Graphcut Textures: Image and Video Synthesis Using Graph Cuts

Vivek Kwatra Arno Schödl Irfan Essa Greg Turk Aaron Bobick

GVU Center / College of Computing
Georgia Institute of Technology
http://www.cc.gatech.edu/cpl/projects/graphcuttextures

This banner was generated by merging the source images in Figure 6 using our interactive texture merging technique.

Abstract

In this paper we introduce a new algorithm for image and video texture synthesis. In our approach, patch regions from a sample image or video are transformed and copied to the output and then stitched together along optimal seams to generate a new (and typically larger) output. In contrast to other techniques, the size of the patch is not chosen *a-priori*, but instead a *graph cut* technique is used to determine the optimal patch region for any given offset between the input and output texture. Unlike dynamic programming, our graph cut technique for seam optimization is applicable in any dimension. We specifically explore it in 2D and 3D to perform video texture synthesis in addition to regular image synthesis. We present approximative offset search techniques that work well in conjunction with the presented patch size optimization. We show results for synthesizing regular, random, and natural images and videos. We also demonstrate how this method can be used to interactively merge different images to generate new scenes.

Keywords: Texture Synthesis, Image-based Rendering, Image and Video Processing, Machine Learning, Natural Phenomenon.

1 Introduction

Generating a newer form of output from a smaller example is widely recognized to be important for computer graphics applications. For example, sample-based image texture synthesis methods are needed to generate large realistic textures for rendering of complex graphics scenes. The primary reason for such example-based synthesis underlies the concept of *texture*, usually defined as an infinite pattern that can be modeled by a stationary stochastic process. In this paper, we present a new method to generate such an infinite pattern from a small amount of training data; using a small example patch of the texture, we generate a larger pattern with similar stochastic properties. Specifically, our approach for texture synthesis generates textures by copying input texture patches. Our algorithm first searches for an appropriate location to place the patch; it then uses a *graph cut* technique to find the optimal region of the patch to transfer to the output. In our approach, textures are not limited to spatial (image) textures, and include spatio-temporal (video) textures. In addition, our algorithm supports iterative refinement of the output by allowing for successive improvement of the patch seams.

When synthesizing a texture, we want the generated texture to be perceptually similar to the example texture. This concept of perceptual similarity has been formalized as a Markov Random Field (MRF). The output texture is represented as a grid of nodes, where each node refers to a pixel or a neighborhood of pixels in the input texture. The marginal probability of a pair of nodes depends on the similarity of their pixel neighborhoods, so that pixels from similar-looking neighborhoods in the input texture end up as neighbors in the generated texture, preserving the perceptual quality of the input. The goal of texture synthesis can then be restated as the solution for the nodes of the network, that maximizes the total likelihood. This formulation is well-known in machine-learning as the problem of probabilistic inference in graphical models and is proven to be *NP-hard* in case of cyclic networks. Hence, all techniques that model the texture as a MRF [DeBonet 1997; Efros and Leung 1999; Efros and Freeman 2001; Wei and Levoy 2000] compute some approximation to the optimal solution.

In particular, texture synthesis algorithms that generate their output by copying patches (or their generalizations to higher dimensions) must make two decisions for each patch: (1) where to position the input texture relative to the output texture (the *offset* of the patch), and (2) which parts of the input texture to transfer into the output space (the patch *seam*) (Figure 1). The primary contribution of this paper is an algorithm for texture synthesis, which after finding a good patch offset, computes the best patch seam (the seam yielding the highest possible MRF likelihood among all possible seams for that offset). The algorithm works by reformulating the

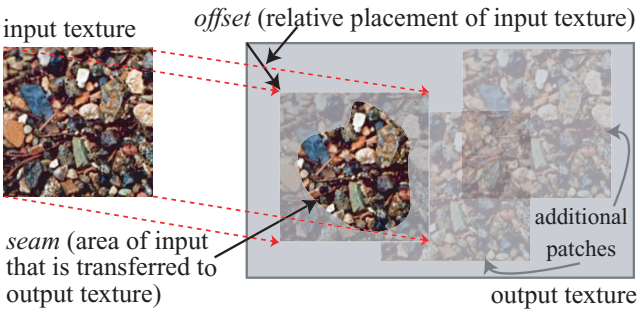

Figure 1: Image texture synthesis by placing small patches at various offsets followed by the computation of a seam that enforces visual smoothness between the existing pixels and the newly placed patch.

problem as a minimum cost graph cut problem: the MRF grid is augmented with special nodes, and a minimum cut of this grid between two special terminal nodes is computed. This minimum cut encodes the optimal solution for the patch seam. We also propose a set of algorithms to search for the patch offset at each iteration. These algorithms try to maintain the large scale structure of the texture by matching large input patches with the output. An important observation is that the flexibility of the our seam optimization technique to paste large patches at each iteration in a non-causal fashion is really what permits the design of our offset search algorithms. The offset searching and seam finding methods are therefore complementary to each other, and work in tandem to generate the obtained results.

Efros and Freeman [2001] were the first to incorporate seam finding by using dynamic programming. However, dynamic programming imposes an artificial grid structure on the pixels and therefore does not treat each pixel uniformly. This can potentially mean missing out on good seams that cannot be modeled within the imposed structure. Moreover, dynamic programming is a memoryless optimization procedure and cannot explicitly improve existing seams. This restricts its use to appending new patches to existing textures. Our graph cut method treats each pixel uniformly and is also able to place patches *over* existing texture.

Most previous work on texture is geared towards 2D images, but the texture problem in a very similar form also appears in three dimensions for the generation of spatio-temporal textures [Szummer and Picard 1996; Schödl et al. 2000; Wei and Levoy 2000; Bar-Joseph et al. 2001]. Unlike dynamic programming, which is restricted to 2D, the seam optimization presented in this paper generalizes to any dimensionality. Based on this seam optimization, we have developed algorithms for both two and three dimensions to generate spatial (2D, images) and spatio-temporal (3D, video) textures.

Finally, we have extended our algorithm to allow for multiple scales and different orientations which permits the generation of larger images with more variety and perspective variations. We have also implemented an interactive system that allows for merging and blending of different types of images to generate composites without the need for any a *priori* segmentation.

2 Related work

Texture synthesis techniques that generate an output texture from an example input can be roughly categorized into three classes. The first class uses a fixed number of parameters within a compact parametric model to describe a variety of textures. Heeger and Bergen [1995] use color histograms across frequency bands as a texture description. Portilla and Simoncelli's model [2000] includes a variety of wavelet features and their relationships, and is probably the best parametric model for image textures to date. Szummer and Picard [1996], Soatto et al. [2001], and Wang and Zhu [2002] have proposed parametric representations for video. Parametric models cannot synthesize as large a variety of textures as other models described here, but provide better model generalization and are more amenable to introspection and recognition [Saisan et al. 2001]. They therefore perform well for analysis of textures and can provide a better understanding of the perceptual process.

The second class of texture synthesis methods is non-parametric, which means that rather than having a fixed number of parameters, they use a collection of *exemplars* to model the texture. DeBonet [1997], who pioneered this group of techniques, samples from a collection of multi-scale filter responses to generate textures. Efros and Leung [1999] were the first to use an even simpler approach, directly generating textures by copying pixels from the input texture. Wei and Levoy [2000] extended this approach to multiple frequency bands and used vector quantization to speed up the processing. These techniques all have in common that they generate textures one pixel at a time.

The third, most recent class of techniques generates textures by copying whole *patches* from the input. Ashikmin [2001] made an intermediate step towards copying patches by using a pixel-based technique that favors transfer of coherent patches. Liang et al. [2001], Guo et al. [2000], and Efros and Freeman [2001] explicitly copy whole patches of input texture at a time. Schödl et al. [2000] perform video synthesis by copying whole frames from the input sequence. This last class of techniques arguably creates the best synthesis results on the largest variety of textures. These methods, unlike the parametric methods described above, yield a limited amount of information for texture analysis.

Across different synthesis techniques, textures are often described as Markov Random Fields [DeBonet 1997; Efros and Leung 1999; Efros and Freeman 2001; Wei and Levoy 2000]. MRFs have been studied extensively in the context of computer vision [Li 1995]. In our case, we use a graph cut technique to optimize the likelihood of the MRF. Among other techniques using graph cuts [Greig et al. 1989], we have chosen a technique by Boykov et al. [1999], which is particularly suited for the type of cost function found in texture synthesis.

3 Patch Fitting using Graph Cuts

We synthesize new texture by copying irregularly shaped patches from the sample image into the output image. The patch copying process is performed in two stages. First a candidate rectangular patch (or patch offset) is selected by performing a comparison between the candidate patch and the pixels already in the output image. We describe our method of selecting candidate patches in a later section (Section 4). Second, an optimal (irregularly shaped) portion of this rectangle is computed and only these pixels are copied to the output image (Figure 1). The portion of the patch to copy is determined by using a graph cut algorithm, and this is the heart of our synthesis technique.

In order to introduce the graph cut technique, we first describe how it can be used to perform texture synthesis in the manner of Efros and Freeman's image quilting [2001]. Later we will see that it is a much more general tool. In image quilting, small blocks (*e.g.,* 32×32 pixels) from the sample image are copied to the output image. The first block is copied at random, and then subsequent blocks are placed such that they partly overlap with previously placed blocks of pixels. The overlap between old and new blocks is typically 4 or 8 pixels in width. Efros and Freeman use dynamic programming to choose the minimum cost path from one

end of this overlap region to the other. That is, the chosen path is through those pixels where the old and new patch colors are similar (Figure 2(left)). The path determines which patch contributes pixels at different locations in the overlap region.

To see how this can be cast into a graph cut problem, we first need to choose a matching quality measure for pixels from the old and new patch. In the graph cut version of this problem, the selected path will run *between* pairs of pixels. The simplest quality measure, then, will be a measure of color difference between the pairs of pixels. Let s and t be two adjacent pixel positions in the overlap region. Also, let $\mathbf{A}(s)$ and $\mathbf{B}(s)$ be the pixel colors at the position s in the old and new patches, respectively. We define the matching quality cost M between the two adjacent pixels s and t that copy from patches $\mathbf{A}$ and $\mathbf{B}$ respectively to be:

$$M(s,t,\mathbf{A},\mathbf{B}) = \|\mathbf{A}(s)-\mathbf{B}(s)\| + \|\mathbf{A}(t)-\mathbf{B}(t)\| \qquad (1)$$

where $\|\cdot\|$ denotes an appropriate norm. We consider a more sophisticated cost function in a later section. For now, this match cost is all we need to use graph cuts to solve the path finding problem.

Consider the graph shown in Figure 2(right) that has one node per pixel in the overlap region between patches. We wish to find a low-cost path through this region from top to bottom. This region is shown as 3×3 pixels in the figure, but it is usually more like 8×32 pixels in typical image quilting problems (the overlap between two 32×32 patches). The arcs connecting the adjacent pixel nodes s and t are labelled with the matching quality cost $M(s,t,\mathbf{A},\mathbf{B})$. Two additional nodes are added, representing the old and new patches (**A** and **B**). Finally, we add arcs that have infinitely high costs between some of the pixels and the nodes **A** or **B**. These are *constraint* arcs, and they indicate pixels that we insist will come from one particular patch. In Figure 2, we have constrained pixels 1, 2, and 3 to come from the old patch, and pixels 7, 8, and 9 must come from the new patch. To find out which patch each of the pixels 4, 5, and 6 will come from is determined by solving a graph cut problem. Specifically, we seek the minimum cost cut of the graph, that separates node **A** from node **B**. This is a classical graph problem called min-cut or max-flow [Ford and Fulkerson 1962; Sedgewick 2001] and algorithms for solving it are well understood and easy to code. In the example of Figure 2, the red line shows the minimum cut, and this means pixel 4 will be copied from patch **B** (since its portion of the graph is still connected to node **B**), whereas pixels 5 and 6 will be from the old patch **A**.

3.1 Accounting for Old Seams

The above example does not show the full power of using graph cuts for texture synthesis. Suppose that several patches have already been placed down in the output texture, and that we wish to lay down a new patch in a region where multiple patches already meet. There is a potential for visible seams along the border between old patches, and we can measure this using the arc costs from the graph cut problem that we solved when laying down these patches. We can

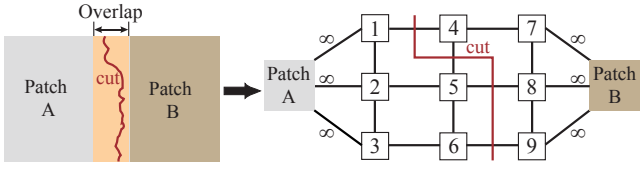

Figure 2: (Left) Schematic showing the overlapping region between two patches. (Right) Graph formulation of the seam finding problem, with the red line showing the minimum cost cut.

incorporate these old seam costs into the new graph cut problem, and thus we can determine which pixels (if any) from the new patch should cover over some of these old seams. To our knowledge, this cannot be done using dynamic programming – the old seam and its cost at each pixel needs to be *remembered*; however, dynamic programming is a memoryless optimization procedure in the sense that it cannot keep track of old solutions.

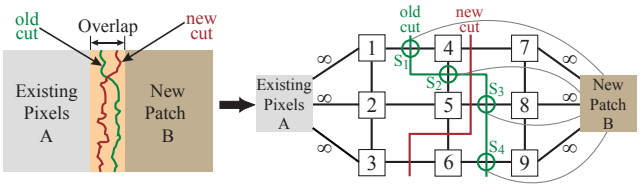

Figure 3: (Left) Finding the best new cut (red) with an old seam (green) already present. (Right) Graph formulation with old seams present. Nodes s_1 to s_4 and their arcs to **B** encode the cost of the old seam.

We illustrate this problem in Figure 3. In the graph formulation of this problem, all of the old patches are represented by a single node **A**, and the new patch is **B**. Since **A** now represents a collection of patches, we use $\mathbf{A}_s$ to denote the particular patch that pixel s copies from. For each seam between old pixels, we introduce a *seam node* into the graph between the pair of pixel nodes. We connect each seam node with an arc to the new patch node **B**, and the cost of this arc is the old matching cost when we created this seam, *i.e.,* $M(s,t,\mathbf{A}_s,\mathbf{A}_t)$ where s and t are the two pixels that straddle the seam. In Figure 3, there is an old seam between pixels 1 and 4, so we insert a seam node s_1 between these two pixel nodes. We also connect s_1 to the new patch node **B**, and label this arc with the old matching cost $M(1,4,\mathbf{A}_1,\mathbf{A}_4)$. We label the arc from pixel node 1 to s_1 with the cost $M(1,4,\mathbf{A}_1,\mathbf{B})$ (the matching cost when only pixel 4 is assigned the new patch) and the arc from s_1 to pixel node 4 with the cost $M(1,4,\mathbf{B},\mathbf{A}_4)$ (the matching cost when only pixel 1 is assigned the new patch). If the arc between a seam node and the new patch node **B** is cut, this means that the old seam remains in the output image. If such an arc is *not* cut, this means that the seam has been overwritten by new pixels, so the old seam cost is not counted in the final cost. If one of the arcs between a seam node and the pixels adjacent to it is cut, it means that a new seam has been introduced at the same position and a new seam cost (depending upon which arc has been cut) is added to the final cost. In Figure 3, the red line shows the final graph cut: the old seam at s_3 has been replaced by a new seam, the seam at s_4 has disappeared, and fresh seams have been introduced between nodes 3 and 6, 5 and 6, and 4 and 7.

This equivalence between seam cost and the min-cut of the graph holds if and only if at most one of the three arcs meeting at the seam nodes is included in the min-cut. The cost of this arc is the new seam cost, and if no arc is cut, the seam is removed and the cost goes to zero. This is true only if we ensure that M is a metric (satisfies the triangle inequality) [Boykov et al. 1999], which is true if the norm in Equation (1) is a metric. Satisfying the triangle inequality implies that picking two arcs originating from a seam node is always costlier than picking just one of them, hence at most one arc is picked in the min-cut, as desired. Our graph cut formulation is equivalent to the one in [Boykov et al. 1999] and the addition of patches corresponds to the α-expansion step in their work. In fact, our implementation uses their code for computing the graph min-cut. Whereas they made use of graph cuts for image noise removal and image correspondence for stereo, our use of graph cuts for texture synthesis is novel.

3.2 Surrounded Regions

So far we have shown new patches that overlap old pixels only along a border region. In fact, it is quite common in our synthesis approach to attempt to place a new patch over a region where the entire area has already been covered by pixels from earlier patch placement steps. This is done in order to overwrite potentially visible seams with the new patch, and an example of this is shown in Figure 4. The graph formulation of this problem is really the same as the problem of Figure 3. In this graph cut problem, all of the pixels in a border surrounding the placement region are constrained to come from existing pixels. These constraints are reflected in the arcs from the border pixels to node **A**. We have also placed a single constraint arc from one interior pixel to node **B** in order to force at least one pixel to be copied from patch **B**. In fact, this kind of a constraint arc to patch **B** isn't even required. To avoid clutter in this figure, the nodes and arcs that encode old seam costs have been omitted. These omitted nodes make many connections between the central portion of the graph and node **B**, so even if the arc to **B** were removed, the graph would still be connected. In the example, the red line shows how the resulting graph cut actually forms a closed loop, which defines the best irregularly-shaped region to copy into the output image.

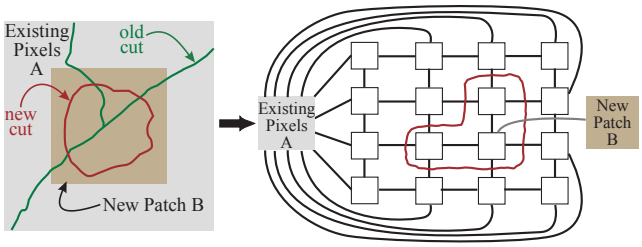

Figure 4: (Left) Placing a patch surrounded by already *filled* pixels. Old seams (green) are partially overwritten by the new patch (bordered in red). (Right) Graph formulation of the problem. Constraint arcs to **A** force the border pixels to come from old image. Seam nodes and their arcs are not shown in this image for clarity.

Finding the best cut for a graph can have a worst-case $O(n^2)$ cost for a graph with n nodes [Sedgewick 2001]. For the kinds of graphs we create, however, we never approach this worst-case behavior. Our timings appear to be $O(n\log(n))$.

4 Patch Placement & Matching

Now we describe several algorithms for picking candidate patches. We use one of three different algorithms for patch selection, based on the type of texture we are synthesizing. These selection methods are: (1) *random placement*, (2) *entire patch matching*, and (3) *sub-patch matching*.

In all these algorithms, we restrict the patch selection to previously unused offsets. Also, for the two matching-based algorithms, we first find a region in the current texture that needs a lot of improvement. We use the cost of existing seams to quantify the error in a particular region of the image, and pick the region with the largest error. Once we pick such an *error-region*, we force the patch selection algorithm to pick only those patch locations that completely overlap the error-region. When the texture is being initialized, *i.e.,* when it is not completely covered with patches of input texture, the error-region is picked differently and serves a different purpose: it is picked so that it contains both initialized and uninitialized portions of the output texture – this ensures that the texture is extended by some amount and also that the extended portion is consistent with the already initialized portions of the texture.

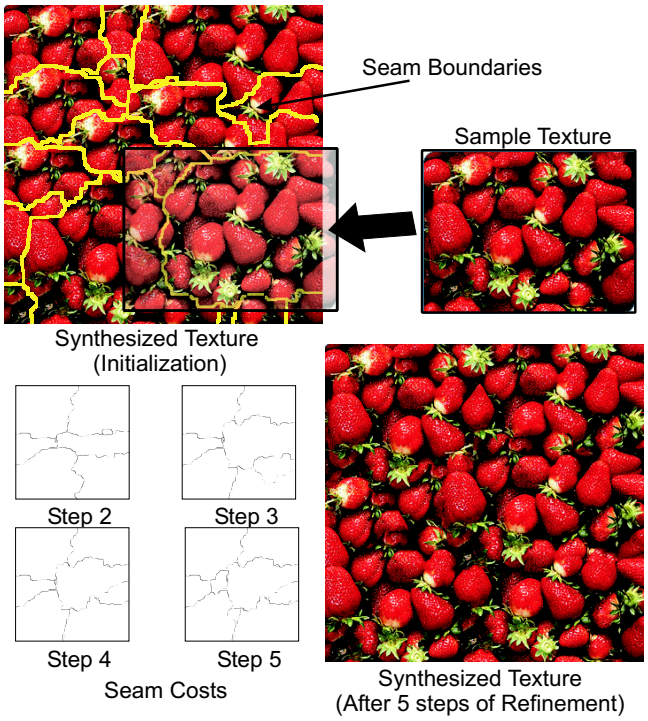

Figure 5: This figure illustrates the process of synthesizing a larger texture from an example input texture. Once the texture is initialized, we find new patch locations appropriately so as to refine the texture. Note the irregular patches and seams. Seam error measures that are used to guide the patch selection process are shown. This process is also shown in the video.

Now we discuss the three patch placement and matching methods in some detail. The same three placement algorithms are used for synthesis of image (spatial) and video (spatio-temporal) textures, discussed in Sections 6 and 7 respectively. Note that patch placement is really just a translation applied to the input before it is added to the output.

Random placement: In this approach, the new patch, (the entire input image), is translated to a random offset location. The graph cut algorithm selects a piece of this patch to lay down into the output image, and then we repeat the process. This is the fastest synthesis method and gives good results for random textures.

Entire patch matching: This involves searching for translations of the input image that match well with the currently synthesized output. To account for partial overlaps between the input and the output, we normalize the sum-of-squared-differences (SSD) cost with the area of the overlapping region. We compute this cost for all possible translations of the input texture as:

$$C(t) = \frac{1}{|\mathbf{A}_t|} \sum_{p \in \mathbf{A}_t} |\mathbf{I}(p) - \mathbf{O}(p+t)|^2 \qquad (2)$$

where $C(t)$ is the cost at translation t of the input, **I** and **O** are the input and output images respectively, and $\mathbf{A}_t$ is the portion of the translated input overlapping the output. We pick the new patch location stochastically from among the possible translations according to the probability function:

$$P(t) \propto e^{-\frac{C(t)}{k\sigma^2}} \quad (3)$$

where σ is the standard deviation of the pixel values in the input image and k controls the randomness in patch selection. A low value of k leads to picking of only those patch locations that have a very good match with the output whereas a high value of k leads to more random patch selection. In our implementation, we varied k between 0.001 and 1.0. Note that we also constrain the selected patch to overlap the error-region as described above. This method gives the best results for structured and semi-structured textures since their inherent periodicity is captured very well by this cost function.

Sub-patch matching: This is the most general of all our patch selection techniques. It is also the best method for stochastic textures as well as for video sequences involving textural motion such as water waves and smoke (Section 7). The motion in such sequences is spatially quite unstructured with different regions of the image exhibiting different motions; however, the motion itself is structured in that it is locally coherent. In sub-patch matching, we first pick a small sub-patch (which is usually significantly smaller than the input texture) in the output texture. In our implementation, this *output-sub-patch* is the same or slightly larger than the error-region that we want to place the patch over. We now look for a sub-patch in the input texture that matches this output-sub-patch well. Equivalently, we look for translations of the input such that the portion of the input overlapping the output-sub-patch matches it well – only those translations that allow complete overlap of the input with the output-sub-patch are considered. The cost of a translation of the input texture is now defined as:

$$C(t) = \sum_{p \in \mathbf{S_O}} |\mathbf{I}(p-t) - \mathbf{O}(p)|^2 \quad (4)$$

where $\mathbf{S_O}$ is the output-sub-patch and all other quantities are the same as in (2). The patch is again picked stochastically using a probability function similar to (3).

5 Extensions & Refinements

Now we briefly describe a few improvements and extensions that we have implemented for image and video synthesis. These extensions include improvements to the cost functions that account for frequency variations, inclusion of feathering and multi-resolution techniques to smooth out visible artifacts, and speed ups in the SSD-based algorithms used in patch matching.

Adapting the Cost Function: The match cost in Equation (1) used in determining the graph cut does not pay any attention to the frequency content present in the image or video. Usually, discontinuities or seam boundaries are more prominent in low frequency regions rather than high frequency ones. We take this into account by computing the gradient of the image or video along each direction – horizontal, vertical and temporal (in case of video) – and scale the match cost in (1) appropriately, resulting in the following new cost function.

$$M'(s,t,\mathbf{A},\mathbf{B}) = \frac{M(s,t,\mathbf{A},\mathbf{B})}{\|\mathbf{G}^d_{\mathbf{A}}(s)\| + \|\mathbf{G}^d_{\mathbf{A}}(t)\| + \|\mathbf{G}^d_{\mathbf{B}}(s)\| + \|\mathbf{G}^d_{\mathbf{B}}(t)\|} \quad (5)$$

Here, d indicates the direction of the gradient and is the same as the direction of the edge between s and t. $\mathbf{G}^d_{\mathbf{A}}$ and $\mathbf{G}^d_{\mathbf{B}}$ are the gradients in the patches $\mathbf{A}$ and $\mathbf{B}$ along the direction d. M' penalizes seams going through low frequency regions more than those going through high frequency regions, as desired.

Feathering and multi-resolution splining: Although graph cuts produce the best possible seam around a given texture patch, it can still generate visible artifacts when no good seam exists at that point. It is possible to hide these artifacts by feathering the pixel values across seams. For every pixel s close enough to a seam, we find all patches meeting at that seam.

The pixel s is then assigned the weighted sum of pixel values $\mathbf{P}(s)$ corresponding to each such patch $\mathbf{P}$. Most of the time, this form of feathering is done using a Gaussian kernel.

We also use multi-resolution splining [Burt and Adelson 1983] of patches across seams, which is helpful when the seams are too obvious, but it also tends to reduce the contrast of the image or video when a lot of small patches have been placed in the output. In general, we have found it useful to pick between feathering and multi-resolution splining on a case-by-case basis.

FFT-Based Acceleration: The SSD-based algorithms described in Section 4 can be computationally expensive if the search is carried out naively. Computing the cost $C(t)$ for all valid translations is $O(n^2)$ where n is the number of pixels in the image or video. However, the search can be accelerated using *Fast Fourier Transforms (FFT)* [Kilthau et al. 2002; Soler et al. 2002] . For example, we can rewrite (4) as:

$$C(t) = \sum_p \mathbf{I}^2(p-t) + \sum_p \mathbf{O}^2(p) - 2\sum_p \mathbf{I}(p-t)\mathbf{O}(p) \quad (6)$$

The first two terms in (6) are sum of squares of pixel values over sub-blocks of the image or video and can be computed efficiently in $O(n)$ time using summed-area tables [Crow 1984]. The third term is a convolution of the input with the output and can be computed in $O(n\log(n))$ time using FFT. Since n is extremely large for images and especially for video, we get a huge speed up using this method – for a $150 \times 100 \times 30$ video sequence, $n \approx 10^6$, and the time required to search for a new patch reduces from around *10 minutes* (using naive search) to *5 seconds* (using FFT-based search).

6 Image Synthesis

We have applied our technique for image and texture synthesis to generate regular, structured and random textures as well as to synthesize extensions of natural images. Figures 8, 9, and 10 show results for a variety of two-dimensional image textures. We used *entire patch matching* as our patch selection algorithm for the TEXT, NUTS, ESCHER, and KEYBOARD images, while *sub-patch matching* was used for generating CHICK PEAS, MACHU PICCHU, CROWDS, SHEEP, OLIVES, BOTTLES, and LILIES. The computation for images is quite fast, mainly due to the use of FFT-based search. All image synthesis results presented here took between 5 seconds and 5 minutes to run. The LILIES image took 5 minutes because it was originally generated to be 1280×1024 in size. We also compare some of our results with that of Image Quilting [Efros and Freeman 2001] in Figure 9. Now we briefly describe a few specialized extensions and applications of our 2D texture synthesis technique.

A. Additional Transformations of Source Patches: Our algorithm relies on placing the input patch appropriately and determining a seam that supports efficient patching of input images. Even though we have only discussed the possibility of translating the input patch over the output region, one could generalize this concept to include other *transformations* of the input patch like rotation, scaling, affine or projective. For images, we have experimented with the use of *rotated*, *mirrored*, and *scaled* versions of the input texture. Allowing more transformations gives us more flexibility and variety in terms of the kind of output that can be generated. However, as we increase the potential transformations

of the input texture, the cost of searching for good transformations also increases. Therefore, we restrict the transformations other than translations to a small number. Note that the number of candidate translations is of the order of the number of pixels. We generate the transformed versions of the input before we start synthesis. To avoid changing the searching algorithm significantly, we put all the transformed images into a single image juxtaposed against each other. This makes the picking of any transformation equivalent to the picking of a translation. Then, only the portion containing the particular transformed version of the image is sent to the graph cut algorithm instead of the whole mosaic of transformations.

In Figure 9, we make use of rotational and mirroring transformations to reduce repeatability in the synthesis of the OLIVES image. Scaling allows mixing different sizes of texture elements together. One interesting application of scaling is to generate images conveying deep perspective. We can constrain different portions of the output texture to copy from different scales of the input texture. If we force the scale to vary in a monotonic fashion across the output image, it gives the impression of an image depicting perspective. For example, see BOTTLES and LILIES in Figure 10.

B. Interactive Merging and Blending: One application of the graph cut technique is interactive image synthesis along the lines of [Mortensen and Barrett 1995; Brooks and Dodgson 2002]. In this application, we pick a set of source images and combine them to form a single output image. As explained in the section on patch fitting for texture synthesis (Section 3), if we constrain some pixels of the output image to come from a particular patch, then the graph cut algorithm finds the best seam that goes through the remaining unconstrained pixels. The patches in this case are the source images that we want to combine. For merging two such images, the user first specifies the locations of the source images in the output and establishes the constrained pixels interactively. The graph cut algorithm then finds the best seam between the images.

This is a powerful way to combine images that are not similar to each other since the graph cut algorithm finds any regions in the images that *are* similar and tries to make the seam go through those regions. Note that our cost function as defined in Equation (5) also favors the seam to go through edges. Our results indicate that both kinds of seams are present in the output images synthesized in this fashion. In the examples in Figure 11, one can see that the seam goes through (a) the middle of the water which is the region of similarity between the source images, and (b) around the silhouettes of the people sitting in the raft which is a high gradient region.

The SIGGRAPH banner on the title page of this paper was generated by combining flowers and leaves interactively: the user had to place a flower image over the leaves background and constrain some pixels of the output to come from within a flower. The graph cut algorithm was then used to compute the appropriate seam between the flower and the leaves automatically. Each letter of the word SIGGRAPH was synthesized individually and then these letters were combined, again using graph cuts, to form the final banner – the letter G was synthesized only once, and repeated. Approximate interaction time for each letter was in the range of 5-10 minutes. The source images for this example are in Figure 6.

It is worthwhile to mention related work on Intelligent Scissors by Mortensen and Barrett [1995] in this context. They follow a two-step procedure of segmentation followed by composition to achieve similar effects. However, in our work, we don't segment the objects explicitly; instead we leave it to the cost function to choose between object boundaries and perceptually similar regions for the seam to go through. Also, the cost function used by them is different than ours.

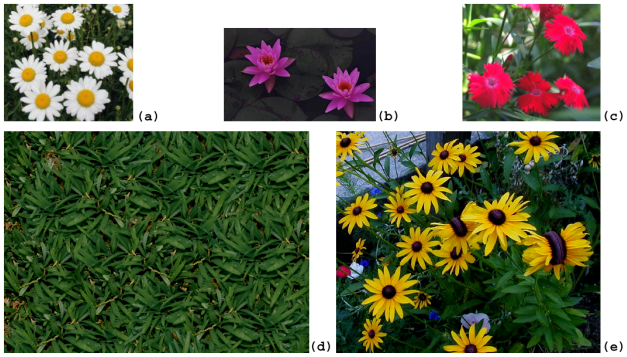

Figure 6: The source images used to generate the SIGGRAPH banner on the title page of this paper. Image credits: (b)©East West Photo, (c)©Jens Grabenstein, (e)©Olga Zhaxybayeva.

7 Video Synthesis

One of the main strengths of the graph cut technique proposed here is that it allows for a straightforward extension to video synthesis. Consider a video sequence as a 3D collection of voxels, where one of the axes is time. Patches in the case of video are then the whole 3D space-time blocks of video, which can be placed anywhere in the 3D (space-time) volume. Hence, the same two steps from image texture synthesis, patch placement and seam finding, are also needed for video texture synthesis.

Similar to 2D texture, the patch selection method for video must be chosen based on the type of video. Some video sequences just show temporal stationarity whereas others show stationarity in space as well as time. For the ones showing only temporal stationarity, searching for patch translations in all three dimensions (space and time) is unnecessary. We can restrict our search just to patch offsets in time, *i.e.,* we just look for temporal translations of the patch. However, for videos that are spatially and temporally stationary, we do search in all three dimensions.

We now describe some of our video synthesis results. We start by showing some examples of temporally stationary textures in which we find spatio-temporal seams for video transitions. These results improve upon video textures [Schödl et al. 2000] and compare favorably against dynamic textures [Soatto et al. 2001]. Then we discuss spatio-temporally stationary type of video synthesis that improves upon [Wei and Levoy 2000; Bar-Joseph et al. 2001]. All videos are available off our web page and/or included in the DVD.

A. Finding Seams for Video Transitions: Video textures [Schödl et al. 2000] turn existing video into an infinitely playing form by finding smooth *transitions* from one part of the video to another. These transitions are then used to infinitely loop the input video. This approach works only if a pair of similar-looking frames can be found. Many natural processes like fluids and small-scale motion are too chaotic for any frame to reoccur. To ease visual discontinuities due to frame mismatches, video textures used blending and morphing techniques. Unfortunately, a blend between transitions introduces an irritating blur. Morphing also does not work well for chaotic motions because it is hard to find corresponding features. Our seam optimization allows for a more sophisticated approach: the two parts of the video interfacing at a transition, represented by two 3D spatio-temporal texture patches, can be spliced together by computing the optimal seam between the two 3D patches. The seam in this case is actually a 2D surface that sits in 3D (Figure 7).

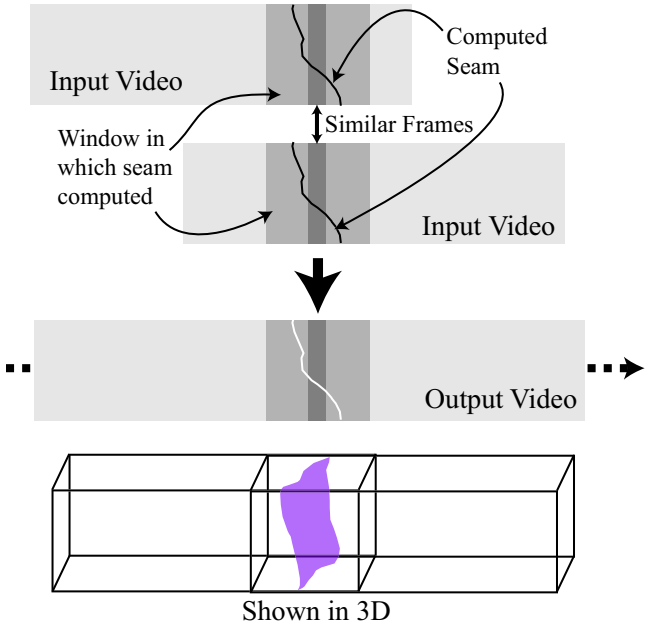

Figure 7: Illustration of seams for temporal texture synthesis. Seams shown in 2D and 3D for the case of video transitions. Note that the seam is a surface in case of video.

To find the best relative offset of the spatio-temporal texture patches, we first find a good transition by pair-wise image comparison as described in [Schödl et al. 2000]. We then compute an optimal seam for a limited number of good transitions within a window around the transition. The result is equivalent to determining the time of the transition on a per-pixel basis rather than finding a single transition time for the whole image. The resulting seam can then be repeated to form a video loop as shown in Figure 7.

We have generated several (infinitely long) videos using this approach. For each sequence, we compute the optimal seam within a 60-frame spatio-temporal window centered around the best transition. Examples include WATERFALL A, GRASS, POND, FOUNTAIN, and BEACH. WATERFALL A and GRASS have been borrowed from Schödl et al. [2000]. Their results on these sequences look intermittently blurred during the transition. Using our technique, we are able to generate sequences without any perceivable artifacts around the transitions, which eliminates the need for any blurring. We have also applied (our implementation of) dynamic textures [Soatto et al. 2001] to WATERFALL A, the result of which is much blurrier than our result. The BEACH example shows the limitations of our approach. Although the input sequence is rather long – 1421 frames – even the most similar frame pair does not allow a smooth transition. During the transition, a wave gradually disappears. Most disconcertingly, parts of the wave vanish from bottom to top, defying the usual dynamics of waves.

B. Random Temporal Offsets For very short sequences of video, looping causes very noticeable periodicity. In this case, we can synthesize a video by applying a series of input texture patches, which are randomly displaced in time. The seam is computed within the whole spatio-temporal volume of the input texture.

We have applied this approach to FIRE, SPARKLE, OCEAN, and SMOKE. The result for FIRE works relatively well and, thanks to random input patch displacements, is less repetitive than the comparable looped video. The SPARKLE result is also very nice, although electric sparks sometimes detach from the ball. In the case of OCEAN, the result is overall good, but the small amount of available input footage causes undesired repetitions. SMOKE is a failure of this method. There is no continuous part in this sequence that tiles well in time. Parts of the image appear almost static. The primary reason for this is the existence of a dominant direction of motion in the sequence, which is very hard to capture using temporal translations alone. Next, we discuss how to deal with such textures using spatio-temporal offsets.

C. Spatio-Temporally Stationary Videos: For videos that show spatio-temporal stationarity (like OCEAN and SMOKE), only considering translations in time does not produce good results. This is because, for such sequences, there usually is some dominant direction of motion for most of the spatial elements, that cannot be captured by just copying pixels from different temporal locations; we need to move the pixels around in both space and time. We apply the sub-patch matching algorithm in 3D for spatio-temporal textures. Using such translations in space and time for spatio-temporal textures shows a remarkable improvement over using temporal translations alone.

The OCEAN sequence works very well with this approach, and the motion of the waves is quite natural. However, there are still slight problems with sea grass appearing and disappearing on the surface of the water. Even SMOKE shows a remarkable improvement. It is no longer static, as was the case with the previous method, showing the power of using spatio-temporal translations. RIVER, FLAME, and WATERFALL B also show very good results with this technique.

We have compared our results for FIRE, OCEAN, and SMOKE with those of Wei and Levoy [2000] – we borrowed these sequences and their results from Wei and Levoy's web site. They are able to capture the local statistics of these temporal textures quite well but fail to reproduce their global structure. Our results show an improvement over them by being able to reproduce both the local and the global structure of the phenomena.

D. Temporal Constraints for Video One of the advantages of constraining new patch locations for video to temporal translations is that even though we find a spatio-temporal seam within a window of a few frames, the frames outside that window stay as is. This allows us to loop the video infinitely (see Figure 7). When we allow the translations to be in both space and time, this property is lost and it is non-trivial to make the video loop. It turns out, however, that we can use the graph cut algorithm to perform constrained synthesis (as in the case of interactive image merging) and therefore looping. We fix the first k and last k frames of the output sequence to be the same k frames of the input ($k = 10$ in our implementation). The pixels in these frames are now constrained to stay the same. This is ensured by adding links of infinite cost between these pixels and the patches they are constrained to copy from, during graph construction. The graph cut algorithm then computes the best possible seams given that these pixels don't change. Once the output has been generated, we remove the first k frames from it. This ensures a video loop since the k^{th} frame of the output is the same as its last frame before this removal operation.

Using this technique, we have been able to generate looped sequences for almost all of our examples. One such case is WATERFALL B, which was borrowed from [Bar-Joseph et al. 2001]. We are able to generate an infinitely long sequence for this example, where as [Bar-Joseph et al. 2001] can extend it to a finite length only. SMOKE is one example for which looping does not work very well.

E. Spatial Extensions for Video We can also increase the frame-size of the video sequence if we allow the patch translations

to be in both space and time. We have been able to do so successfully for video sequences exhibiting spatio-temporal stationarity. For example, the spatial resolution of the RIVER sequence was increased from 170×116 to 210×160. By using temporal constraints, as explained in the previous paragraph, we were even able to loop this *enlarged* video sequence.

The running times for our video synthesis results ranged from 5 minutes to 1 hour depending on the size of video and the search method employed – searching for purely temporal offsets is faster than that for spatio-temporal ones. The use of FFT-based acceleration in our search algorithms was a huge factor in improving efficiency.

8 Summary

We have demonstrated a new algorithm for image and video synthesis. Our graph cut approach is ideal for computing seams of patch regions and determining placement of patches to generate perceptually smooth images and video. We have shown a variety of synthesis examples that include structured and random image and video textures. We also show extensions that allow for transformations of the input patches to permit variability in synthesis. We have also demonstrated an application that allows for merging of two different source images interactively. In general, we believe that our technique significantly improves upon the state of the art in texture synthesis by providing the following benefits: (a) no restrictions on shape of the region where seam will be created, (b) consideration of old seam costs, (c) easy generalization to creation of seam surfaces for video, and (d) a simple method for adding constraints.

Acknowledgements

We would like to thank Professor Ramin Zabih and his students at Cornell University for sharing their code for computing graph min-cut [Boykov et al. 1999]. Thanks also to the authors of [Efros and Freeman 2001], [Wei and Levoy 2000] and [Bar-Joseph et al. 2001] for sharing their raw footage and results via their respective web sites to facilitate direct comparisons. Thanks also to Rupert Paget for maintaining an excellent web page on texture research and datasets, and to Martin Szummer for the MIT temporal texture database. Thanks also to the following for contributing their images or videos: East West Photo, Jens Grabenstein, Olga Zhaxybayeva, Adam Brostow, Brad Powell, Erskine Wood, Tim Seaver and Artbeats Inc.. Finally, we would like to thank our collaborators Gabriel Brostow, James Hays, Richard Szeliski and Stephanie Wojtkowski for their insightful suggestions and help with the production of this work. Arno Schödl is now at think-cell Software GmbH in Berlin, Germany.

This work was funded in part by grants from NSF (IIS-9984847, ANI-0113933 and CCR-0204355) and DARPA (F49620-001-0376), and funds from Microsoft Research.

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. *2001 ACM Symposium on Interactive 3D Graphics* (March), 217–226. ISBN 1-58113-292-1.

BAR-JOSEPH, Z., EL-YANIV, R., LISCHINSKI, D., AND WERMAN, M. 2001. Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics 7*, 2, 120–135.

BOYKOV, Y., VEKSLER, O., AND ZABIH, R. 1999. Fast approximate energy minimization via graph cuts. In *International Conference on Computer Vision*, 377–384.

BROOKS, S., AND DODGSON, N. A. 2002. Self-similarity based texture editing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002) 21*, 3 (July), 653–656.

BURT, P. J., AND ADELSON, E. H. 1983. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics 2*, 4, 217–236.

CROW, F. C. 1984. Summed-area tables for texture mapping. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, 207–212. ISBN 0-89791-138-5.

DEBONET, J. S. 1997. Multiresolution sampling procedure for analysis and synthesis of texture images. *Proceedings of SIGGRAPH 97* (August), 361–368. ISBN 0-89791-896-7. Held in Los Angeles, California.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001* (August), 341–346. ISBN 1-58113-292-1.

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

FORD, L., AND FULKERSON, D. 1962. *Flows in Networks*. Princeton University Press.

GREIG, D., PORTEOUS, B., AND SEHEULT, A. 1989. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society Series B, 51*, 271–279.

GUO, B., SHUM, H., AND XU, Y.-Q. 2000. Chaos mosaic: Fast and memory efficient texture synthesis. Tech. Rep. MSR-TR-2000-32, Microsoft Research.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-based texture analysis/synthesis. *Proceedings of SIGGRAPH 95* (August), 229–238. ISBN 0-201-84776-0. Held in Los Angeles, California.

KILTHAU, S.L., DREW, M., AND MOLLER, T. 2002. Full search content independent block matching based on the fast fourier transform. In *ICIP02*, I: 669–672.

LI, S. Z. 1995. *Markov Random Field Modeling in Computer Vision*. Springer-Verlag.

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.

MORTENSEN, E. N., AND BARRETT, W. A. 1995. Intelligent scissors for image composition. *Proceedings of SIGGRAPH 1995* (Aug.), 191–198.

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.

SAISAN, P., DORETTO, G., WU, Y., AND SOATTO, S. 2001. Dynamic texture recognition. In *Proceeding of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, II:58–63.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. *Proceedings of SIGGRAPH 2000* (July), 489–498. ISBN 1-58113-208-5.

SEDGEWICK, R. 2001. *Algorithms in C, Part 5: Graph Algorithms*. Addison-Wesley, Reading, Massachusetts.

SOATTO, S., DORETTO, G., AND WU, Y. 2001. Dynamic textures. In *Proceeding of IEEE International Conference on Computer Vision 2001*, II: 439–446.

SOLER, C., CANI, M.-P., AND ANGELIDIS, A. 2002. Hierarchical pattern mapping. *ACM Transactions on Graphics 21*, 3 (July), 673–680.

SZUMMER, M., AND PICARD, R. 1996. Temporal texture modeling. In *Proceeding of IEEE International Conference on Image Processing 1996*, vol. 3, 823–826.

WANG, Y., AND ZHU, S. 2002. A generative method for textured motion: Analysis and synthesis. In *European Conference on Computer Vision*.

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

Figure 8: 2D texture synthesis results. We show results for textured and natural images. The smaller images are the example images used for synthesis. Shown are CHICK PEAS, TEXT, NUTS, ESCHER, MACHU PICCHU©Adam Brostow, CROWDS and SHEEP from left to right and top to bottom.

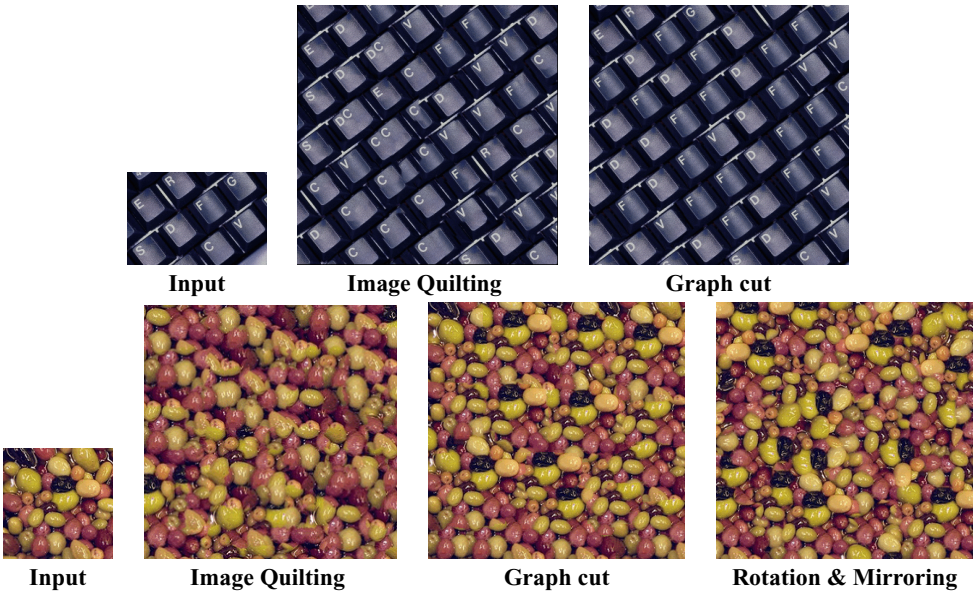

Figure 9: Comparison of our graph cut algorithm with Image Quilting [Efros and Freeman 2001]. Shown are KEYBOARD and OLIVES. For OLIVES, an additional result is shown that uses rotation and mirroring of patches to increase variety. The quilting result for KEYBOARD was generated using our implementation of Image Quilting; the result for OLIVES is courtesy of Efros and Freeman.

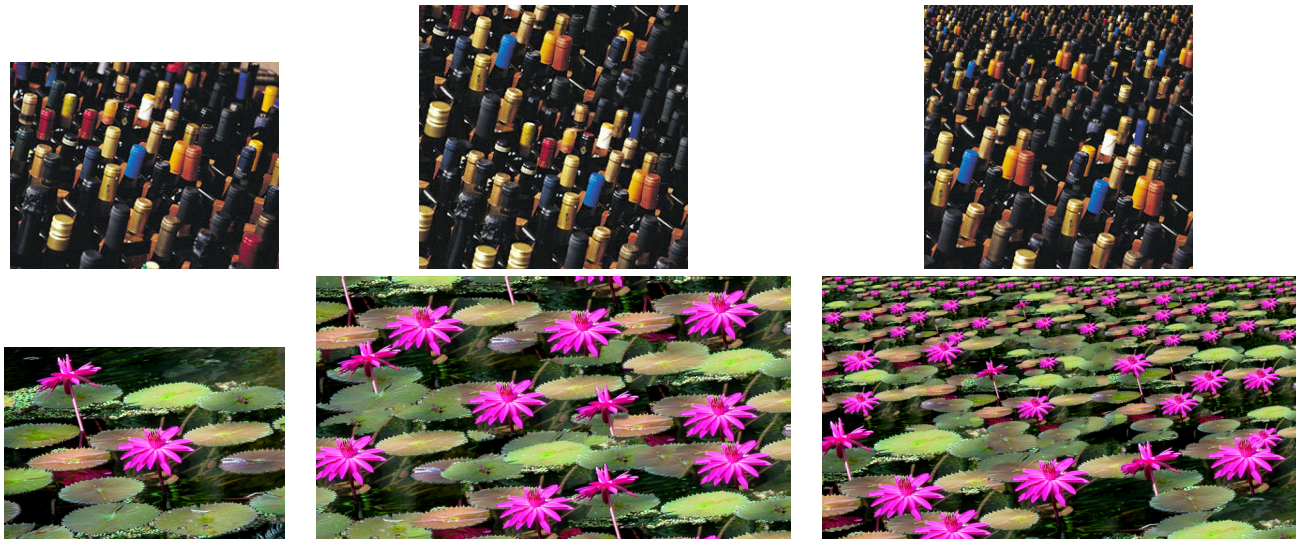

Figure 10: Images synthesized using multiple scales yield perspective effects. Shown are BOTTLES and LILIES©Brad Powell. We have, from left to right: input image, image synthesized using one scale, and image synthesized using multiple scales.

Figure 11: Examples of interactive blending of two source images. Shown are HUT and MOUNTAIN©Erskine Wood in the top row, and RAFT and RIVER©Tim Seaver in the bottom row. The results are shown in the third column. In the last column, the computed seams are also rendered on top of the results.

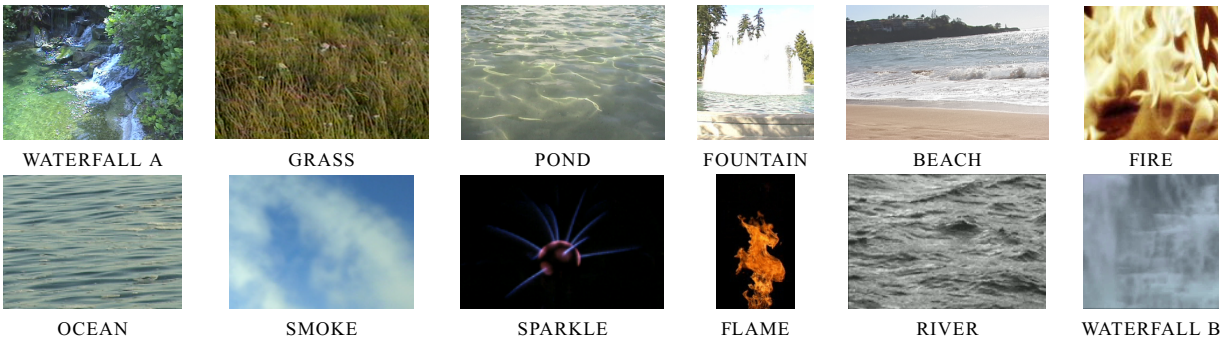

Figure 12: The variety of videos synthesized using our approach.