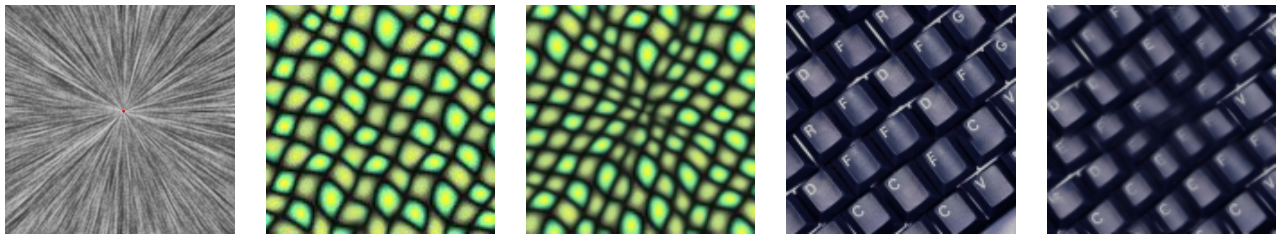
Texture Optimization for Example-based Synthesis

Vivek Kwatra Irfan Essa Aaron Bobick Nipun Kwatra

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



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

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

Abstract

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

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—color, shading, shadowing, and texture.

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

1 Introduction

Synthesis of novel photo-realistic imagery from limited example input is of wide importance in computer graphics. Many example-based synthesis approaches rely on the presence of texture. Texture refers to the class of imagery that can be categorized as a portion of an infinite pattern consisting of stochastically repeating elements. This inherent repeatability present in textures is the key behind texture synthesis techniques. These techniques generate output textures that are larger in size than the input sample but perceptually similar to it. Researchers have also extended these techniques to allow for controllable synthesis in the presence of constraints.

Texture synthesis techniques can be broadly categorized into local region-growing methods and global optimization-based methods. Local methods grow the texture one pixel or patch at a time with the goal of maintaining coherence of the grown region with nearby pixels [Efros and Leung 1999; Wei and Levoy 2000; Efros and Freeman 2001]. In such approaches, small errors can accumulate over large distances leading to inconsistencies in the synthesized texture. On the other hand, global methods evolve the entire texture as a whole, based on some criteria for evaluating similarity with the input. Most existing global approaches either model only pixel-to-pixel interactions that may be insufficient to capture large scale structures of the texture [Heeger and Bergen 1995; Paget and Longstaff 1998], or lead to complex formulations that are difficult to optimize [Portilla and Simoncelli 2000; Freeman et al. 2002].

We present an approach for texture synthesis that is based on optimization of texture quality with respect to a similarity metric. This similarity metric is motivated by the Markov Random Field (MRF)-based similarity criterion used in most local pixel-based synthesis techniques. Our contribution is to merge these locally defined similarity measures into a global metric that can be used to jointly optimize the entire texture. This global metric allows modeling of interactions between large neighborhoods; nevertheless, it can be optimized using a simple iterative algorithm with reasonable computational cost.

The MRF property of textures requires that *locality* and *stationarity* be satisfied. Locality implies that the color at a pixel's location is dependent only on a neighborhood of pixels around it, while stationarity means that this dependency is independent of the actual location of the pixel. Exploiting this property, one can measure *energy* of the synthesized texture with respect to the input texture by comparing local neighborhoods in the two textures. The energy of a single synthesized neighborhood is defined as *its distance to the closest neighborhood in the input*. The total energy of the synthe-

sized texture is then equal to the sum of energies over individual neighborhoods. Texture synthesis proceeds by optimizing this *texture energy* using an iterative algorithm similar to the well-known Expectation Maximization (EM) algorithm [McLachlan and Krishnan 1997]. This optimization procedure improves the entire texture through successive iterations of the algorithm. This form of progressive refinement of textures can potentially be very useful in situations that demand fast or real-time computations with level-of-detail aspects, *e.g.,* video games.

A unique aspect of our technique is that it is intermediate between pixel and patch based methods. The neighborhood size used to define texture energy determines the granularity at which synthesis is performed. We begin synthesis using large neighborhood sizes ($\sim 32 \times 32$ pixels), which allows large scale elements of the texture to settle coherently in the output. We then further refine the texture using smaller neighborhood sizes (16×16, 8×8, etc). The use of large neighborhoods gives our technique a patch-based flavor, but these patches are not merely copied over. Instead each pixel value is allowed to deviate differently from the input patch depending upon weights and overlaps, giving it a pixel-based flavor.

A significant feature of our optimization-based approach is that it directly supports controllable synthesis of textures. This is achieved by adding a simple extension that augments the texture energy with a *control energy* term, which measures the consistency of the synthesized texture with the control criteria. We specifically demonstrate controllability by synthesizing *texture sequences* that follow a given flow field. The flow field must be a two-dimensional vector field that may be dynamically changing over time. Many textures like water, fire, smoke, etc, are visual manifestations of phenomena that can be physically described as fluid flow. One can envision using our technique as a rendering mechanism in conjunction with fluid simulation techniques, which usually generate flow fields as output. Another interesting application is flow visualization using arbitrary textures. Our technique synthesizes texture sequences that animate the input texture as guided by the given flow field. Hence, it facilitates flow visualization using a rich variety of textures.

2 Related Work

There has been a plethora of work towards synthesizing textures from example inputs. Local region-growing techniques synthesize the texture one pixel or one patch at a time. Among them, patch-based techniques [Efros and Freeman 2001; Liang et al. 2001; Cohen et al. 2003; Kwatra et al. 2003; Wu and Yu 2004] are generally more successful at synthesizing high quality textures as they can maintain global structure of the texture well. On the other hand, pixel-based methods [DeBonet 1997; Efros and Leung 1999; Wei and Levoy 2000] are more amenable to constrained synthesis as they have control over individual pixel values. Our technique is intermediate between pixel and patch based methods and combines the advantages of both.

Global synthesis methods have usually employed matching of statistical properties like histograms and wavelet coefficients between input and output textures [Heeger and Bergen 1995; Portilla and Simoncelli 2000]. There has also been previous work that makes use of optimization over MRFs for synthesis. Paget and Longstaff [1998] use local annealing over a multi-scale MRF for texture synthesis. They consider only pixel-to-pixel interactions. In contrast, our technique uses an EM-like approach for optimization and can handle interactions between large neighborhoods. Freeman et al. [2002] use belief propagation over an MRF for super-resolution. They also consider interactions across large neighborhoods. However, theirs is a fully discrete optimization while our approach is semi-discrete-continuous, which leads to a simpler as well as more flexible optimization algorithm. In image analysis, Jojic et al. [2003] use a distance metric similar to our texture energy metric for computing image epitomes. Fitzgibbon et al. [2003] also use a similar metric as texture prior for image-based rendering, while Wexler et al. [2004] use it for hole-filling in video. [Wei and Levoy 2002] is a global pixel-based approach where non-causal neighborhoods are used to simultaneously determine each pixel of the evolving texture; we, on the other hand, directly combine these neighborhoods as patches within our optimization.

Although most work in texture synthesis has focussed on unconstrained synthesis, there has also been some research on adding user control to these methods. Ashikhmin [2001] allows a user to specify large scale properties of the output texture through a painting-like interface. Efros and Freeman [2001] perform texture transfer on arbitrary images by matching correspondence maps. Hertzmann et al. [2001] learn filtering and painting operations from example input-output pairs, which are then applied to new inputs. Zhang et al.[2003] synthesize progressively varying textures by identifying and manipulating textons. Neyret [2003] applies textures to animated fluids through a blend of advection and regeneration of the original texture. The goals of their work and our flow-guided synthesis method are similar, but we can handle a larger class of structured as well as stochastic textures.

Techniques for synthesizing and controlling textures have also been extended to the video domain. Even though we focus on image textures, controllable temporal texture synthesis is relevant in the context of our flow-guided animation technique. In their video textures work, Schödl et al. [2000] blend multiple videos at varying speeds to control the speed of the synthesized video. Doretto and Soatto [2003] edit the speed, intensity, and scale of video textures by editing parameters of a Linear Dynamical System. Bregler et al. [1997] and Ezzat et al. [2002] use speech to control facial animation synthesis from video data. Bhat et al. [2004] provide an interactive system for editing video by specifying flow lines in the input and output sequences. The goal of their work is similar to ours in that low-level control (in the form of flow) is applied to synthesize new video. However, our texture synthesis algorithms differ completely. Our synthesis technique can handle arbitrary time-varying flow fields but uses image textures as input. Their technique, on the other hand, can handle only stationary flow fields but focuses on video.

3 Texture Similarity and Synthesis

We now describe the texture energy metric that measures similarity of the synthesized (output) texture to the input sample. We define this global energy in terms of similarity of local neighborhoods in the output to local neighborhoods in the input. Note that local pixel-growing techniques use similar matching of neighborhoods to maintain local coherence. We, however, combine these local comparisons into a global metric that defines a quality measure for the entire texture. We postulate that a *sufficient* condition for a given texture to be similar to the input sample is that all neighborhoods in the given texture be similar to some neighborhood in the input. Of course, the neighborhood size should be large enough to capture repeating elements of the texture. We define the energy of a single neighborhood to be its distance to the closest neighborhood in the input. The total energy of the texture is then equal to the sum of energies over individual local neighborhoods in the texture.

Formally, let X denote the texture over which we want to compute the texture energy and Z denote the input sample to be used as reference. Let $\mathbf{x}$ be the *vectorized* version of X, *i.e.,* it is formed by concatenating the intensity values of all pixels in X. For a prespecified neighborhood width w, let $\mathcal{N}_p$ represent the neighborhood in X centered around pixel p. Then, the *sub-vector* of $\mathbf{x}$ that corresponds to the pixels in $\mathcal{N}_p$ is denoted by $\mathbf{x}_p$. Further, let $\mathbf{z}_p$ be the vectorized pixel neighborhood in Z whose appearance is most similar to $\mathbf{x}_p$ under the Euclidean norm. Then, we define the texture

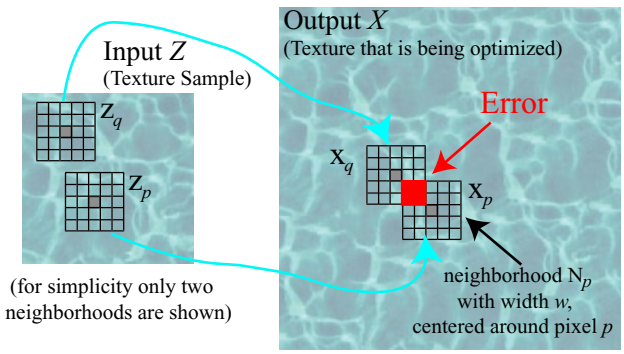


Figure 2: Schematic demonstrating our texture similarity metric. The energy of neighborhood $\mathbf{x}_p$ centered around pixel p is given by its distance to the closest input neighborhood $\mathbf{z}_p$. When two neighborhoods $\mathbf{x}_p$ and $\mathbf{x}_q$ overlap, then any mismatch between $\mathbf{z}_p$ and $\mathbf{z}_q$ will lead to accumulation of error in the overlapping region (shown in red).

energy over X to be

$$E_t(\mathbf{x};\{\mathbf{z}_p\}) = \sum_{p\in X^\dagger} \|\mathbf{x}_p - \mathbf{z}_p\|^2. \quad (1)$$

This is shown schematically in Figure 2. We only consider neighborhoods centered around pixels in a set $X^\dagger \subset X$ for computing the energy. We do so because in practice, it is redundant and computationally expensive to compute the energy over all neighborhoods in the texture. Therefore, we pick a subset of neighborhoods that sufficiently overlap with each other and define the energy only over this subset. This is also advantageous with respect to our synthesis algorithm (explained in the next paragraph); choosing neighborhoods located on a sparse grid ensures that only a small number of these neighborhoods simultaneously affect a given pixel. This prevents the synthesized texture from getting too blurry in regions where there is a mismatch between overlapping neighborhoods. We have empirically chosen $X^\dagger$ to consist of neighborhood centers that are $w/4$ pixels apart, where w is the width of each neighborhood.

We can use the above formulation to perform texture synthesis by iteratively refining an initial estimate of the texture, decreasing the texture energy at each iteration. During each iteration, we alternate between $\mathbf{x}$ and $\{\mathbf{z}_p : p \in X^\dagger\}$ as the variables with respect to which (1) is minimized. Given an initialization of the texture, $\mathbf{x}$, we first find the closest input neighborhood $\mathbf{z}_p$ corresponding to each output neighborhood $\mathbf{x}_p$. We then update $\mathbf{x}$ to be the texture that minimizes the energy in (1) – note that we treat $\mathbf{x}$ as a real-valued continuous vector variable. Since $\mathbf{x}$ changes after this update step, the set of closest input neighborhoods $\{\mathbf{z}_p\}$ may also change. Hence, we need to repeat the two steps iteratively until convergence, *i.e.,* until the set $\{\mathbf{z}_p\}$ stops changing. If $\mathbf{x}$ is initially unknown, then we bootstrap the algorithm by assigning a random neighborhood from the input to each $\mathbf{z}_p$. Algorithm 1 describes the pseudocode for our texture synthesis algorithm.

Our approach is algorithmically similar to Expectation-Maximization (EM) [McLachlan and Krishnan 1997]. EM is used for optimization in circumstances where, in addition to the desired variables, the parameters of the energy function being optimized are also unknown. Therefore, one alternates between estimating the variables and the parameters in the E and the M steps respectively. In our case, the desired variable is the texture image, $\mathbf{x}$, while the parameters are the input neighborhoods, $\{\mathbf{z}_p\}$. The two steps of our algorithm can be thought of as E and M steps. The estimation of $\mathbf{x}$ by minimizing the texture energy in (1) corresponds to the E-step, while finding the set of closest input neighborhoods, $\{\mathbf{z}_p\}$, corresponds to the M-step.

Algorithm 1 Texture Synthesis

$\mathbf{z}_p^0 \leftarrow$ random neighborhood in $Z \;\; \forall p \in X^\dagger$
for iteration $n = 0 : N$ **do**
  $\mathbf{x}^{n+1} \leftarrow \arg\min_{\mathbf{x}} E_t(\mathbf{x};\{\mathbf{z}_p^n\})$
  $\mathbf{z}_p^{n+1} \leftarrow$ nearest neighbor of $\mathbf{x}_p^{n+1}$ in $Z \;\; \forall p \in X^\dagger$
  if $\mathbf{z}_p^{n+1} = \mathbf{z}_p^n \;\; \forall p \in X^\dagger$ **then**
    $\mathbf{x} \leftarrow \mathbf{x}^{n+1}$
    break
  end if
end for

In the E-step, we need to minimize (1) w.r.t. $\mathbf{x}$. This is done by setting the derivative of (1) w.r.t. $\mathbf{x}$ to zero, which yields a linear system of equations that can be solved for $\mathbf{x}$. One can think of each term in (1) as the potential resulting from a force that pulls the pixels in $\mathbf{x}_p$ towards pixels in $\mathbf{z}_p$. Minimization of this potential corresponds to bringing each sub-vector $\mathbf{x}_p$ as close to $\mathbf{z}_p$ as possible. If neighborhoods centered around different pixels p and q overlap with each other, then the corresponding sub-vectors $\mathbf{x}_p$ and $\mathbf{x}_q$ will also contain common pixels. Each such common pixel is pulled towards possibly different intensity values by $\mathbf{z}_p$ and $\mathbf{z}_q$. The outcome of the minimization procedure is to assign an intensity value to the common pixel that is equal to the average of the corresponding values in $\mathbf{z}_p$ and $\mathbf{z}_q$. Note that for a quadratic $E_t(\mathbf{x};\{\mathbf{z}_p\})$ (as in (1)), this minimization is equivalent to computing the expected value (or mean) of $\mathbf{x}$ under the following probability distribution[1]:

$$p(\mathbf{x};\{\mathbf{z}_p\}) \propto \exp(-E_t(\mathbf{x};\{\mathbf{z}_p\})).$$

The M-step of our algorithm minimizes (1) with respect to the set of input neighborhoods, $\{\mathbf{z}_p\}$, keeping $\mathbf{x}$ fixed at the value estimated in the E-step. This requires us to solve a nearest neighbor search problem: for each $\mathbf{x}_p$, we need to find its nearest neighbor $\mathbf{z}_p$ from the input. To accelerate this search, we use hierarchical clustering to organize the input neighborhoods into a tree structure [Johnson 1967; Elkan 2003; Dellaert et al. 2005]. Starting at the root node, we perform *k-means* clustering (with $k = 4$) over all input neighborhoods contained in that node. We then create k children nodes corresponding to the k clusters and recursively build the tree for each of these children nodes. The recursion stops when the number of neighborhoods in a node falls below a threshold (1% of total in our implementation). In order to handle large neighborhood sizes, we employ a memory-efficient adaptation that does not explicitly store neighborhood sub-vectors at leaf nodes. Instead, it records just the neighborhood's location in the input image and the corresponding sub-vectors are constructed on the fly, as necessary.

Intuitively, our algorithm tries to find *good* relative arrangements of input neighborhoods in order to synthesize a new texture. During each iteration, the M-step chooses an arrangement of input neighborhoods that best explains the current estimate of the texture. The averaging in the E-step allows overlapping neighborhoods to communicate consistency information among each other: neighborhoods that don't match well with each other cause blurring in the synthesized texture. This blurred region represents a transition between two inconsistent regions and may be significantly different in appearance from the input neighborhoods that determine it. This allows the next iteration of the M-step to replace the input neighborhoods corresponding to this region with ones that are more consis-

[1]For exact EM, the E-step should also compute the covariance of $\mathbf{x}$ in addition to its mean. Our formulation only computes the mean as it is based on an energy function and not a probability distribution. Consequently, we do not perform exact EM.

tent with each other, *i.e.,* neighborhoods that carry out the transition but get rid of the blurring.

Gradient-based Energy: We can generalize the energy function defined in (1) to incorporate other characteristics of the texture besides color. For example, in order to use image gradients as an additional similarity metric, we define the energy as

$$E_t(\mathbf{x};\{\mathbf{z}_p\}) = \sum_{p\in X^\dagger} \|\mathbf{x}_p - \mathbf{z}_p\|^2 + \mu \sum_{p\in X^\dagger} \|\mathbf{D}\mathbf{x}_p - \mathbf{D}\mathbf{z}_p\|^2,$$

where $\mathbf{D}$ is the differentiation operator and μ is a relative weighting coefficient ($\mu = 10$ in our experiments). Minimizing this function w.r.t. $\mathbf{x}$ requires us to solve a linear system very similar to the Poisson equation (see [Pérez et al. 2003] for image editing applications). Even though we have experimented with color and gradient, one could use other energy functions of the form $\|\psi(\mathbf{x}_p) - \psi(\mathbf{z}_p)\|^2$ where $\psi(\mathbf{x}_p)$ measures some property of the texture neighborhood $\mathbf{x}_p$. The only requirement is that ψ should be optimizable w.r.t. $\mathbf{x}$.

3.1 Robust Optimization

The quadratic texture energy as defined in (1) performs least squares estimation of $\mathbf{x}$ w.r.t. $\mathbf{z}_p$. This causes outliers – $\mathbf{z}_p$ that are not very close to $\mathbf{x}_p$ – to have an undue influence on $\mathbf{x}$. Also, it is desirable to not change $\mathbf{x}_p$ by much if it is already very close to $\mathbf{z}_p$. This can be accomplished by using a robust energy function: we replace the squared term $\|\mathbf{x}_p - \mathbf{z}_p\|^2$ in (1) with $\|\mathbf{x}_p - \mathbf{z}_p\|^r$, where $r < 2$. This energy function belongs to a class of robust regressors, called M-estimators, that are typically solved using iteratively reweighted least squares (IRLS) [Coleman et al. 1980]. IRLS is an iterative technique in which a weighted least squares problem is solved during every iteration. The weights are adjusted at the end of the iteration, and this procedure is repeated. Our synthesis algorithm naturally lends itself to IRLS: before applying the E-step, we choose, for each neighborhood $\mathcal{N}_p$, a weight $\omega_p = \|\mathbf{x}_p - \mathbf{z}_p\|^{r-2}$ – in our implementation, we have used $r = 0.8$. We then minimize the modified energy function:

$$E_t(\mathbf{x};\{\mathbf{z}_p\}) = \sum_{p\in X^\dagger} \omega_p \|\mathbf{x}_p - \mathbf{z}_p\|^2.$$

Additionally, we apply a per pixel weight within the energy term for each neighborhood, based on a Gaussian fall-off function. This ensures that pixels closer to the center of the neighborhood have a greater bearing on the texture than those far away.

3.2 Multi-level Synthesis

We use our algorithm in a multi-resolution and multi-scale fashion. We first synthesize our texture at a coarse resolution, and then up-sample it to a higher resolution via interpolation. This serves as the initialization of the texture at the higher resolution. Also, within each resolution level, we run our synthesis algorithm using multiple neighborhood sizes in order from largest to smallest. Such a multi-level approach is helpful because it allows the finer scale synthesis to begin with a good initialization of the texture, thereby avoiding undesirable local minima. Intuitively, at a lower resolution, texture neighborhoods are spatially close to each other and it is easier to propagate consistency information across the entire texture. Also, by starting with a large neighborhood size, we are able to synthesize large scale structures of the texture first. Subsequently, synthesis with smaller neighborhood sizes removes fine scale errors from the texture. In our experiments, we generally use three resolution levels and successive neighborhood sizes of 32×32, 16×16, and 8×8 pixels – at each resolution, only those neighborhood sizes are used that fit in the the corresponding (potentially sub-sampled) input image.

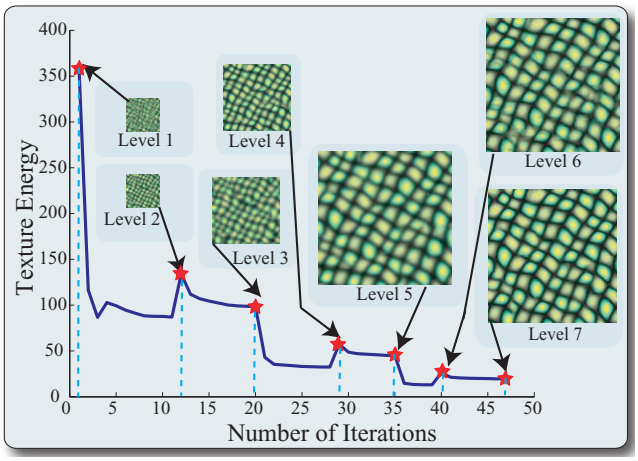


Figure 3: Texture energy plotted as a function of number of iterations. Also shown is the synthesized texture after each resolution and scale (neighborhood size) level. Level 1 shows the random initialization. Level 2 shows synthesis at (1/4 resolution, 8×8 neighborhood), Level 3: (1/2, 16×16), Level 4: (1/2, 8×8), Level 5: (1, 32×32), Level 6: (1, 16×16), Level 7: (1, 8×8).

In Figure 3, we plot the energy of the synthesized texture as a function of number of iterations. The iterations for various resolution and scale levels are concatenated in the order of synthesis – we normalize the energy at each level by the number of pixels and neighborhood size. We also show the synthesized texture at the end of each synthesis level. The texture energy generally decreases as the number of iterations increase, thereby improving texture quality. Note that intermediate iterations produce textures that appear to be coarse-scale approximations of the final texture. One can potentially exploit this progressive refinement property to synthesize textures in a level-of-detail fashion. The jumps seen in the energy plot are due to change of synthesis level – resolution or neighborhood size. They are in fact desirable because they help the algorithm to get out of poor local minima.

4 Controllable Synthesis

We perform controllable texture synthesis within our framework by augmenting the texture energy function with an additional term representing the control criteria. Optimization then proceeds by minimizing the total energy sum at each iteration. A simple example of adding control is to perform *soft*-constrained synthesis where the desirable color values are specified at certain pixel locations. We can express the energy representing this criterion as the sum of squared distances between the synthesized and specified pixel values:

$$E_c(\mathbf{x};\mathbf{x}^c) = \sum_{k\in\mathscr{C}} (\mathbf{x}(k) - \mathbf{x}^c(k))^2, \qquad (2)$$

where $\mathscr{C}$ is the set of constrained pixels and $\mathbf{x}^c$ is a vector containing the specified color values. One can use a more general energy function of the form $E_c(\mathbf{x};\mathbf{u})$ where $\mathbf{u}$ represents a control vector. We then optimize the total energy, defined as

$$E(\mathbf{x}) = E_t(\mathbf{x};\{\mathbf{z}_p\}) + \lambda E_c(\mathbf{x};\mathbf{u}),$$

where λ is a relative weighting coefficient. The control term, $E_c(\mathbf{x};\mathbf{u})$, attempts to satisfy the control criteria in the synthesized

texture, while the texture term, $E_t(\mathbf{x};\{\mathbf{z}_p\})$, tries to ensure that it is a representative sample of the input texture. The minimization of $E(\mathbf{x})$ is done in a similar fashion as that of $E_t(\mathbf{x};\{\mathbf{z}_p\})$. We modify the E and M steps to account for $E_c(\mathbf{x};\mathbf{u})$ as follows.

In the E-step, we solve a new system of linear equations that results from the differentiation of $E(\mathbf{x})$ w.r.t. $\mathbf{x}$. For the soft-constrained synthesis example described above, this corresponds to taking a weighted average of the specified pixel values and those corresponding to the output of texture synthesis.

We also modify the M-step that estimates $\{\mathbf{z}_p\}$. Even though $E_c(\mathbf{x};\mathbf{u})$ does not directly depend on $\{\mathbf{z}_p\}$, they are indirectly related to each other through $\mathbf{x}$. We exploit the fact that each synthesized neighborhood $\mathbf{x}_p$ will be similar to $\mathbf{z}_p$ after the E-step. Hence, when searching for $\mathbf{z}_p$, we look for input neighborhoods that are already consistent with the control criteria. The intuition is that if $\mathbf{z}_p$ has low control energy, then so will $\mathbf{x}_p$. For each $\mathbf{x}_p$, we find the $\mathbf{z}_p$ that minimizes $\|\mathbf{x}_p-\mathbf{z}_p\|^2+\lambda E_c(\mathbf{y};\mathbf{u})$, where, $\mathbf{y}$ is constructed from $\mathbf{x}$ by replacing its pixels in the neighborhood $\mathcal{N}_p$ with $\mathbf{z}_p$, *i.e.,*

$$\mathbf{y}(q)=\begin{cases}\mathbf{z}_p(q-p+w/2) & q\in\mathcal{N}_p\\ \mathbf{x}(q) & \text{otherwise,}\end{cases}$$

where $w=(w_x,w_y)$ encodes the neighborhood width in both spatial dimensions. As an example, in the case of soft-constrained synthesis, the new M-step searches for $\mathbf{z}_p$ that minimizes $\|\mathbf{x}_p-\mathbf{z}_p\|^2+\lambda\|\mathbf{z}_p-\mathbf{x}_p^c\|^2$, *i.e.,* an input neighborhood whose pixel values at the constrained locations are already close to the specified ones. Algorithm 2 describes the pseudocode for controllable texture synthesis.

Algorithm 2 Controllable Texture Synthesis

$\mathbf{z}_p^0 \leftarrow$ random neighborhood in $Z\ \ \forall p\in X^\dagger$
for iteration $n=0:N$ **do**
  $\mathbf{x}^{n+1}\leftarrow \arg\min_{\mathbf{x}}[E_t(\mathbf{x};\{\mathbf{z}_p^n\})+\lambda E_c(\mathbf{x};\mathbf{u})]$
  $\mathbf{z}_p^{n+1}\leftarrow \arg\min_{\mathbf{v}}[\|\mathbf{x}_p-\mathbf{v}\|^2+\lambda E_c(\mathbf{y};\mathbf{u})]$, $\mathbf{v}$ is a neighborhood in Z and $\mathbf{y}$ is the same as $\mathbf{x}$ except for neighborhood $\mathbf{x}_p$ which is replaced with $\mathbf{v}$
  if $\mathbf{z}_p^{n+1}=\mathbf{z}_p^n\ \ \forall p\in X^\dagger$ **then**
    $\mathbf{x}\leftarrow\mathbf{x}^{n+1}$
    break
  end if
end for

4.1 Flow-guided Synthesis

We have experimented with flow as a control criterion: we synthesize texture sequences that move according to a given input flow field. To this end, we augment our energy function with a control term that measures consistency of the sequence with the given flow.

Let $\mathbf{f}$ denote the input flow field using which we want to animate the texture. $\mathbf{f}$ is a sequence of 2D flow fields $(f_1,f_2,\ldots f_{L-1})$, where L is the length of the texture sequence being synthesized, and f_i is the desirable flow field between frame i and $i+1$. For a given pixel location p in frame i, $f_i(p)$ gives its location in frame $i+1$ after it has been transported through the flow field. Let X_i and X_{i+1} denote two consecutive frames of the sequence[2]. Then we want the pixels in X_i and X_{i+1} that are related via f_i, to be similar in appearance. Pixels (p,i) and $(q,i+1)$ are said to be related via f_i if $q=f_i(p)$. Intuitively, this means that we want pixels to retain their appearance as much as possible while moving along the flow field. Hence, we define the control energy function to be

$$E_c(\mathbf{x};\mathbf{f})=\sum_{i\in 1:L-1}\sum_{(p,i)\in X_i}(\mathbf{x}(p,i)-\mathbf{x}(q,i+1))^2,\ \ q=f_i(p). \quad (3)$$

We perform synthesis in a frame-by-frame fashion. We are able to do so because texture neighborhoods are 2D, hence any coupling between neighborhoods in different frames occurs only via the flow field. During synthesis of frame $i+1$, all pixels of frame i, denoted by $\mathbf{x}(p,i)$ in (3), are already known. Consequently, the flow energy function in (3) has a very similar form to the one defined in (2). Hence, we can treat it as a soft-constrained synthesis problem with the pixels of frame i serving as the constraint vector. In particular, we synthesize the first frame of the sequence using regular texture synthesis. For each subsequent frame, the desirable pixel values at each location are obtained by warping the previous frame using its flow field. The warped previous frame also acts as an initialization for the current frame. We then synthesize the frame using Algorithm 2.

Our technique is well-suited for flow-guided synthesis because it exercises flexibility in choosing the granularity at which different texture elements are synthesized – achieved through the use of multiple neighborhood sizes. Also, modulation of the energy function with appropriate weights allows for additional control over individual pixel values. Consequently, our technique can be regarded as intermediate between patch and pixel based methods. Among existing techniques, pixel-based methods are more flexible than patch-based methods and one could extend them for flow-guided synthesis. However, they are known to generate lower quality results than patch-based techniques. On the other hand, patch-based methods affect large sets of pixels at the same time. Hence, they may be more difficult to adapt for flow-guided synthesis which requires control over individual pixel values. Our technique is successful because it combines the advantages of both pixel and patch based paradigms.

5 Results

Figure 4 shows some of our results for image texture synthesis. Our technique works well for a wide range of textures varying from stochastic to structured, as shown in the figure. In Figure 5, we show comparisons with other techniques. The results of our optimization-based approach are at par with state-of-the-art patch-based synthesis techniques like image quilting and graph cuts, and compare favorably against pixel-based techniques like that of Wei and Levoy.

We demonstrate our controllable synthesis algorithm by animating textures using flow fields. Figure 6 shows a sampling of the flow fields used in our experiments. It also shows the variability in the distortion that texture elements undergo under the effect of different flows. We have used an interactive vector field design technique [Zhang et al. 2004] to create most of our flows. In Figure 7, we compare the result of our approach to that obtained by applying a simple warp based on a rotational flow field to a keyboard texture. Warping leaves holes in the texture and does not preserve the appearance of texture elements. On the other hand, our technique maintains the shape as well as orientation of the keys as it rotates the texture. We can use our technique to synthesize texture animations for time-varying as well as stationary flow fields. We show results for both cases in the supplementary video.

The computational complexity of our technique is dominated by nearest neighbor search (M-step). It is linear in the number of nearest neighbor calls, which are $O(\frac{n_o}{w^2})$ per iteration – n_o is the number of pixels in the output texture while w is the width of the neighborhood. Theoretically, the time taken per call is $O(w^2)$. In practice, however, we found it to be less than that – the exact dependence is not known to us as it is governed by MATLAB's vectorized matrix

[2] For a texture sequence, X is 3D and X_i represents a 2D slice within X. We denote pixels in X as (p,i) where p is the pixel location in 2D and i is the frame number.

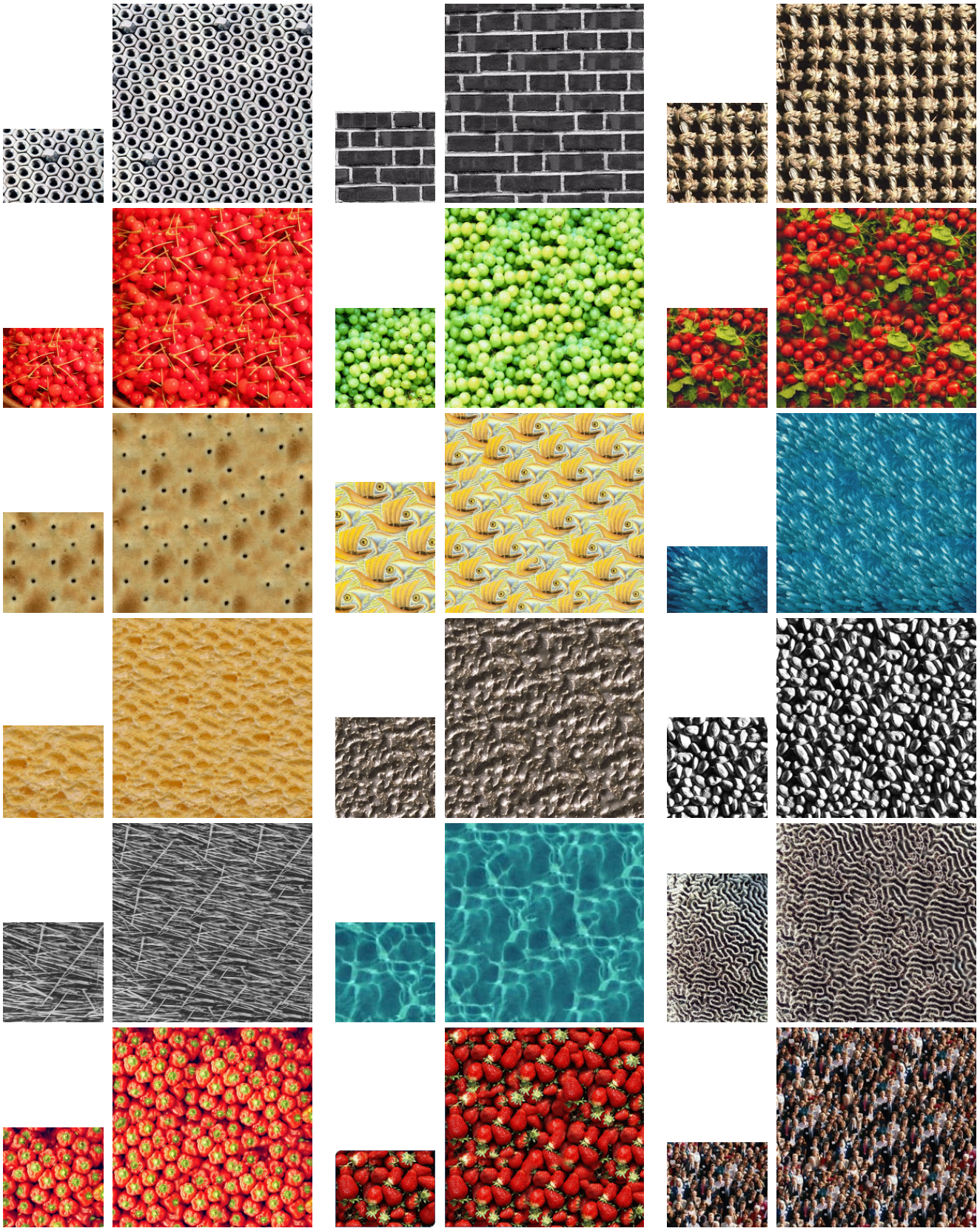


Figure 4: Results for image texture synthesis. For each texture, the input is on the left and the output on the right.

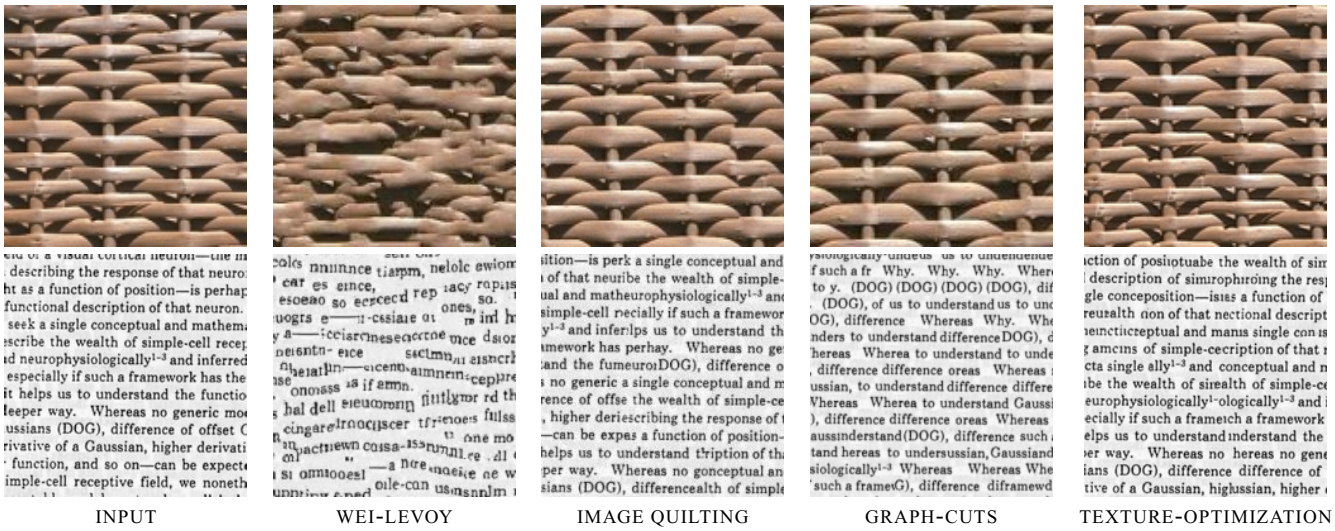


Figure 5: Comparison with various other texture synthesis techniques. Results for other techniques were obtained from their web pages.

multiplication algorithm. The implication is that execution is faster for larger w. Actual execution time per iteration at different resolution levels (averaged over the various neighborhood sizes used within that resolution) was: 2.5 seconds at 64×64, 10 seconds at 128×128, and 25 seconds at 256×256. Usually 3-5 iterations per resolution/scale level were found to be enough. Average total execution time for multi-level synthesis of 256×256 textures was 7-10 minutes, while for 128×128 textures, it was 1-3 minutes. For flow-guided synthesis, each frame was synthesized only at a single resolution. Execution time per frame was between 20-60 seconds. All timing results are reported for our unoptimized MATLAB code, running on a dual-processor 2.4GHz PC with 2GB RAM.

6 Summary & Future Work

We have presented a novel optimization-based technique for texture synthesis. Our results for image texture synthesis are comparable to the state-of-the-art. We define a texture energy function that allows us to quantitatively measure quality of the synthesized texture. Our synthesis algorithm iteratively improves this texture quality and is therefore suitable for progressive refinement of texture. We can easily extend our technique to perform controlled synthesis by adding new terms to the energy formulation. We demonstrate this by presenting a technique for flow-guided animation of image textures.

A limitation of our technique is that because it tries to decrease the energy at each iteration, it can get stuck in local minima. This is manifested as blurring or misalignment of texture elements. It happens because spatially distant neighborhoods communicate with each other only through intermediate overlapping neighborhoods. Multi-level synthesis partially alleviates this problem by bringing distant neighborhoods closer to each other via down-sampling.

As future work, we want to extend our flow-guided synthesis technique to handle video textures. Also, we wish to explore other types of control criteria besides motion, *e.g.,* illumination, shape, etc. Another research direction is to experiment with other texture properties besides color and gradient in defining texture energy.

Acknowledgements

We thank the reviewers for their insightful comments. We thank Eugene Zhang and his collaborators for sharing their vector field design code, and Delphine Nain for her flow interpolation code. Thanks to Greg Turk, Gabriel Brostow, Stephanie Brubaker, Raffay Hamid, and Drew Steedly for useful discussions and suggestions, and to Ravi Ruddarraju, Gaurav Chanda, Mitch Parry, and Pei Yen for helping with production. Funding was provided in part by the SRI/DARPA CALO grant.

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. *2001 ACM Symposium on Interactive 3D Graphics* (March), 217–226.

BHAT, K. S., SEITZ, S. M., HODGINS, J. K., AND KHOSLA, P. K. 2004. Flow-based video synthesis and editing. *ACM Transactions on Graphics (SIGGRAPH 2004) 23*, 3 (August).

BREGLER, C., COVELL, M., AND SLANEY, M. 1997. Video rewrite: Driving visual speech with audio. *Proceedings of SIGGRAPH 97* (August), 353–360. ISBN 0-89791-896-7. Held in Los Angeles, California.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. *ACM Transactions on Graphics, SIGGRAPH 2003 22*, 3, 287–294.

COLEMAN, D., HOLLAND, P., KADEN, N., KLEMA, V., AND PETERS, S. C. 1980. A system of subroutines for iteratively reweighted least squares computations. *ACM Trans. Math. Softw. 6*, 3, 327–336.

DEBONET, J. S. 1997. Multiresolution sampling procedure for analysis and synthesis of texture images. *Proceedings of ACM SIGGRAPH 97* (August), 361–368.

DELLAERT, F., KWATRA, V., AND OH, S. M. 2005. Mixture trees for modeling and fast conditional sampling with applications in vision and graphics. In *IEEE Computer Vision and Pattern Recognition*.

DORETTO, G., AND SOATTO, S. 2003. Editable dynamic textures. In *IEEE Computer Vision and Pattern Recognition*, II: 137–142.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, 341–346.

EFROS, A., AND LEUNG, T. 1999. Texture synthesis by non-parametric sampling. In *International Conference on Computer Vision*, 1033–1038.

ELKAN, C. 2003. Using the triangle inequality to accelerate k-means. In *International Conference on Machine Learning*.

EZZAT, T., GEIGER, G., AND POGGIO, T. 2002. Trainable videorealistic speech animation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 388–398.

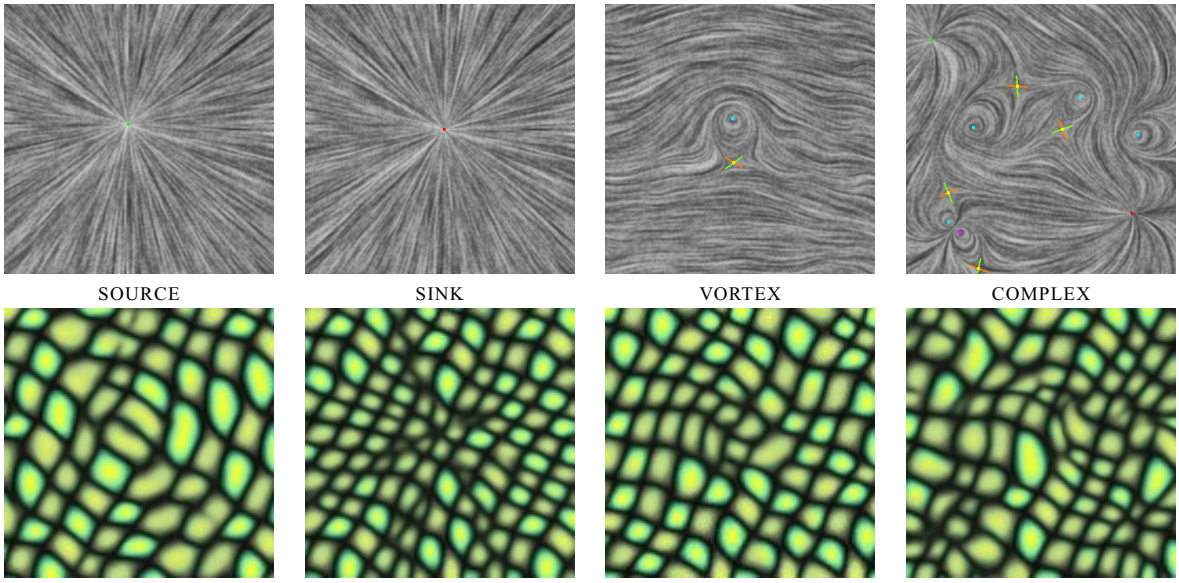


Figure 6: Flow-guided synthesis using different flow fields. Shown is the 25th frame of each sequence. All sequences start with the same frame (not shown).

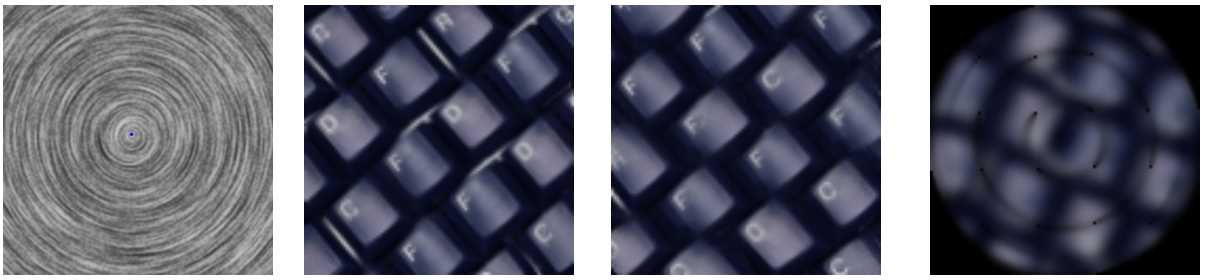


Figure 7: Comparison of flow-guided synthesis with simple warping. The keys on the keyboard maintain their orientation while rotating.

FITZGIBBON, A., WEXLER, Y., AND ZISSERMAN, A. 2003. Image-based rendering using image-based priors. In *International Conference on Computer Vision*.

FREEMAN, W. T., JONES, T. R., AND PASZTOR, E. C. 2002. Example-based super-resolution. *IEEE Comput. Graph. Appl. 22*, 2, 56–65.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-based texture analysis/synthesis. *Proceedings of ACM SIGGRAPH 95* (August), 229–238.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. *Proceedings of SIGGRAPH 2001* (August), 327–340. ISBN 1-58113-292-1.

JOHNSON, S. C. 1967. Hierarchical clustering schemes. *Psychometrika 2*, 241–254.

JOJIC, N., FREY, B., AND KANNAN, A. 2003. Epitomic analysis of appearance and shape. In *International Conference on Computer Vision*.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003 22*, 3 (July), 277–286.

LIANG, L., LIU, C., XU, Y.-Q., GUO, B., AND SHUM, H.-Y. 2001. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics Vol. 20, No. 3* (July), 127–150.

MCLACHLAN, G., AND KRISHNAN, T. 1997. *The EM algorithm and extensions*. Wiley series in probability and statistics. John Wiley & Sons.

NEYRET, F. 2003. Advected textures. *Symposium on Computer Animation'03* (July).

PAGET, R., AND LONGSTAFF, I. D. 1998. Texture synthesis via a noncausal nonparametric multiscale markov random field. *IEEE Transactions on Image Processing 7*, 6 (June), 925–931.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Transactions on Graphics, SIGGRAPH 2003 22*, 3, 313–318.

PORTILLA, J., AND SIMONCELLI, E. P. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision 40*, 1 (October), 49–70.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. *Proceedings of ACM SIGGRAPH 2000* (July), 489–498.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. *Proceedings of ACM SIGGRAPH 2000* (July), 479–488. ISBN 1-58113-208-5.

WEI, L.-Y., AND LEVOY, M. 2002. Order-independent texture synthesis. Tech. Rep. TR-2002-01, Stanford University CS Department.

WEXLER, Y., SHECHTMAN, E., AND IRANI, M. 2004. Space-time video completion. In *CVPR 2004*, 120–127.

WU, Q., AND YU, Y. 2004. Feature matching and deformation for texture synthesis. *ACM Transactions on Graphics (SIGGRAPH 2004)* (August).

ZHANG, J., ZHOU, K., VELHO, L., GUO, B., AND SHUM, H.-Y. 2003. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics, SIGGRAPH 2003 22*, 3, 295–302.

ZHANG, E., MISCHAIKOW, K., AND TURK, G. 2004. Vector field design on surfaces. Tech. Rep. 04-16, Georgia Institute of Technology.