigms, such as patch- [Wu and Yu 2004] or optimization-[Kwatra et al. 2005] based methods, to the multiscale setting in order to inherit their desirable qualities (*e.g.*, feature preservation).

9 Conclusion and Future Directions

We have introduced the multiscale setting for example-based texture synthesis, enabling the creation of textures spanning large or infinite ranges of scale. Our exemplar graph representation and accompanying synthesis framework provide an intuitive method for generating multiscale textures. Furthermore, we show how to optimize for efficient and even real-time GPU synthesis.

Multiscale textures offer many exciting new paths for further research. Particularly, we feel that there are several open directions in multiscale texture *analysis*. We typically arrived at our examples through careful design and iteration, but in the future we anticipate significant research towards user-friendly exemplar graph editing and preview applications. A better understanding of consistency, for instance, could lead to automatic tuning of exemplar

graph edges; one might even imagine applications where exemplar graphs are automatically constructed from large datasets.

In the future, we expect that the multiscale approach will be useful for other classes of textures besides color, such as normal maps [Han et al. 2007] or appearance vectors [Lefebvre and Hoppe 2006]. Other interesting ideas include the synthesis of multiscale textures directly on large meshes, or in application to solid texture synthesis [Kopf et al. 2007]. In summary, we feel that multiscale texture synthesis is simple in concept but powerful in application; it directly provides novel and useful applications, while introducing a host of new avenues for investigation.

Acknowledgments: We would like to thank our anonymous reviewers for their insightful comments. This work was supported in part by the NSF (grants CCF 03-05322, IIS 04-30258, CCF 04-46916, CCF 05-28402, CNS 06-14770, CCF 06-43268, and CCF 07-01775), an ATI Fellowship, a Sloan Research Fellowship, and an ONR Young Investigator award N00014-07-1-0900. We also thank Elsevier, NVIDIA, and Walt Disney Animation Studios for their generous donations.

References

A , M. 2001. Synthesizing natural textures. In *SI3D*, 217–226.

B -J , Z., E -Y , R., L , D., W , M. 2001. Texture mixing and texture movie synthesis using statistical learning. *IEEE TVCG 7*, 2, 120–135.

D B , J. 1997. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH*, 361–368.

E , D. S., M , F. K., P , D., P , K., W - , S. 2003. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann, San Francisco, CA.

E , A., F , W. 2001. Image quilting for texture synthesis and transfer. In *SIGGRAPH*, 341–346.

E , A., L , T. 1999. Texture synthesis by non-parametric sampling. In *ICCV*, 1033–1038.

F , W. T., J , T. R., P , E. C. 2001. Example-based super-resolution. Tech. Rep. TR-2001-30, MERL.

H , C., S , B., R , R., G , E. 2007. Frequency domain normal map filtering. In *SIGGRAPH*, 28.

H , D. J., B , J. R. 1995. Pyramid-based texture analysis/synthesis. In *SIGGRAPH*, 229–238.

H , A., J , C. E., O , N., C , B., S , D. H. 2001. Image analogies. In *SIGGRAPH*, 327–340.

K , J., F , C.-W., C -O , D., D , O., L , D., W , T.-T. 2007. Solid texture synthesis from 2d exemplars. In *SIGGRAPH*, 2.

K , V., S , A., E , I., T , G., B , A. 2003. Graphcut textures: Image and video synthesis using graph cuts. In *SIGGRAPH*, 277–286.

K , V., E , I., B , A. F., K , N. 2005. Texture optimization for example-based synthesis. In *SIGGRAPH*, 795–802.

L , S., H , H. 2005. Parallel controllable texture synthesis. In *SIGGRAPH*, 777–786.

L , S., H , H. 2006. Appearance-space texture synthesis. In *SIGGRAPH*, 541–548.

L , L., L , C., X , Y., G , B., S , H. 2001. Real-time texture synthesis by patch-based sampling. Tech. Rep. MSR-TR-2001-40, Microsoft Research.

M , W., Z , M., D , F. 2005. Texture design using a simplicial complex of morphable textures. In *SIGGRAPH*, 787–794.

P , K. 1985. An image synthesizer. In *SIGGRAPH*, 287–296.

P , K., P , R. 1993. Novel cluster-based probability model for texture synthesis, classification, and compression. In *SPIE VCIP*, 756–768.

P , J., S , E. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *IJCV 40*, 1, 49–70.

P , E., F , A., H , H. 2000. Lapped textures. In *SIGGRAPH*, 465–470.

T , X., Z , J., L , L., W , X., G , B., S , H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH*, 665–672.

T , L., W , M. 2002. Towards local control for image-based texture synthesis. In *SIBGRAPI*, 252.

W , L., L , M. 2000. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH*, 355–360.

W , L., L , M. 2002. Order-independent texture synthesis. Tech. Rep. TR-2002-01, Stanford University CS Dept.

W , L.-Y. 2002. *Texture synthesis by fixed neighborhood searching*. PhD thesis, Stanford University.

W , Q., Y , Y. 2004. Feature matching and deformation for texture synthesis. In *SIGGRAPH*, 364–367.

Z , A., F , V., C , G., G , L. V. 2005. Composite texture synthesis. *IJCV 62*, 1-2, 161–176.

Z , S., G , M. 2002. Towards real-time texture synthesis with the jump map. In *EGWR*, 99–104.

Z , J., Z , K., V , L., G , B., S , H.-Y. 2003. Synthesis of progressively-variant textures on arbitrary surfaces. In *SIGGRAPH*, 295–302.

Appendix: superexemplar, formalized

The superexemplar is a tree with root E_0^0 and directed red and black edges. Each vertex, $(i,k,t) \in \mathbb{V}^*$, points to a stack level, E_k^i, and its name includes a red depth counter, t. We build the superexemplar from the exemplar graph by induction:

Base step: the *root* vertex is $(0,0,0) \in \mathbb{V}^*$.

Inductive step 1 (black edge): The admissible destinations of a correction step for stack level E_k^i are determined by the directed edges, and associated scaling relations, of the exemplar graph:

$$\mathcal{A}^*(i,k,t) = \{\,(j,l,t) \mid \exists (i,j,k-l) \in \mathbb{E},\ 0 \le l \le L\,\},$$
$$(i,k,t) \in \mathbb{V}^* \longrightarrow \mathcal{A}^*(i,k,t) \subset \mathbb{V}^*\,.$$

Inductive step 2 (red edge): The upsampling step maps a texel in stack level E_k^i to a texel in stack level E_{k+1}^i, for $k < L$:

$$(i,k,t) \in \mathbb{V}^* \longrightarrow (i,k+1,t+1) \in \mathbb{V}^*\,, \qquad \text{for } 0 \le k < L\,.$$