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

Permission to make digital/hard copy of part of all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 2003 ACM 0730-0301/03/0700-0277 $5.00

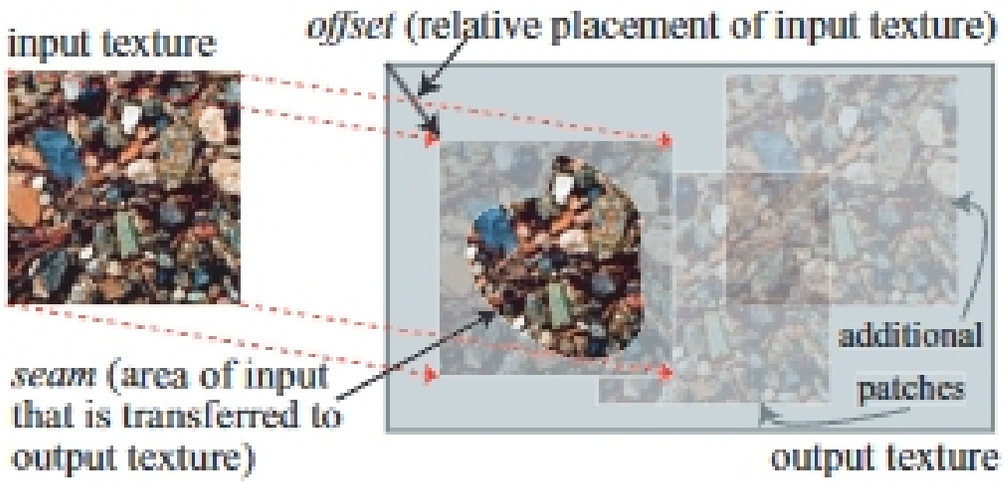

Figure 1: Image texture synthesis by placing small patches at various offsets followed by the computation of a seam that enforces visual smoothness between the existing pixels and the newly placed patch.

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 ($\mathbf{A}$ and $\mathbf{B}$). Finally, we add arcs that have infinitely high costs between some of the pixels and the nodes $\mathbf{A}$ or $\mathbf{B}$. 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 $\mathbf{A}$ from node $\mathbf{B}$. 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 $\mathbf{B}$ (since its portion of the graph is still connected to node $\mathbf{B}$), whereas pixels 5 and 6 will be from the old patch $\mathbf{A}$.

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

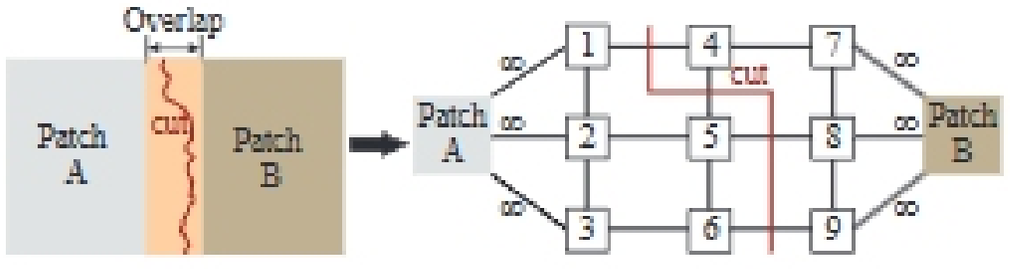

Figure 2: (Left) Schematic showing the overlapping region between two patches. (Right) Graph formulation of the seam finding problem, with the red line showing the minimum cost cut.

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.

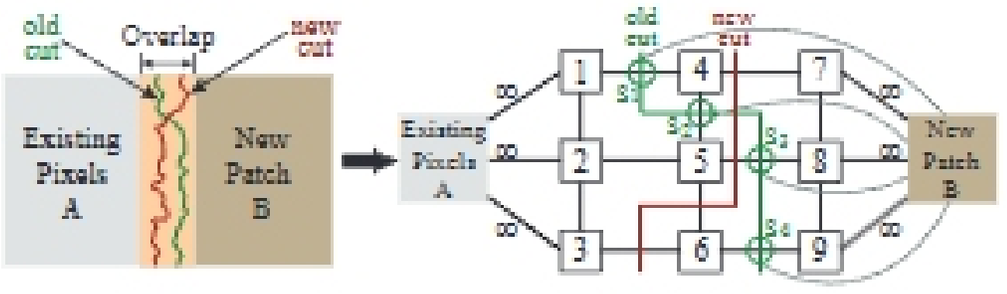

Figure 3: (Left) Finding the best new cut (red) with an old seam (green) already present. (Right) Graph formulation with old seams present. Nodes s_1 to s_4 and their arcs to $\mathbf{B}$ encode the cost of the old seam.

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 $\mathbf{A}$, and the new patch is $\mathbf{B}$. Since $\mathbf{A}$ 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 $\mathbf{B}$, 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 $\mathbf{B}$, 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 $\mathbf{B}$ 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.