Image Analogies

Aaron Hertzmann[1,2] *Charles E. Jacobs*[2] *Nuria Oliver*[2] *Brian Curless*[3] *David H. Salesin*[2,3]

[1]New York University [2]Microsoft Research [3]University of Washington

A A' B B'

Figure 1 An image analogy. Our problem is to compute a new "analogous" image B' that relates to B in "the same way" as A' relates to A. Here, A, A', and B are inputs to our algorithm, and B' is the output. The full-size images are shown in Figures 10 and 11.

Abstract

This paper describes a new framework for processing images by example, called "image analogies." The framework involves two stages: a *design phase*, in which a pair of images, with one image purported to be a "filtered" version of the other, is presented as "training data"; and an *application phase*, in which the learned filter is applied to some new target image in order to create an "analogous" filtered result. Image analogies are based on a simple multi-scale autoregression, inspired primarily by recent results in texture synthesis. By choosing different types of source image pairs as input, the framework supports a wide variety of "image filter" effects, including *traditional image filters*, such as blurring or embossing; *improved texture synthesis*, in which some textures are synthesized with higher quality than by previous approaches; *super-resolution*, in which a higher-resolution image is inferred from a low-resolution source; *texture transfer*, in which images are "texturized" with some arbitrary source texture; *artistic filters*, in which various drawing and painting styles are synthesized based on scanned real-world examples; and *texture-by-numbers*, in which realistic scenes, composed of a variety of textures, are created using a simple painting interface.

CR Categories: I.2.6 [Artificial Intelligence]: Learning– Analogies; I.3.3 [Computer Graphics]: Picture/Image Generation– Display algorithms; I.4.10 [Image Processing and Computer Vision]: Image Representation– Statistical; J.5 [Computer Applications]: Arts and Humanities– Fine arts

Additional Keywords: example-based rendering, texture synthesis, non-photorealistic rendering, Markov random fields, autoregression, texture-by-numbers, texture transfer

Please see **http://grail.cs.washington.edu/projects/image-analogies/** for additional information and results.

ACM SIGGRAPH 2001, 12-17 August 2001, Los Angeles, CA, USA

1 Introduction

> **a·nal·o·gy** n. *A systematic comparison between structures that uses properties of and relations between objects of a source structure to infer properties of and relations between objects of a target structure.* [14]
>
> *A native talent for perceiving analogies is . . . the leading fact in genius of every order.*
>
> —William James, 1890 [28]

Analogy is a basic reasoning process, one that we as humans employ quite commonly, and often unconsciously, to solve problems, provide explanations, and make predictions [44]. In this paper, we explore the use of analogy as a means for creating complex image filters (Figure 1). In particular, we attempt to solve the following problem:

> **Problem** ("IMAGE ANALOGIES"): Given a pair of images A and A' (the *unfiltered* and *filtered source images*, respectively), along with some additional *unfiltered target image* B, synthesize a new *filtered target image* B' such that
>
> $$A : A' :: B : B'$$

In other words, we want to find an "analogous" image B' that relates to B in "the same way" as A' relates to A. In general, this is a very difficult problem to solve; in this paper, we describe an approach that works well in many cases.

An advantage of image analogies is that they provide a very natural means of specifying image transformations. Rather than selecting from among myriad different filters and their settings, a user can simply supply an appropriate exemplar (along with a corresponding unfiltered source image) and say, in effect: "Make it look like this." Ideally, image analogies should make it possible to learn very complex and non-linear image filters—for instance, filters that can convert a photograph into various types of artistic renderings having the appearance of oil, watercolor, or pen-and-ink, by analogy with actual (real-life) renderings in these styles. In addition, these various types of filters would not need to be invented individually or programmed explicitly; ideally, the same general mechanism could be used instead to provide this very broad variety of effects.

While image analogies are clearly a desirable goal, it is not so clear how they might be achieved.

For one thing, a crucial aspect of the image analogies problem statement is the definition of similarity used to measure not only the relationship between each unfiltered image and its respective filtered version, but also the relationship between the source pair and the target pair when taken as a whole. This issue is tricky, in that we want to use some metric that is able to preserve recognizable features of the original image filter from A to A', while at the same time is broad enough to be applied to some completely different target image B. Moreover, it is not obvious what features of a training pair constitute the "style" of the filter: in principle, an infinite number of different transformations could be inferred from a pair of images. In this paper, we use a similarity metric that is based on an approximation to a Markov random field model, using raw pixel values and, optionally, steerable filter responses [45]. To measure relationships between the source and target image pair, we sample joint statistics of small neighborhoods within the images, as discussion in Section 3.

In addition, we would like the synthesis of the filtered target image B' to proceed at a reasonable rate. Thus, we will need a way to index and efficiently search over the various images A, A', and B, using the similarity metric, to choose the appropriate parts of the transform $A \rightarrow A'$ in synthesizing $B \rightarrow B'$. We use an autoregression algorithm, based primarily on recent work in texture synthesis by Wei and Levoy [49] and Ashikhmin [2]. Indeed, our approach can be thought of as a combination of these two approaches, along with a generalization to the situation of corresponding *pairs* of images, rather than single textures.

Finally, in order to allow statistics from an image A to be applied to an image B with completely different colors, we sometimes operate in a preprocessed luminance space, as described in detail in Sections 3.3 and 3.4.

In actual usage, we envision image analogies involving two stages. In the *design* (or *training*) phase, a designer (possibly an expert) creates a filter by selecting the training images A and A' (for example, from scanned imagery), annotating the images if desired, and (directly or indirectly) selecting parameters that control how various types of image features will be weighted in the image analogy. The filter can then be stored away in a library. Later, in the *application* phase, a user (possibly someone with no expertise at all in creating image filters) applies the filter to some target image B.

Obviously, we cannot expect our image analogies framework to do a perfect job in learning and simulating all possible image filters, especially from just a single training pair. Moreover, many of the filters that we would like our framework to be able to learn are, in fact, *extremely* difficult even for humans to master. Nevertheless, we have found our image analogies framework to work rather surprisingly well in a variety of situations, as demonstrated in Section 4. These include:

- ***traditional image filters***, such as blurring or "embossing" (Section 4.1);
- ***improved texture synthesis***, in which some textures are synthesized with higher quality than previous approaches (Section 4.2);
- ***super-resolution***, in which a higher-resolution image is inferred from a low-resolution source (Section 4.3);
- ***texture transfer***, in which images are "texturized" with some arbitrary source texture (Section 4.4);
- ***artistic filters***, in which various drawing and painting styles, including oil, watercolor, and line art rendering, are synthesized based on either digitally filtered or scanned real-world examples (Section 4.5); and
- ***texture-by-numbers***, in which realistic scenes, composed of a variety of textures, are created using a simple "painting" interface (Section 4.6).

In all of these cases, producing the various different effects is primarily just a matter of supplying different types of source image pairs as input. For example, a blur filter is "learned" by supplying an image and its blur as the (A, A') pair. Similarly, an oil-painting style is learned by supplying an image and its oil-painted equivalent as the input pair. Ordinary texture synthesis can be viewed as a special case of image analogies in which the unfiltered images A and B are null (i.e., considered to match trivially everywhere), and the analysis/synthesis is performed just on A' and B'. Alternatively, texture-by-numbers is achieved by using a realistic image, such as a landscape, as A' and supplying a simplified, hand-segmented version of the landscape as A—for instance, where one solid color in A corresponds to "sky texture" in A', another to "grass texture," and so on. These same colors can then be painted onto B to generate a new realistic landscape B' with similar textures.

Finally, we also describe a real-time, interactive version of our algorithm, which can be used to provide image analogies underneath the footprint of a digital painting tool (Section 5). While texture-by-numbers is a natural application for this tool, it can be used with any type of image analogy.

While successful in many ways, image analogies do not work in every case since they attempt to model only low-level statistics of the image pairs. Thus, higher-level features such as broad, coherent brush strokes are not always captured or transferred very effectively. Section 6 discusses the limitations of our approach and suggests areas of future research.

2 Related work

Image analogies build upon a great deal of previous work in several disparate areas, including machine learning, texture synthesis, non-photorealistic rendering, and image-based rendering. As far as we know, some of the applications supported by image analogies are completely new, such as the ability to learn "artistic filters" from digitized imagery of real artistic renderings. Others, such as super-resolution and texture transfer, have previously been addressed by other works, in some form or another. Although the algorithms we describe compare favorably—and improve upon—much of this previous work, it is the generality and convenience of the image analogies framework that we believe makes it so interesting and useful. Here, we survey some of the most closely related work.

Generalizing from a set of known examples, as we attempt to do in this paper, is a central problem in machine learning. Analogical reasoning is central to problem solving, learning, and creativity [19, 31, 33]. For this reason, a goal from the early days of artificial intelligence has been to build systems able to reason by analogy; early works include Evan's ANALOGY program [15] and Winston's seminal work on finding and exploiting parallels in simple theories [52]. In this paper, we propose a novel statistical approach for finding analogies between images, from the perspective of modeling transformations that are not just shifts, scales, and rotations but rather mappings of one sort of object or relation into another.

Recently, a number of applications of machine learning to problems in computer graphics have been published, including Video Rewrite [7], Voice Puppetry [5], Video Textures [43], and Style Machines [6]. Our paper continues in the two-word-title tradition of these earlier works. In research that is perhaps most closely related to our own, Freeman *et al.* [17] use Markov random fields (MRFs) for scene learning, in which they attempt to learn the transformation from some captured image to the scene interpretation, such as extracting high-resolution data from low-resolution data ("super-resolution") or inferring optical flow from image pairs. Our method differs from previous MRF modeling techniques in that we do not require an iterative algorithm in order to apply the model. More importantly, our work introduces a variety of new applications of these methods for computer graphics.

In the last few years, a great deal of work has been published in both the computer graphics and computer vision communities on the problem of *texture synthesis*: the creation of images that match the texture appearance of a given digitized sample. Heeger and Bergen [23] introduced this problem to the computer graphics community in 1995. More recently, De Bonet [4] and Efros and Leung [11] showed that a nearest-neighbor search can perform high-quality texture synthesis in a single pass, using multiscale and single-scale neighborhoods, respectively. (This search may be viewed as an approximation to sampling from an MRF, an approach used by Zhu et al. [54] and Portilla and Simoncelli [40].) Wei and Levoy [49] unify these approaches, using neighborhoods consisting of pixels both at the same scale and at coarser scales. Vector quantization [20] or other clustering may be used to summarize and accelerate the nearest-neighbors computation [17, 34, 35, 39, 49].

In unpublished work, Eilhauer *et al.* [13] demonstrate a method for synthesizing texture to match a given image, work that we extend in this paper under the name "texture transfer." In these same proceedings, Efros and Freeman [12] describe improved methods for texture synthesis and show how these methods can also be used for texture transfer. Our texture transfer method also bears some resemblance to Veryovka and Buchanan's methods for halftoning one image with the texture of another [47] and for using multiple textures to illustrate 3D meshes [46].

In recently published work, Ashikhmin [2] describes a texture synthesis method that works by greedily extending existing patches whenever possible, rather than by searching the entire example texture. The algorithm is very fast and produces results that often look much better than the output from previous synthesis methods. However, the greedy search seems to have difficulty restarting effectively when a patch being copied ends (e.g., runs off the end of the image). In our tests, this problem produces abrupt discontinuities between the texture patches. As mentioned earlier, our work builds upon this algorithm, combining it with Wei and Levoy's approach and generalizing this combination to image analogies. Ashikhmin also allows a user to draw a color image as a target for texture synthesis, in a manner that has some resemblance to our "texture-by-numbers" application. However, as discussed in detail in Section 4.6, image analogies are able to address a number of drawbacks in Ashikhmin's approach by virtue of using a pair of images, rather than a single image, to control the synthesis procedure.

One important application of image analogies that we explore in this paper is the automatic synthesis of various artistic styles from examples. In the past few years, there has been a great deal of work in creating artistic styles by computer [10, 21, 24, 26, 32, 36, 41, 42, 50, 51], a field that has come to be known as *non-photorealistic rendering* (NPR). One drawback of most of these previous works, however, is that the methods have had to be specifically tailored to a specific rendering style (or space of styles); image analogies, by contrast, although they may not be able to create any single style as well, can be used for a very broad range of effects.

In this paper, we aim to broaden the range of NPR techniques by exploring what could be thought of as *example-based rendering* (EBR), in which scanned-in artistic imagery is reused for NPR synthesis. The earliest examples of this sort of approach were tools like the "clone brush" in Adobe Photoshop, or the "image hose" in Corel (originally, Fractal Design) Painter. Recently, a number of EBR approaches to NPR have been proposed in a variety of research systems [8, 27, 30, 53]. Perhaps most similar in spirit to our own approach (albeit in somewhat different domains) are a method for creating pen strokes from examples [18] and a method for estimating parameters of a 3D line-drawing illustration system from an example rendering made within the same system [22].

3 Image analogies

Here, we describe a set of data structures and algorithms to support image analogies.

3.1 Definitions and data structures

As input, our algorithm takes a set of three images, the *unfiltered source image* A, the *filtered source image* A', and the *unfiltered target image* B. It produces the *filtered target image* B' as output.

Our approach assumes that the two source images are *registered*; that is, the colors at and around any given pixel p in A correspond to the colors at and around that same pixel p in A', through the image filter that we are trying to learn. Thus, we will use the same index p to specify both a pixel in A and its corresponding pixel in A'. We will use a different index q to specify a pixel in the target pair B and B'.

For the purposes of this exposition, we will assume that the various images contain not just an RGB color, but additional channels of information as well, such as luminance and various filter responses. Together, all of these channels (including RGB) comprise the *feature vector* for each pixel p. We use $A(p)$ (or $A'(p)$) to denote the complete feature vector of A (or A') at pixel p and, similarly, $B(q)$ (or $B'(q)$) to specify the feature vector at pixel q. Note that the features used for the A and B images need not be the same as for the A' and B' images. The particular features we use are described in more detail in Section 3.3 below (however, experimenting with alternate or additional features is certainly a rich area for future research). As we shall see, these features will be used to guide the matching process, in order to help select the most suitable pixels from A' to use in the synthesis of B'.

Finally, our algorithm will need to keep track of the position p of the source pixel that was copied to pixel q of the target. Thus, we will store an additional data structure $s(\cdot)$ (for "source"), which is indexed by q, and has the property $s(q) = p$.

In summary, our algorithm maintains the following data structures, of which the RGB channels of $A(p)$, $A'(p)$, and $B(q)$ are inputs, the RGB channels of $B'(q)$ is the output, and the other channels of A, A', B, and B', as well as $s(q)$, are intermediate computed results in the synthesis process:

$A(p)$: **array** $p \in$ *SourcePoint* **of** *Feature*
$A'(p)$: **array** $p \in$ *SourcePoint* **of** *Feature*$'$
$B(q)$: **array** $q \in$ *TargetPoint* **of** *Feature*
$B'(q)$: **array** $q \in$ *TargetPoint* **of** *Feature*$'$
$s(q)$: **array** $q \in$ *TargetPoint* **of** *SourcePoint*

where *SourcePoint* and *TargetPoint* are 2D pixel locations in the source and target pairs, respectively.

We will actually use a multiscale representation of all five of these quantities in our algorithm. Thus, we will typically index each of these arrays by their multiscale level ℓ using subscripts. For example, if A_ℓ represents the source image A at a given resolution, then $A_{\ell-1}$ represents a corresponding lower-resolution image at the next coarser level, with half as many pixels in each dimension. We will use L to denote the maximum level, i.e., the level for the highest-resolution versions of the images.

3.2 The algorithm

Given this notation, the image analogies algorithm is easy to describe. First, in an initialization phase, multiscale (Gaussian pyramid) representations of A, A', and B is constructed, along with their feature vectors and some additional indices used for speeding the matching process (e.g., an approximate-nearest-neighbor search (ANN), as described below). The synthesis then proceeds from coarsest resolution to finest, computing a multiscale representation of B', one level at a time. At each level ℓ, statistics pertaining

to each pixel q in the target pair are compared against statistics for every pixel p in the source pair, and the "best" match is found. The feature vector $B'_\ell(q)$ is then set to the feature vector $A'_\ell(p)$ for the closest-matching pixel p, and the pixel that matched best is recorded in $s_\ell(q)$.

The algorithm can be described more precisely in pseudocode as follows:

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

The heart of the image analogies algorithm is the BESTMATCH subroutine. This routine takes as input the three complete images A, A' and B, along with the partially synthesized B', the source information s, the level ℓ, and the pixel q being synthesized in B'. It finds the pixel p in the source pair that best matches the pixel being synthesized, using two different approaches: an *approximate search*, which attempts to efficiently find the closest-matching pixel according to the feature vectors of p, q, and their neighborhoods; and a *coherence search*, based on Ashikhmin's approach [2], which attempts to preserve coherence with the neighboring synthesized pixels. In general, the latter approach will usually not return a pixel that matches as closely with respect to the feature vectors; however, since the L_2-norm is an imperfect measure of perceptual similarity, coherent pixels will often look better than the best match under L_2. We therefore rescale the approximate-search distance according to a *coherence parameter* κ, in order to make it artificially larger when comparing the two choices. Thus, the larger the value of κ, the more coherence is favored over accuracy in the synthesized image. In order to keep the coherence term consistent at different scales, we attenuate it by a factor of $2^{\ell-L}$ since pixel locations at coarser scales are spaced further apart than at finer scales. (In a sense, $2^{\ell-L}\kappa$ represents an estimate of the scale of "textons" [29] at level ℓ.) We typically use $2 \le \kappa \le 25$ for color non-photorealistic filters, $\kappa = 1$ for line art filters, and $0.5 \le \kappa \le 5$ for texture synthesis.

Here is a more precise statement of this algorithm:

function BESTMATCH(A, A', B, B', s, ℓ, q):
  $p_{\text{app}} \leftarrow$ BESTAPPROXIMATEMATCH(A, A', B, B', ℓ, q)
  $p_{\text{coh}} \leftarrow$ BESTCOHERENCEMATCH(A, A', B, B', s, ℓ, q)
  $d_{\text{app}} \leftarrow \|F_\ell(p_{\text{app}}) - F_\ell(q)\|^2$
  $d_{\text{coh}} \leftarrow \|F_\ell(p_{\text{coh}}) - F_\ell(q)\|^2$
  if $d_{\text{coh}} \le d_{\text{app}}(1 + 2^{\ell-L}\kappa)$ **then**
    return p_{coh}
  else
    return p_{app}

Here, we use $F_\ell(p)$ to denote the concatenation of *all* the feature vectors within some neighborhood $N(p)$ of both source images A and A' at both the current resolution level ℓ and at the coarser resolution level $\ell - 1$. We have used 5×5 neighborhoods in the fine level and 3×3 neighborhoods in the coarse level (Figure 2). Similarly, we use $F_\ell(q)$ to denote the same concatenation for the target images B and B', although in the case of the filtered target image B' the neighborhood at the finest resolution includes only the portion of the image that has already been synthesized. (Note that $F(\cdot)$ is overloaded in our notation; the index p or q will be used to determine whether a particular $F(\cdot)$ is a source or target neighborhood feature vector.) In each case, the norm $\|F_\ell(p) - F_\ell(q)\|^2$

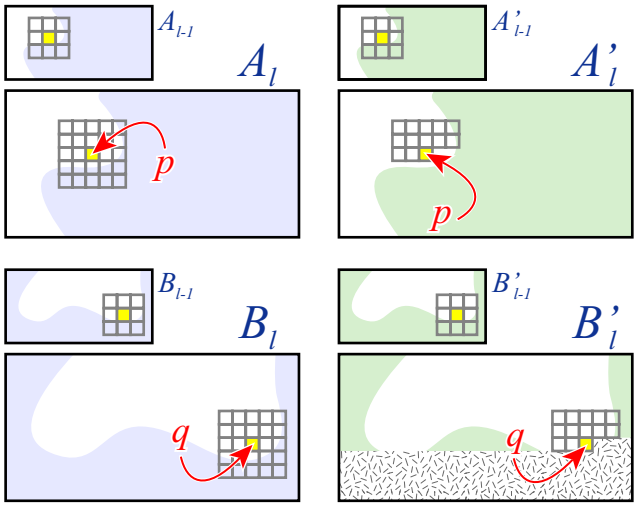


Figure 2 Neighborhood matching. In order to synthesize the pixel value at q in the filtered image B'_ℓ, we consider the set of pixels in B'_ℓ, B_ℓ, $B'_{\ell-1}$, and $B_{\ell-1}$ around q in the four images. We search for the pixel p in the A images that give the closest match. The synthesis proceeds in scan-line ordering in B'_ℓ.

is computed as a weighted distance over the feature vectors $F(p)$ and $F(q)$, using a Gaussian kernel, so that differences in the feature vectors of pixels further from p and q have a smaller weight relative to the differences at p and q. We also normalize the vectors so that each scale of the pyramid has equal weight. Note that some special processing is required at boundaries, as well as at the lowest resolution of the pyramid, since the neighborhoods are a little bit different in these areas. We perform brute-force searches with only the partial neighborhoods in these cases.

For the BESTAPPROXIMATEMATCH procedure, we have tried using both *approximate-nearest-neighbor search* (ANN) [1] and *tree-structured vector quantization* (TSVQ) [20], using the same norm over the feature vectors. In our experience, ANN generally provides more accurate results for the same computation time, although it is also more memory intensive. We used ANN for all of the examples shown in this paper. Principal components analysis (PCA) can be used to reduce the dimensionality of feature vectors leading to a substantial speed-up in the search. We generally keep 99% of the variance, which can lead to a reduction in dimensionality of about an order of magnitude. However, using PCA can degrade the quality of the results on some simple cases; it is most useful in cases with large feature vector sizes (e.g., when steerable filters are used).

The BESTCOHERENCEMATCH procedure simply returns $s(r^\star) + (q - r^\star)$, where

$$r^\star = \arg\min_{r \in N(q)} \|F_\ell(s(r) + (q - r)) - F_\ell(q)\|^2$$

and $N(q)$ is the neighborhood of already synthesized pixels adjacent to q in B'_ℓ. This formula essentially returns the best pixel that is coherent with some already-synthesized portion of B'_ℓ adjacent to q, which is the key insight of Ashikhmin's method.

3.3 Features

Feature selection and representation is a large open problem and an active area of research in machine learning. For now, we have experimented with several different components for the feature vectors. Using the RGB channels themselves is the most obvious choice (and the first thing we tried). However, for some filters, we found that our source pairs did not contain enough data to match the target pair well using RGB color. This is due to the well-known "curse of

dimensionality:" the neighborhood space for RGB images is much larger than for grayscale images, and thus a single image pair provides a correspondingly sparser sampling of the space for RGB than for grayscale. Consequently, the neighborhood histogram of A may still be poorly matched to B, whereas this is less of a problem for grayscale images.

An alternative, which we have used to generate many of the results shown in this paper, is to compute and store the luminance at each pixel and use it in place of RGB in the distance metric. Luminance can be computed in a number of ways; we use the Y channel from the YIQ color space [16], where the I and Q channels are "color difference" components. This approach is motivated by vision science: we are much more sensitive to changes in the luminance channel than to changes in color difference channels [48]. After processing in luminance space, we can recover the color simply by copying the I and Q channels of the input B image into the synthesized B' image, followed by a conversion back to RGB. An added benefit of this approach is the speedup inherent in performing the matching and synthesis with one third as many color channels. The downside, however, is that color dependencies in the analogy filter are lost. In this paper, we worked in luminance space for the blur filter, super-resolution, and artistic filter examples.

Another way to improve the perceptual results of matching is to compute multiple scales of oriented derivative filters [4, 23, 54]. To this end, we can compute a steerable pyramid [45] for the luminance of A and B and concatenate the filter responses to the feature vectors for these images. The distance metric then becomes a weighted combination of similarity in luminance, as well as similarity in orientation among regions of the unfiltered images. We used third-derivative steerable filters comprised of four filter kernels for synthesizing the line art examples (Figure 8); for our other experiments, we found them to make little or no difference.

3.4 Luminance remapping

Converting images to luminance can still give poor overlap between image neighborhood histograms; for example, a light A will be of little use when processing a dark B. As a preconditioning step, we would like to discover a luminance transformation that brings the histograms into correspondence. One standard approach is histogram matching [9]; however, we find that it uses non-smooth mappings with undesirable side-effects.

Our approach, instead, is to apply a linear map that matches the means and variances of the luminance distributions. More concretely, if $Y(p)$ is the luminance of a pixel in image A, then we remap it as

$$Y(p) \leftarrow \frac{\sigma_B}{\sigma_A}(Y(p) - \mu_A) + \mu_B$$

where μ_A and μ_B are the mean luminances, and σ_A and σ_B are the standard deviations of the luminances, both taken with respect to luminance distributions in A and B, respectively. We apply the same linear transform to A', in order to keep the training pair consistent. In this paper, luminance remapping is only used for the color artistic filters (Section 4.5).

This same approach can be extended to matching color distributions in a fairly straightforward way [25]. It is not used in this paper because we found synthesis in luminance to give better results.

4 Applications

By supplying different types of images as input, the image analogies framework can be used for learning filters for many different types of applications. We describe here the applications that we have experimented with so far. Timings for these tests are discussed in the next section. Additional results can be found at **http://grail.cs.washington.edu/projects/image-analogies/**.

4.1 Traditional image filters

As a simple test, we tried learning some traditional image-processing filters, including a "blur" filter (Figure 3) and an "emboss" filter from Adobe Photoshop (Figure 4). While the image-analogies framework gives adequate results, it is nowhere near as efficient as applying the filter directly. Still, these experiments verify that the image analogies framework works for some basic filters.

4.2 Improved texture synthesis

Texture synthesis is a trivial case of image analogies, where the elements of the A and B images are zero-dimensional or constant. The algorithm we have described, when used in this way for texture synthesis, combines the advantages of the weighted L_2 norm and Ashikhmin's search algorithm, although without the speed of Ashikhmin's algorithm. For example, the synthesized textures shown in Figure 5 have a similar high quality to those of Ashikhmin's algorithm, without the edge discontinuities.

4.3 Super-resolution

Image analogies can be used to effectively "hallucinate" more detail in low-resolution images, given some low- and high-resolution pairs (used as A and A') for small portions of the images. (The image analogies framework we have described is easily extended to handle more than a single source image pair, which is what we have done for these examples.) Figure 6 demonstrates this application, using images of a set of maple trees and of a Dobag rug, respectively. An interesting area for future work is to choose the training pairs automatically for image compression, similar to fractal image compression [3].

4.4 Texture transfer

In texture transfer, we filter an image B so that it has the texture of a given example texture A' (Figure 7). Texture transfer is achieved by using the same texture for both A and A'. We can trade off the appearance between that of the unfiltered image B and that of the texture A by introducing a weight w into the distance metric that emphasizes similarity of the (A, B) pair over that of the (A', B') pair. Increasing w causes the input image to be reproduced more faithfully, whereas decreasing w ties the image more closely to the texture. When $w = 0$, texture transfer reduces to ordinary texture synthesis. For somewhat better results, we also modify the neighborhood matching by using single-scale 1×1 neighborhoods in the A and B images. Thus, $F_\ell(p)$ contains the value of the pixel p in A_ℓ (or B_ℓ), as well as the usual neighborhoods around p in A'_ℓ and $A'_{\ell-1}$ (or B'_ℓ and $B'_{\ell-1}$).

This application of image analogies may be somewhat counterintuitive, since an intuitive interpretation of an "analogy" in which A is the same as A' is as the identity filter. However, texture transfer is actually another valid interpretation [25]. Image analogies synthesize images drawn from the statistical distribution of neighborhoods in A'—in texture transfer, this is done while trying to match the B image as closely as possible.

4.5 Artistic filters

Although the problem is in general very difficult, we have had some success in using image analogies to transfer various artistic styles from one image to another, as shown in Figures 8–13 and 15.

For many example images, we do not have a source photograph available; hence, a substitute must be created. We generally view the A' image as providing texture (e.g., pen strokes or paint texture) to an untextured image. To create A from A', we apply an anisotropic diffusion [37] or similar filter to A' (we used the "Smart Blur" filter from Adobe Photoshop), in order to maintain sharp contours but eliminate texture. For line art filters, we blur the image before applying anisotropic diffusion.

For the color artistic filters in this paper, we performed synthesis in luminance space, using the preprocessing described in Sections 3.3 and 3.4. (In general, we find that matching with color gives richer and more appealing results, but can often fail quite dramatically.)

For line art filters, using steerable filter responses in feature vectors leads to significant improvement. We suspect that this is because line art depends significantly on gradient directions in the input images. We use steerable filter responses only for matching in A and B (but not in A'/B').

We have also had some success when A is a known photograph and A' is a painting or drawing made by hand from the photograph. (Unfortunately, these results are not shown here, for copyright reasons.) In these cases, some care must be taken to carefully register the photograph to the painting since our algorithm currently assumes that the images are in approximate pointwise correspondence. For our training pairs of this type, we first aligned the images by manually estimating a global translation, rotation, and scale. We then warped the example source image with a custom image warping program designed for local image adjustments [25].

The scale of the training images determines the fineness of the features in the B' image and may be chosen by the filter designer, or left as a parameter to the user applying the filter.

4.6 Texture-by-numbers

Texture-by-numbers allows new imagery to be synthesized by applying the statistics of a labeled example image to a new labeling image B. For example, given a labeling of the component textures of a realistic image, a new realistic one may be painted just by painting the arrangement of the component textures (Figure 14). An example of this kind of synthesis, performed in real-time as part of a "painting" interface, is demonstrated in the accompanying video.

A major advantage of texture-by-numbers is that it allows us to synthesize from images for which ordinary texture synthesis would produce poor results. Consider the photograph of an oxbow shown in Figure 14. Although the image has textural regions, attempting to create a new version of the river via ordinary texture synthesis would produce very poor results, as the synthesis process would mix unrelated textures. In statistical terms, the problem is that the texture distribution is not stationary. On the other hand, specifying a corresponding A image makes the A' image into a useful texture, in the sense that the *conditional* density of the A' image is stationary (now conditioned on A). Given a new B image, we can generate a new scene to match it. Note that, in addition to filling in textures in a sensible manner, the boundaries between texture regions also match the examples, since they are synthesized from examples in A with similar boundary shapes.

A more sophisticated example is shown in Figure 16. Treating the scenery as a single texture produces poor results because the synthesis mixes foreground texture with background texture. In other words, the texture is stationary horizontally but not vertically. In this case, we provide a gradient in the red channel of the A image, which constrains the synthesis so that near elements will not be mixed with far elements.

Texture-by-numbers requires an appropriate choice of the A image in order to factor out non-stationary components of A'. In our experience, the synthesis is somewhat forgiving, degrading gracefully as the assumptions become less appropriate. In principle, the A image can be of arbitrary dimension and content. For example, it could include additional information about normals, depths, or orientations [42] of the A' image to improve the texture-by-numbers process.

This application bears some resemblance to the user-guided texturing described by Ashikhmin [2]; however, it fixes several of the problems with that method. In Ashikhmin's method, multiple passes are usually required for a good match. In addition, Ashikhmin's greedy search may create poor matches when a very large example texture is used, since the synthesis cannot "restart" until it finishes copying a patch. More significantly, the colors in the target must be distinct: the algorithm would have difficulty, for example, distinguishing between green trees and green grass. In addition, our algorithm also allows for extra channels of information (such as depth, normals, etc.) to be used to control the synthesis.

5 Interactive editing

For many applications, the ability to directly manipulate an image via a user interface is crucial. In the accompanying video, we demonstrate an application in which a user can "paint" a landscape by coarsely specifying locations for the trees, sky, etc. The main difficulty is that a single change to a B image could theoretically affect the rest of the image, and the full synthesis algorithm is currently too slow to run at interactive rates. However, we can exploit the fact that, in practice, user painting operations only affect a small area of the image at a time, and, under the locality assumption, these operations will have exponentially-decaying influence on distant image pixels. Hence, we can maintain an acceptable image by updating only the modified pixels and their neighbors.

The user interface presents a painting interface for placing RGB values into the B or B' images. The B image is initialized with some default value (e.g., a blank image or a predefined image), and a corresponding B' image is synthesized from the initial B'. The initial B' may also be precomputed.

The key to making interactive painting efficient in this context is to provide an immediate update of B' using a coherence search (Section 3.2) as the user paints, and to refine B' with the full search (approximate plus coherence search) progressively, only as processing cycles allow. Our implementation has two threads, an event handler and a synthesis thread. When the user paints into the B image, the event handler queues the painting locations at all scales for updating. The synthesis thread performs a coherence search on the changed pixels in scan-line order, though it uses the full search with causal neighborhoods for every tenth pixel. Pixels that have not been updated are marked as pixels to be ignored when comparing neighborhood distances during coherence search. These pixels are placed in another queue, and, whenever the first queue is empty, the update thread performs full search with non-causal neighborhoods on the contents of the second queue.

6 Discussion and future work

In this paper, we have described a framework for image processing by example, which generalizes texture synthesis for the case of two corresponding image pairs. We have shown how the framework described is applicable to a wide variety of image texturing problems, including image filtering, texture synthesis, super-resolution, texture transfer, artistic filters, and texture-by-numbers. As the framework is very general, we suspect that it (or some related framework, based on the same kinds of analogic reasoning) may eventually be applicable to a much broader domain of problems, perhaps extending well outside of image processing—or even image synthesis—to encompass such disparate varieties of data as motion capture, music synthesis, and so on. Thus, we are intrigued by the many possibilities for future research!

There is still much work to be done, both in improving the methods that we have presented, and in investigating other approaches. Here is a partial list of some of the directions we would like to pursue:

Speeding it up. The performance of our algorithm is logarithmic with respect to the size (in pixels) of the training pair (due to using heuristic search techniques), and linear with respect to the size of the target. Although we have made use of several techniques to enhance performance (e.g., PCA, ANN), our algorithm is still rather slow, taking anywhere from on the order of tens of seconds to do simple texture synthesis, to a few minutes for the texture transfer

examples, to a few hours for the artistic renderings on a 1GHz PC processor. One reason for this fairly poor performance is that our implementation has never been hand-tuned, since we have been interested more in ease of prototyping than in speed. We expect that optimizing the implementation would give about a factor of 5 speed-up. In addition, we would like to explore better regression and search techniques that could speed up the algorithm even more.

Estimating other types of image statistics. Capturing the style of a filter from an example is a fundamentally difficult problem because the solution is not unique and often depends on prior knowledge that may be difficult to implement. Our approach attempts to capture low-level statistical features of the filter. Unfortunately, many artistic styles involve larger-scale features that our approach does not capture so well. More specialized image analogies methods could be tailored to specific applications. For example, it would be interesting to try to learn NPR filters by estimating statistics of stroke shapes and positions; however, this likely entails difficult combinatorial optimization problems, as well as the problem of acquiring training data.

Better features and similarity metrics. None of the neighborhood distance metrics that we use provide perfect measures of perceptual similarity, since each one fails in a variety of situations. Our algorithm, which attempts to combine the best attributes of the L_2 metric with those of Ashikhmin's algorithm, still produces some undesirable artifacts. Finding a better matching function and/or better features is an important open area of research for image analysis and synthesis applications.

Better color processing. In order to achieve good results for small training sets, we have sacrificed the full use of color information by using luminance space for some filters. There are many possible ways to improve color processing. A simple improvement would be to generate the I and Q channels of B' by fitting a linear transformation from the I and Q channels of A to A' (similar to our luminance matching step), and then applying this transform to the I and Q channels of B. Another approach would be to improve preprocessing to enable synthesis in full color, such as by allowing different linear transformations for different image patches, or by fitting a smooth, non-linear histogram matching function.

Learning image correspondence and registration. Another limitation of our technique is that it assumes a pointwise correspondence between the training images, which requires careful registration of the example images. It would be more desirable to learn the filter, while simultaneously learning the correspondence between pixels, and even automatically apply a learned distortion during synthesis.

Combining with automatic texture segmentation. Our work generalizes recent results in texture synthesis. Whereas texture synthesis algorithms can only resynthesize images from stationary statistics, our methods can resynthesize non-stationary statistics based on some user guidance. One possible extension of this work would be to synthesize from non-stationary images in a completely automatic fashion, perhaps using a texture segmentation as an input [35].

Extensions to 3D models and animation. An important extension is to processing images of 3D models, where additional channels of information (e.g., depths, normals, and silhouettes) can be used to improve the filter reproduction. Likewise, there are natural extensions of these ideas to creating video and animation.

Extensions to other domains. We are interested in applying the image analogy algorithm to a number of other filter learning problems, such as scratch removal, alpha estimation, motion synthesis, musical synthesis, and so on. Moreover, it should be possible to generalize the framework we have described to make use of analogies between data in completely different domains. As a whimsical example, A might be an image and A' an interpretive dance based on the image; a new image B could then be used to synthesize a new interpretive dance B'. Several problems would need to be solved to make this possible. For example, the framework we have described requires a a one-to-one correspondence between data in the training pair, and no such obvious correspondence would exist in this case. Still, we are encouraged by our early results, and look forward to improving these methods and exploring new applications of analogic reasoning in computer graphics.

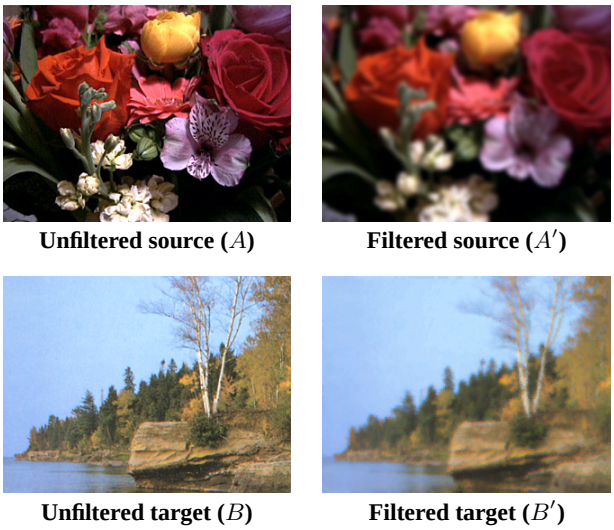


Figure 3 Toy example: Learning a blur filter. The A and A' images comprise the training data. Images were converted to luminance, and then the filter learned from A and A' was applied to B to get B'. (Shore image courtesy John Shaw [38].)

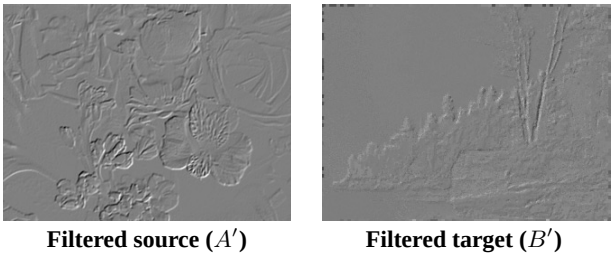


Figure 4 Toy example: Learning an emboss filter. The A and B images are the same as in Figure 3.

Acknowledgements

We are indebted to Douglas Zongker and Wilmot Li for their generous assistance with creating the video and figures for this paper. We are grateful to Matthew Brand for suggesting that we look at histograms, and to Shlomo Gortler for showing us his students' work on texture transfer [13]. Stephen Spencer provided indispensible technical assistance during the final phases of this work. This work was supported in part by NSF grants DGE-9454173 and CCR-9875365, and by an Intel equipment donation.

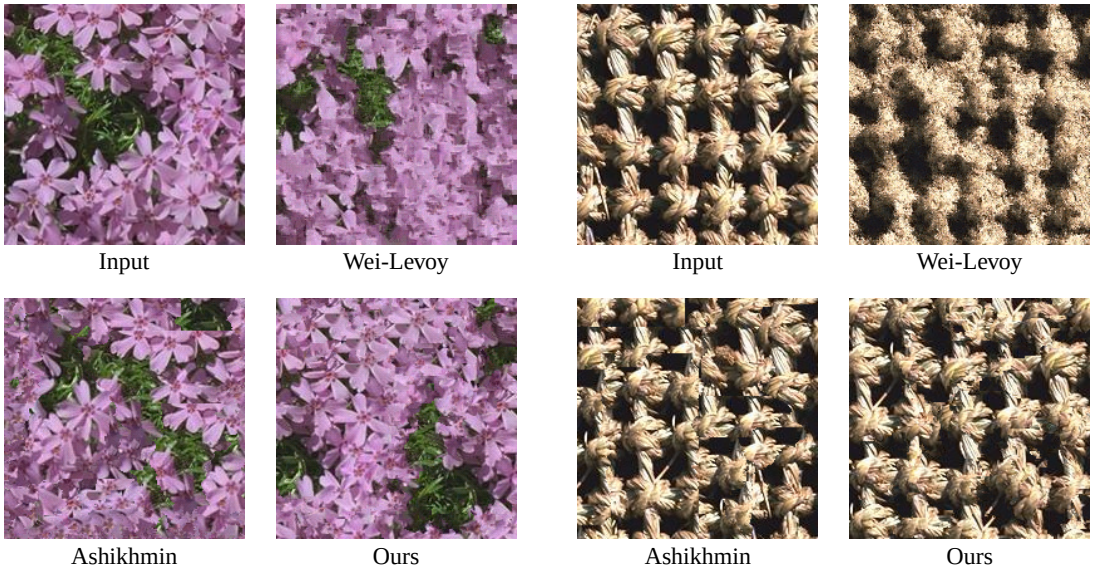


Figure 5 Improved texture synthesis. The textures synthesized with Wei and Levoy's algorithm [49] give blurry results because the L_2 norm is a poor measure of perceptual similarity. Ashikhmin's algorithm [2] gives high-quality coherent patches, but creates horizontal edges when patches reach the end of the source image, such as in the upper right corner of the flower texture. Additionally, Ashikhmin's algorithm does not capture appearance at multiple scales, such as the regular pattern of the weave. Our algorithm combines the advantages of these two previous methods. We used $\kappa = 5$ for both textures in this figure. (The input textures were obtained from the MIT VisTex web page, Copyright © 1995 MIT. All rights reserved).

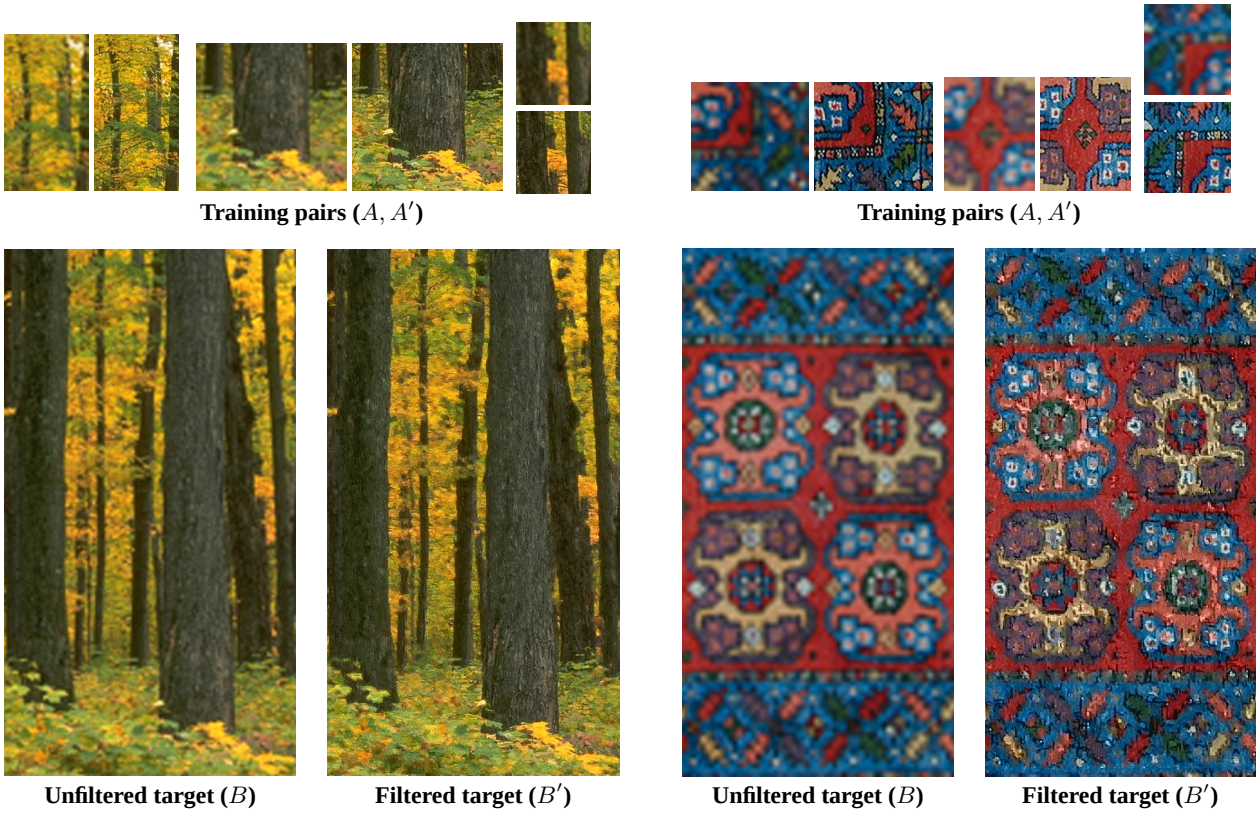


Figure 6 Super-resolution. For each example, the training pairs (above) contain low- and high-resolution versions of a portion of an image. This training data is used to specify a "super-resolution" filter that is applied to a blurred version of the full image (below, left) to recover an approximation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

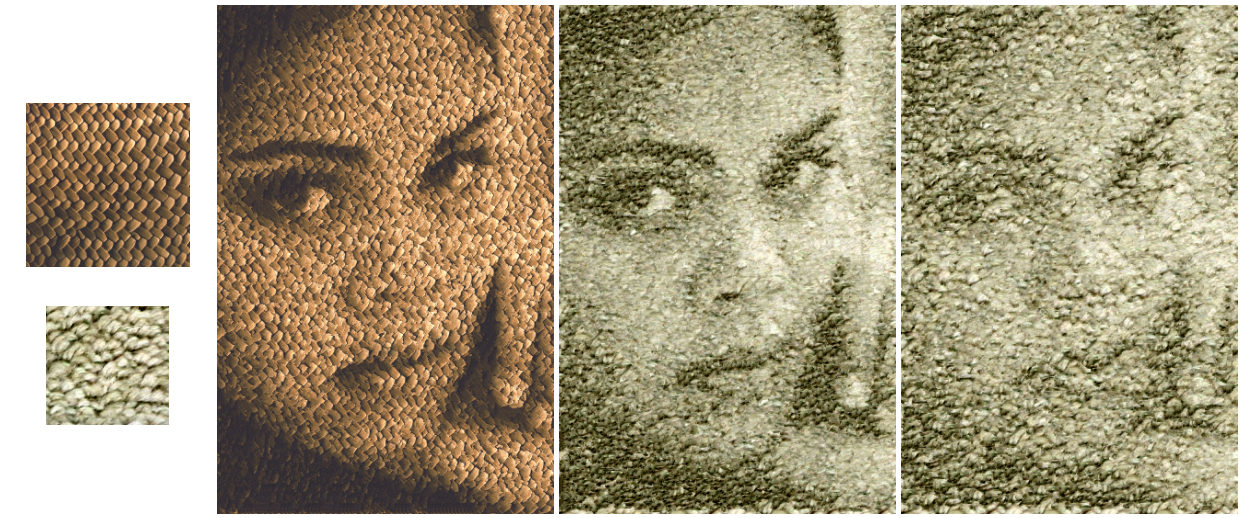


Figure 7 Texture transfer. A photograph (shown in Figure 8) is processed to have the weave and rug textures shown on the left. In each case, the texture is used for both A and A'. The center and right result images show the effect of trading off fidelity to the source image versus fidelity to the texture. (The weave texture was obtained from the MIT VisTex web page, Copyright © 1995 MIT. All rights reserved).

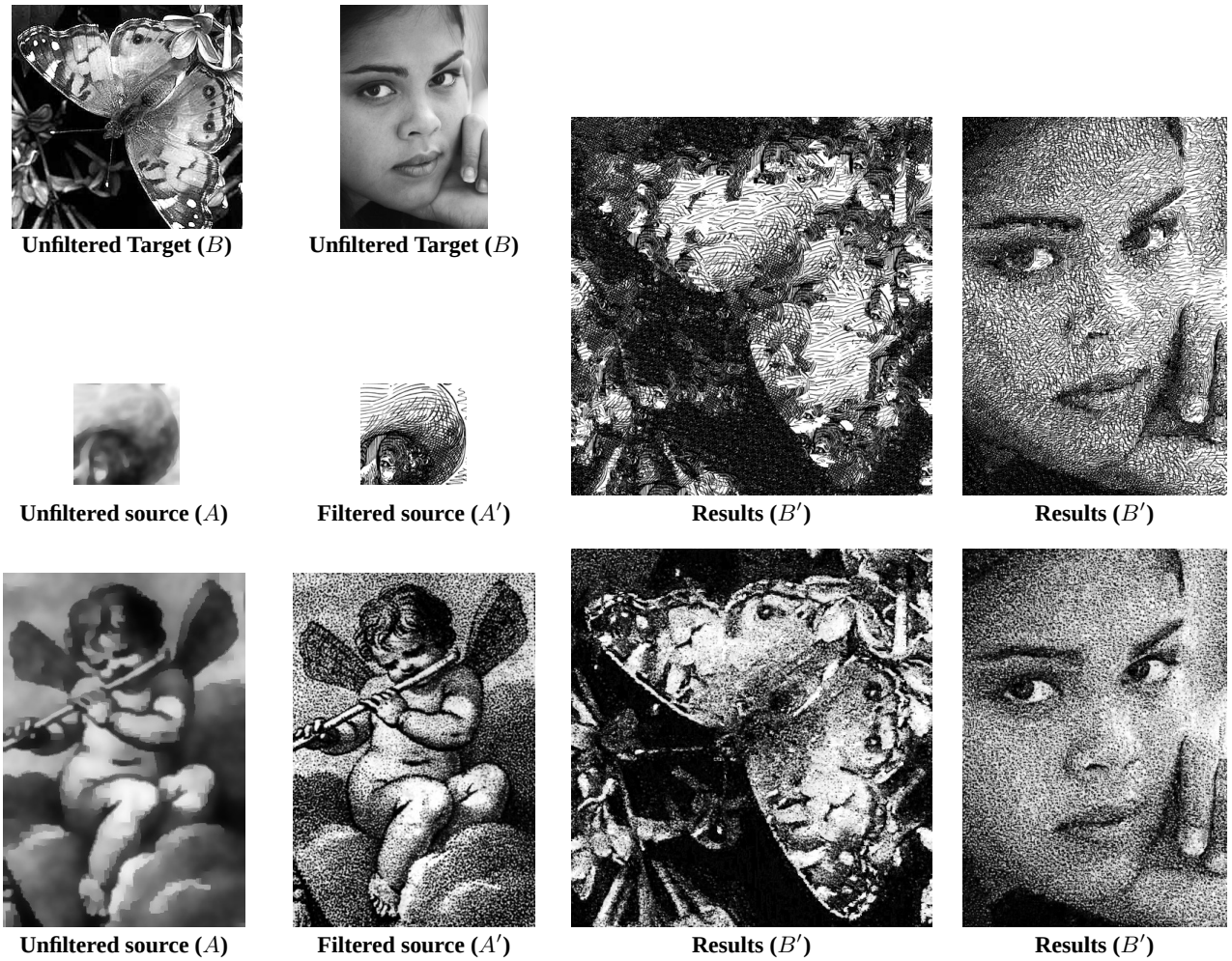


Figure 8 Line art illustrations by example. Two illustration styles, defined by the hand-drawn A' images, are applied to the two B images shown (at reduced size) in the upper left. Each A image was created by applying a blur and an anisotropic diffusion to the corresponding A'. The resulting four B' images are shown on the right. The upper A' image is from Gustave Doré's illustrations for *Don Quixote*; the lower A' is from an engraving by Francesco Bartolozzi, a Renaissance artist.

Figure 9 Unfiltered target images (B) for the NPR filters and texture transfer. (Leftmost image courtesy John Shaw [38].)

Unfiltered examples (A) **Filtered examples (A')**

Figure 10 Training pairs for the color NPR filters used in this paper. The upper A' image is a detail of *Starry Night above the Rhône* by Vincent Van Gogh; the "unfiltered" source image was generated by processing the painting with Photoshop's "Smart Blur" filter. The lower image pair is a photograph and a watercolor created from it with a semi-automatic digital filter [10].

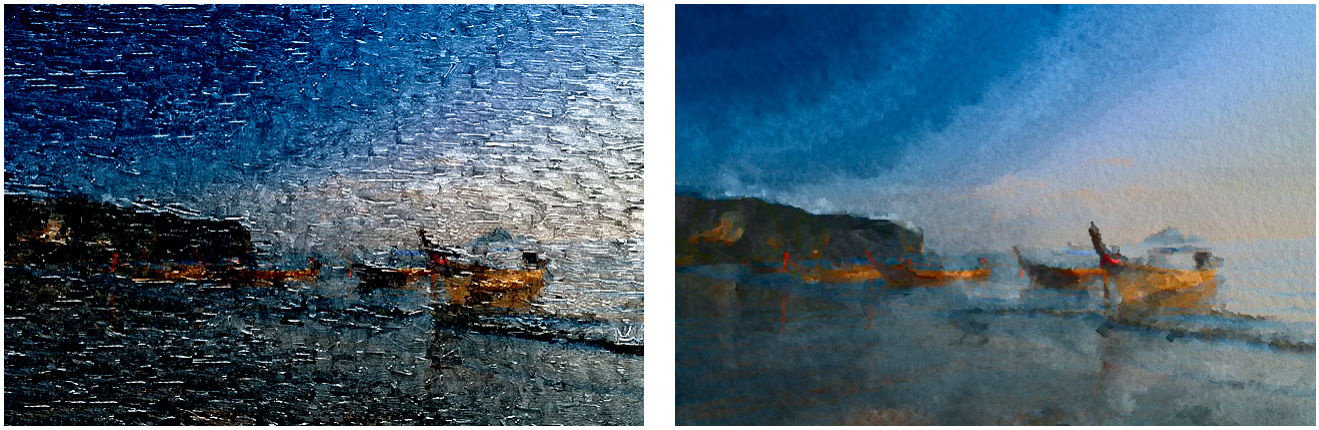
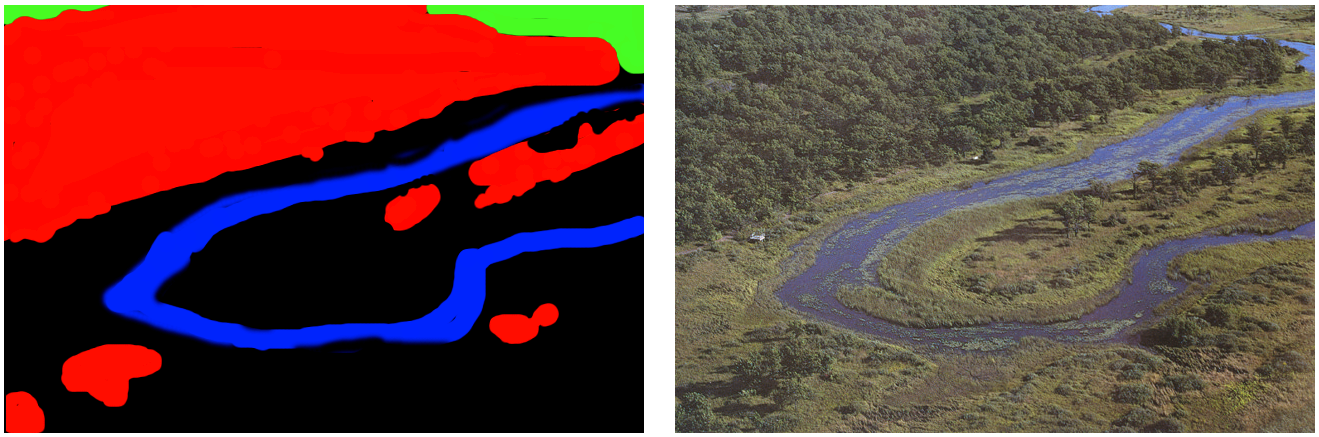


Figure 11 Boat paintings by example. The left image is painted in a style learned from a Van Gogh painting (Figure 10, top row); the right image is in the style of a watercolor filter (Figure 10, bottom row).

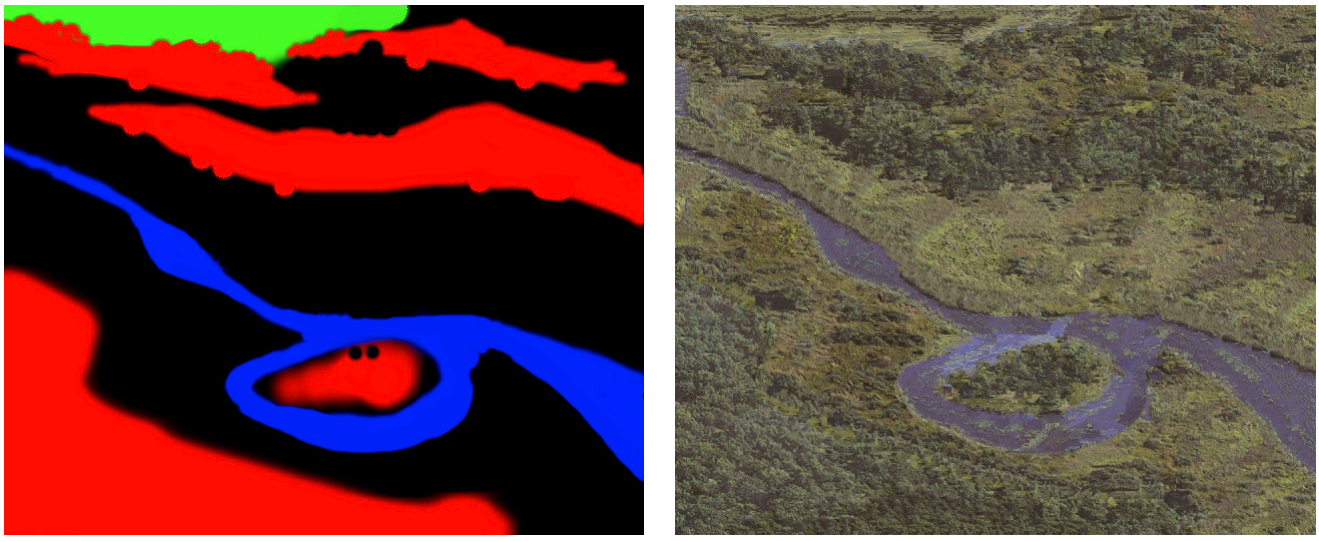
Figure 12 Paintings by example. The left images are painted in a style learned from a Van Gogh painting (Figure 10, top row); the right images are in the style of a watercolor filter (Figure 10, bottom row).

Unfiltered source (A) **Filtered source (A')** **Filtered (B', $\kappa = 5$)**

Figure 13 Paintings by example, varying the coherence parameter κ. The filtered target image (A') is a detail of *Still Life with Melon and Peaches* by Edouard Manet. Additional images are shown on the facing page in Figure 15. The source image is shown in Figure 9.



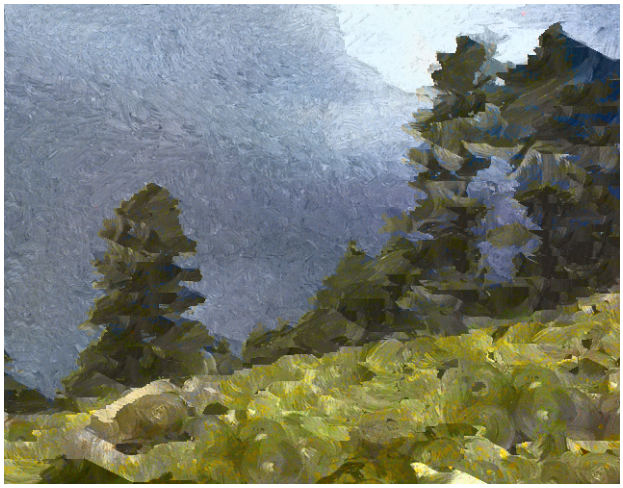
Unfiltered source (A) **Filtered source (A')**



Unfiltered (B) **Filtered (B')**

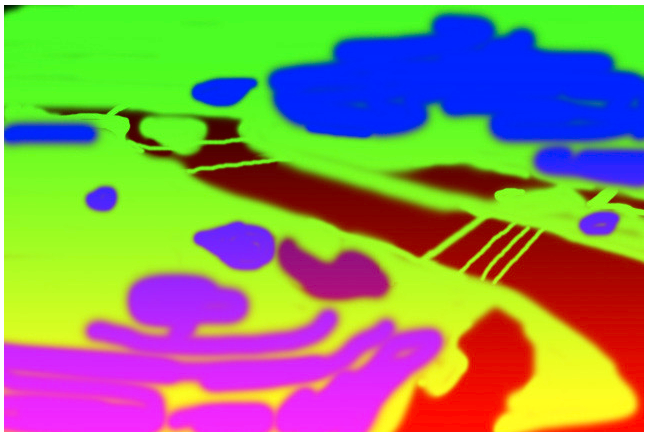
Figure 14 Texture-by-numbers. The unfiltered source image (A) was painted by hand to annotate A'. The unfiltered target image (B) was created in a paint program and refined with our interactive editor; the result is shown in B'. (Oxbow image courtesy John Shaw [38].)

Filtered (B', $\kappa = 10$)



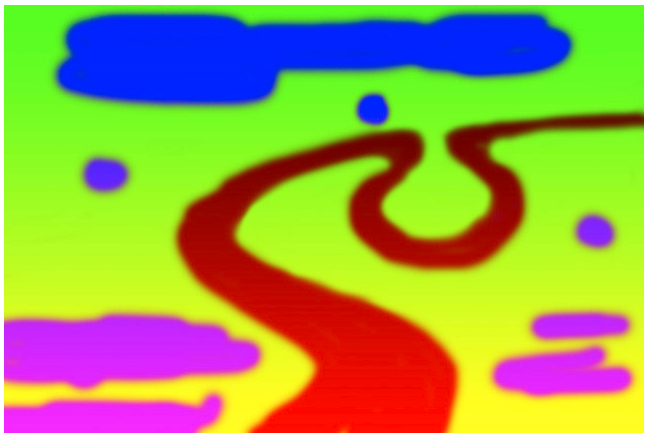
Filtered (B', $\kappa = 15$)

Figure 15 Paintings by example, varying the coherence parameter κ. The training is from a painting by Manet. Additional images are shown on the facing page in Figure 13. The source image is shown in Figure 9.



Unfiltered source (A)

Filtered source (A')



Unfiltered (B)

Filtered (B')

Figure 16 Rerouting the Potomac. Ordinary texture synthesis cannot reproduce the terrain in the photograph because it is not stationary: far elements are different from near elements. The use of the gradient channel in A and B distinguishes near from far, allowing the photograph to be used for texture-by-numbers.

References

[1] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. *Journal of the ACM*, 45(6):891–923, 1998. Source code available from http://www.cs.umd.edu/~mount/ANN.

[2] Michael Ashikhmin. Synthesizing Natural Textures. *2001 ACM Symposium on Interactive 3D Graphics*, pages 217–226, March 2001.

[3] Michael F. Barnsley, Lyman P. Hurd, and Louisa F. Anson. *Fractal Image Compression*. A.K. Peters Ltd, 1993.

[4] Jeremy S. De Bonet. Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images. *Proceedings of SIGGRAPH 97*, pages 361–368, August 1997.

[5] Matthew Brand. Voice Puppetry. *Proceedings of SIGGRAPH 99*, pages 21–28, August 1999.

[6] Matthew Brand and Aaron Hertzmann. Style machines. *Proceedings of SIGGRAPH 2000*, pages 183–192, July 2000.

[7] Christoph Bregler, Michele Covell, and Malcolm Slaney. Video Rewrite: Driving Visual Speech with Audio. *Proceedings of SIGGRAPH 97*, pages 353–360, August 1997.

[8] Ian Buck, Adam Finkelstein, Charles Jacobs, Allison Klein, David H. Salesin, Joshua Seims, Richard Szeliski, and Kentaro Toyama. Performance-driven hand-drawn animation. *NPAR 2000: First International Symposium on Non Photorealistic Animation and Rendering*, pages 101–108, June 2000.

[9] Kenneth Castleman. *Digital Image Processing*. Prentice-Hall, 1996.

[10] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer, and David H. Salesin. Computer-Generated Watercolor. *Proceedings of SIGGRAPH 97*, pages 421–430, August 1997.

[11] Alexei Efros and Thomas Leung. Texture Synthesis by Non-parametric Sampling. *7th IEEE International Conference on Computer Vision*, 1999.

[12] Alexei A. Efros and William T. Freeman. Quilting for Texture Synthesis and Transfer. *Proceedings of SIGGRAPH 2001*, August 2001.

[13] Alex Eilhauer, Alice Pritikin, Dylan Weed, and Steven J. Gortler. Combining Textures and Pictures with Specialized Texture Synthesis, 2000. http://www.people.fas.harvard.edu/∼pritikin/cs/graphics/.

[14] Chris Eliasmith. Dictionary of Philosophy of Mind. http://artsci.wustl.edu/∼philos/MindDict/.

[15] T.G. Evans. A program for the solution of geometric analogy intelligence test questions. In M. Minsky, editor, *Semantic Information Processing*. MIT Press, 1968.

[16] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, 1990.

[17] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael. Learning Low-Level Vision. *Intl. J. Computer Vision*, 40(1):25–47, 2000. See also http://www.merl.com/reports/TR2000-05/.

[18] William T. Freeman, Joshua B. Tenenbaum, and Egon Pasztor. An example-based approach to style translation for line drawings. Technical Report TR99-11, MERL, February 1999.

[19] D. Gentner. Structure mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983.

[20] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[21] Paul E. Haeberli. Paint By Numbers: Abstract Image Representations. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 207–214, August 1990.

[22] J. Hamel and T. Strothotte. Capturing and re-using rendition styles for non-photorealistic rendering. *Computer Graphics Forum*, 18(3):173–182, September 1999.

[23] David J. Heeger and James R. Bergen. Pyramid-Based Texture Analysis/Synthesis. *Proceedings of SIGGRAPH 95*, pages 229–238, August 1995.

[24] Aaron Hertzmann. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In *SIGGRAPH 98 Conference Proceedings*, pages 453–460, July 1998.

[25] Aaron Hertzmann. *Algorithms for Rendering in Artistic Styles*. PhD thesis, New York University, May 2001.

[26] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. *Proceedings of SIGGRAPH 2000*, pages 517–526, July 2000.

[27] Youichi Horry, Ken ichi Anjyo, and Kiyoshi Arai. Tour Into the Picture: Using a Spidery Mesh Interface to Make Animation from a Single Image. *Proceedings of SIGGRAPH 97*, pages 225–232, August 1997.

[28] William James. *The Principles of Psychology*. 1890.

[29] Bela Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290:91–97, 1981.

[30] Allison W. Klein, Wilmot W. Li, Michael M. Kazhdan, Wagner T. Corrêa, Adam Finkelstein, and Thomas A. Funkhouser. Non-photorealistic virtual environments. *Proceedings of SIGGRAPH 2000*, pages 527–534, July 2000.

[31] Arthur Koestler. *The Act of Creation*. Picador, London, 1964.

[32] Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John Hughes. Art-Based Rendering of Fur, Grass, and Trees. *Proceedings of SIGGRAPH 99*, pages 433–438, August 1999.

[33] G. Lakoff and M. Johnson. *Metaphors we live by*. University of Chicago Press, Chicago, IL, 1980.

[34] Thomas Leung and Jitendra Malik. Recognizing surfaces using three-dimensional textons. *7th IEEE International Conference on Computer Vision*, September 1999.

[35] Jitendra Malik, Serge Belongie, Jianbo Shi, and Thomas Leung. Textons, Contours, and Regions: Cue Integration in Image Segmentation. *7th IEEE International Conference on Computer Vision*, September 1999.

[36] Barbara J. Meier. Painterly Rendering for Animation. In *SIGGRAPH 96 Conference Proceedings*, pages 477–484, August 1996.

[37] Pietro Perona and Jitendra Malik. Scale-Space and Edge Detection using Anisotropic Diffusion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12:629–639, December 1990.

[38] Ferdinand Petrie and John Shaw. *The Big Book of Painting Nature in Watercolor*. Watson-Guptill Publications, 1990.

[39] Kris Popat and Rosalind W. Picard. Cluster-based probability model and its application to image and texture processing. *IEEE Trans. on Image Processing*, 6(2):268–284, February 1997.

[40] J. Portilla and E. P. Simoncelli. A Parametric Texture Model based on Joint Statistics of Complex Wavelet Coefficients. *International Journal of Computer Vision*, 40(1):49–71, December 2000.

[41] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive Pen–And–Ink Illustration. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 101–108, July 1994.

[42] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *SIGGRAPH 97 Conference Proceedings*, pages 401–406, August 1997.

[43] Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video Textures. *Proceedings of SIGGRAPH 2000*, pages 489–498, July 2000.

[44] K. Schunn and K. Dunbar. Priming, Analogy and Awareness in complex reasoning. *Memory and Cognition*, 24:271–284, 1996.

[45] Eero P. Simoncelli and William T. Freeman. The Steerable Pyramid: A Flexible Architecture for Multi-Scale Derivative Computation. *Proc. 2nd Int'l Conf on Image Processing*, October 1995.

[46] Oleg Veryovka and John W. Buchanan. Comprehensive Halftoning of 3D Scenes. *Computer Graphics Forum*, 18(3):13–22, September 1999.

[47] Oleg Veryovka and John W. Buchanan. Halftoning With Image-Based Dither Screens. *Graphics Interface '99*, pages 167–174, June 1999.

[48] B. Wandell. *Foundations of Vision*. Sinauer Associates Inc., 1995.

[49] Li-Yi Wei and Marc Levoy. Fast Texture Synthesis Using Tree-Structured Vector Quantization. *Proceedings of SIGGRAPH 2000*, pages 479–488, July 2000.

[50] Georges Winkenbach and David H. Salesin. Computer–Generated Pen–And–Ink Illustration. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 91–100, July 1994.

[51] Georges Winkenbach and David H. Salesin. Rendering Parametric Surfaces in Pen and Ink. In *SIGGRAPH 96 Conference Proceedings*, pages 469–476, August 1996.

[52] P.H. Winston. Learning and Reasoning by Analogy. *Communications of the ACM*, (23) 12, December 1980.

[53] Daniel N. Wood, Adam Finkelstein, John F. Hughes, Craig E. Thayer, and David H. Salesin. Multiperspective panoramas for cel animation. *Proceedings of SIGGRAPH 97*, pages 243–250, August 1997.

[54] Song Chun Zhu, Ying Nian Wu, and David Mumford. Filters, Random fields, And Maximum Entropy: Towards a Unified Theory for Texture Modeling. *International Journal of Computer Vision*, 12(2):1–20, March/April 1998.