Constrained Texture Synthesis via Energy Minimization

Ganesh Ramanarayanan and Kavita Bala, *Member, IEEE*
Department of Computer Science, Cornell University

***Abstract*—**
This paper describes CMS (constrained minimization synthesis), a fast, robust texture synthesis algorithm that creates output textures while satisfying constraints. We show that constrained texture synthesis can be posed in a principled way as an energy minimization problem that requires balancing two measures of quality: constraint satisfaction and texture seamlessness. We then present an efficient algorithm for finding good solutions to this problem using an adaptation of graphcut energy minimization. CMS is particularly well suited to detail synthesis, the process of adding high-resolution detail to low-resolution images. It also supports the full image analogies framework, while providing superior image quality and performance. CMS is easily extended to handle multiple constraints on a single output, thus enabling novel applications that combine both user-specified and image-based control.

***Index Terms*—texture synthesis, detail synthesis, super-resolution, image analogies**

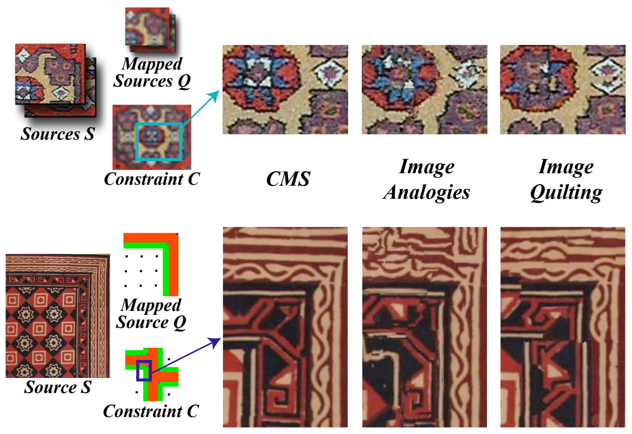


Fig. 1. Results showing detail synthesis (top) and user-controlled texture synthesis for texture-by-numbers (bottom). In both examples, CMS produces output superior to Image Analogies and Image Quilting. *Top*: Carpet. The constraint is a blurry image, and the sources are high resolution texture (left). CMS is able to effectively match the constraint and synthesize plausible high-resolution detail. *Bottom*: Cloth. The user-specified correspondence is a map of texture regions to solid color regions, and the constraint is a rearrangement of these colors (left). CMS is able to seamlessly integrate data from the source image while respecting the color constraints and maintaining local coherence.

I. Introduction

Texture synthesis can decrease onerous modeling tasks by automatically creating textures from examples. Recent advances in texture synthesis technology, such as Graphcut Textures [2], have dramatically improved both texture quality and synthesis performance. However, texture synthesis algorithms are still hard to control; they often fail in undesirable ways, producing unusable output. This has been an obstacle for the adoption of texture synthesis in settings where the user needs the output to have certain properties.

Several authors have recognized the benefit of controlling texture synthesis through analogy [3], [4]. In Image Analogies [3], the user specifies a correspondence in the form of two images A and A'. Then, given a constraint image B, the system tries to find an output B' that is related to B in the same way that A' is related to A. Pixel-based methods are used to compute this analogy, and the technique is applied to a remarkable range of inputs. Image Quilting [4] achieves a similar effect for the specific application of texture transfer by augmenting their image quilting algorithm with a correspondence map. However, these techniques both suffer from two shortcomings. First, it is hard to precisely characterize what kind of image similarity they are aiming for. Second, their output quality and performance can suffer due to limitations in the underlying synthesis and correspondence algorithms.

This paper describes CMS (constrained minimization synthesis), a texture synthesis algorithm that generates constrained output by solving a global optimization problem. CMS is an effort to combine the quality and performance of Graphcut Textures with the power of the Image Analogies framework. Our algorithm is particularly suited to the application of detail synthesis [5] (Figure 1, top) and it supports the wide range of applications of Image Analogies as well, including artistic filtering, texture transfer, and texture-by-numbers (Figure 1, bottom). Constraints can be specified by a user, as in texture-by-numbers, or through a photograph, e.g. a low resolution image for detail synthesis. Additionally, multiple constraints can be combined for greater power, which is a new functionality not demonstrated in previous work.

In this paper we make the following contributions. First, we give a principled formulation of constrained texture synthesis as an energy minimization problem balancing two measures of quality: *constraint matching* and *texture seamlessness*. Second, we demonstrate how to approximate this energy minimization problem and solve it efficiently using an adapted graphcut energy minimization algorithm. Third, in addition to showing our ability to support the general Image Analogies framework, we demonstrate the success of our technique for the particular problem of detail synthesis. We believe that detail synthesis is an important application for image-based modeling, and automated techniques for generating plausible high-resolution detail will become very useful in the future.

e-mail: {graman, kb}@cs.cornell.edu

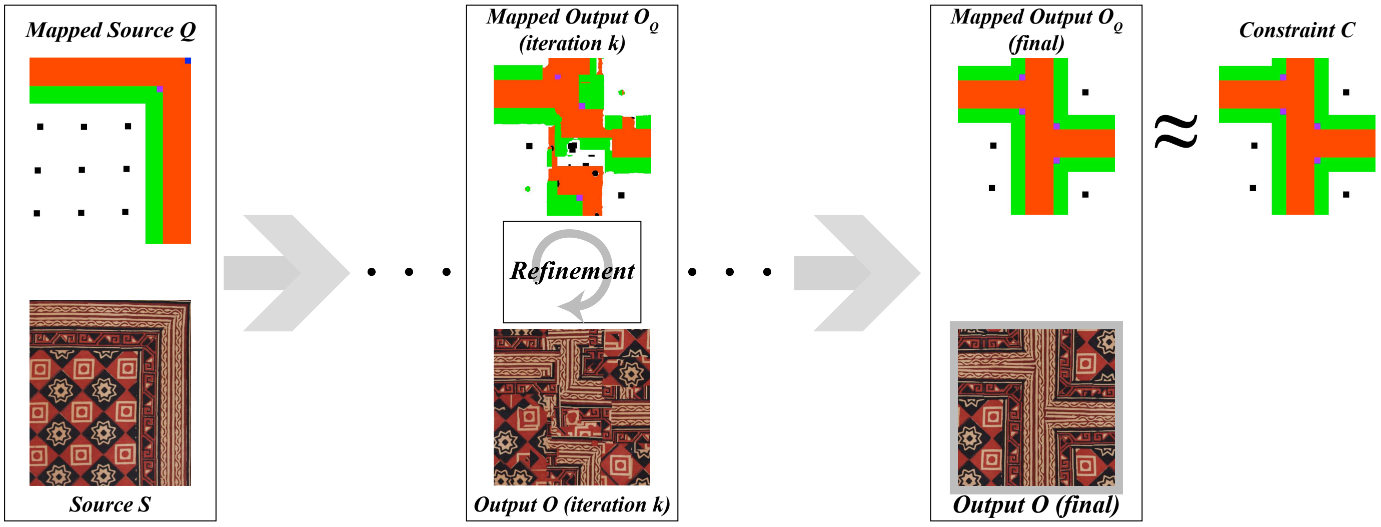


Fig. 2. CMS overview. The user provides input S, and additional inputs Q and C to control the synthesis process. CMS synthesizes from S and Q in parallel, iterating until Q's output O_Q looks like C. When this process is finished, we say the output O has been generated successfully as well.

II. Related Work

Texture synthesis. Texture synthesis algorithms [6] use a small source texture to generate a large output texture that "looks the same". Pixel-based synthesis algorithms [7], [8] grow an output texture pixel-by-pixel, often using scale-space representations to match across different frequency bands. These approaches are quite effective on stochastic textures, but they typically fail on textures with more coherent structure. Patch-based synthesis algorithms [2], [4], [9] copy whole source patches into the output instead of single pixels. Because of this, they tend to be faster and better at capturing visual coherence than pixel-based algorithms. However, when they are forced to paste very small texture patches, their behavior becomes more similar to pixel-based methods. Ashikhmin [10] proposes pixel-by-pixel patch growing, which represents a middle ground between pixel-based and patch-based methods.

In work concurrent with ours, Kwatra et. al. [11] proposes a hybrid method based on energy minimization for unconstrained synthesis, with some extensions for flow-field based constraints. We discuss specific differences between their minimization technique and ours at the end of Section IV-D.

Synthesis with constraints. General constraints to drive texture synthesis have been introduced in pixel-based [3], [10] and patch-based [4] contexts. Ashikhmin [10] proposes user-drawn color constraints as a guide for synthesis, and Image Analogies [3] presents a full framework supporting both user-drawn and image-based constraints. Zhang et. al. [12] use a simplified analogies framework on surfaces to generate textures from texton masks.

Among patch-based analogy techniques, Image Quilting [4] uses a correspondence map and hierarchical patch pasting to compute texture transfer, and Schödl [13] proposes an extension to Graphcut Textures [2] that is related to our work. See Sections V for result comparisons. Other constrained patch-based synthesis techniques include Interactive Digital Photomontage [14] and that of Zhou et. al. [15], where graph-cut formulations are used for user-controlled photomontages and surface BTF painting, respectively. We discuss specific similarities and differences between CMS and all of these algorithms in Section IV-D.

Super-resolution and detail synthesis. Super-resolution and detail synthesis are slightly different approaches to the problem of enhancing a low-resolution image with high-resolution detail. Super-resolution [16] attempts to solve for the *actual* high frequency content in a low-resolution image. Common super-resolution algorithms take a low-resolution video sequence as input and extract additional data for one frame by analyzing its sequence of neighboring frames. Detail synthesis [5], on the other hand, synthesizes *plausible* detail for a single low resolution image, using a couple of small, high resolution samples. Freeman et. al. [17] use a Laplacian-like operation to split training images into low and high frequency, and then add high frequency information to a low-resolution image by quilting patches. The algorithm is promising for subtle edge enhancement at up to 4× linear (16× pixel) zoom, but artifacts can arise due to mismatches between existing data and added high frequencies. Detail synthesis for highly specialized problems has also been attempted for faces [18] and liver cells [19] by using very large sets of training data. Our approach can be useful in these contexts because of its ability to compute dramatic image zoom with just a few sample images.

Graphcut energy minimization. The graphcut energy minimization framework was introduced by Boykov et. al. [20], and it has proven to be a remarkably effective algorithm both in vision [21]–[23] and graphics [2], [14], [15], [24]. Graphcut Textures was one of the first papers to apply this technique to texture synthesis. The CMS algorithm is an extension of the ideas of Graphcut Textures to support analogy applications and constrained synthesis; it is not meant to be used for unconstrained synthesis. For details, refer to Section IV-D.

III. CMS Overview

An overview of CMS is shown in Figure 2. As in normal texture synthesis, the user provides a source image S to synthesize an output O. To control synthesis, the user represents the desired synthesis process in another domain, in the form of two additional input images, the *mapped source* Q and the *constraint* C. For instance, in Figure 2, Q consists of various colored regions corresponding to structures in S, and C is a different arrangement of these colors corresponding to a desired output. The basic idea is to compute the synthesis process that takes Q to C, and apply it to S to get O.

Specifically, CMS takes the image pair $\{S, Q\}$ and synthesizes another image pair $\{O_Q, O\}$. Synthesis is performed *in parallel*; whenever a pixel is copied from S to O, the corresponding pixel from Q is copied into O_Q. If, through multiple synthesis iterations, we can make O_Q look like C, then by analogy we can claim that O has been synthesized properly as well. We also would like O to be visually seamless. Thus the goal of CMS is to take Q, S, and C, and generate an output O such that

- $O_Q = C$ (constraint match)
- O has no visual seams (texture seamlessness)

In this framework, the Image Analogies relationship $A : A' :: B : B'$ is given by $Q : S :: C : O$. We prefer the latter terminology because it emphasizes the specific roles of each image in the algorithm.

Apart from texture-by-numbers, CMS is useful for all other Image Analogies applications, such as artistic filtering and texture transfer. CMS is particularly promising for the application detail synthesis (see Figure 1, top).

IV. Constraint Synthesis via Energy Minimization

The core of CMS is a labeling L that maintains the synthesis moves computed by the algorithm, stored as the pasting of different patches from S. An energy function is defined over L to capture constraint matching and texture seamlessness, and this function is minimized using graphcut energy minimization [20] to generate the final output O. This section explains this construction in detail, starting with an overview of graphcut energy minimization, and ending with a complete description of the algorithm in Section IV-C.

A. Graphcut minimization review

Graphcut minimization [20] can be used to assign labels to a grid of pixels such that the assignment minimizes an energy function of the form

$$E(L) = \sum_p U(p, L(p)) + \sum_{(p,q)} V(p, q, L(p), L(q)) \qquad (1)$$

where p is a pixel location, (p, q) are neighboring pixel locations, and $L(p)$ is the label assigned to p. U, the *assignment cost*, captures how well a label suits a particular pixel. V, the *separation cost*, imposes a penalty if neighboring pixels have incompatible labels. In computer vision problems, labels typically represent disparity or depth. The overall approach is to iteratively improve the labeling L by repeatedly trying

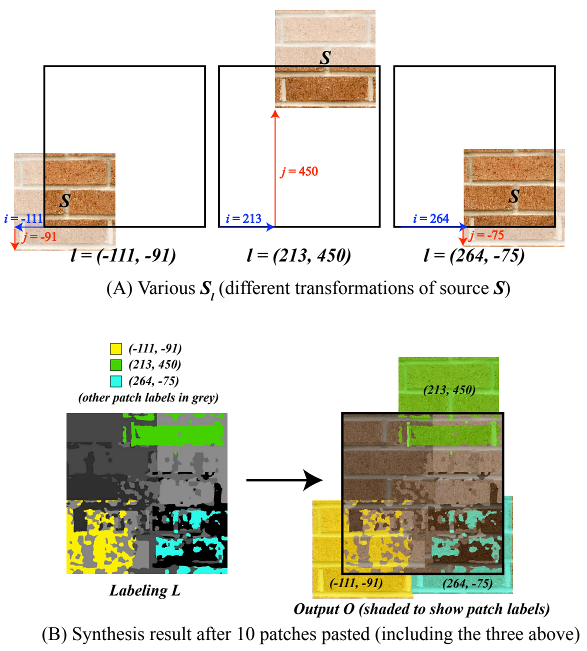

S
i = -111
j = -91
l = (-111, -91)
j = 450
i = 213
l = (213, 450)
i = 264
j = -75
l = (264, -75)
(A) Various S_l (different transformations of source S)
🟨 (-111, -91)
🟩 (213, 450)
🟦 (264, -75)
(other patch labels in grey)
Labeling L
(213, 450)
(-111, -91)
(264, -75)
Output O (shaded to show patch labels)
(B) Synthesis result after 10 patches pasted (including the three above)

Fig. 3. **(A)** Some example labels. The label $l = (i, j)$ corresponds to an image S_l, which represents a translation of S i pixels horizontally and j pixels vertically. **(B)** *Left*: An example labeling, with the labels from (A) marked in color, and all others marked grey. *Right*: the synthesis result, again with corresponding regions marked in color.

many label swapping moves, stopping when the energy cannot be decreased further.

The power of this algorithm comes from the particular swapping moves used to improve the labeling, most notably the α-expansion [20]. An α-expansion permits every pixel to either keep its current label or flip it to α, picking the overall flip that reduces energy the most. Graphcut minimization will repeatedly perform α-expansions for every possible label until convergence. A single α-expansion can be computed very efficiently via graph max-flow/min-cut [25], and it is from this reduction that graphcut minimization gets its name.

The optimality of the solution obtained by this algorithm depends solely on V. For simple V, it is possible to find minima that are a factor of 2 within the global minimum (i.e. a 2-approximation). If V is a metric, it is possible to find what is referred to as a "strong local minimum" [26]. As V's theoretical properties worsen, optimality guarantees get weaker, but results are still excellent in practice.

B. Constrained texture synthesis as minimization

Here we will explain how CMS can be cast as a graphcut minimization problem. We adapt the construction of Graphcut Textures, which just uses the separation cost mentioned above, and augment it with an assignment cost to capture the effect of the constraint C.

1) Defining the label space: To apply graphcut minimization, we must define synthesis in terms of a labeling L. In patch-based texture synthesis, the goal is to paste a set of candidate patches into the output, stitching them together to form seamless image. In our application, the set of candidate

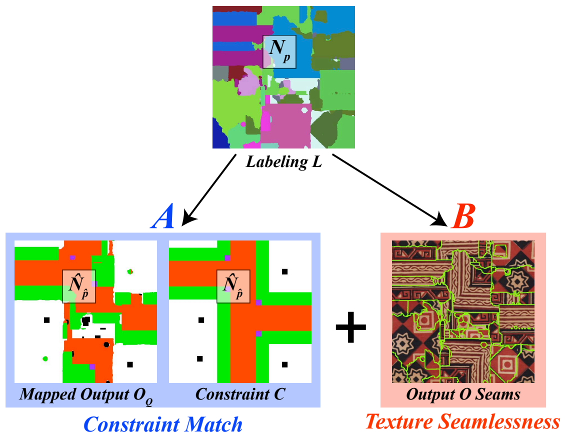


Fig. 4. The energy function $E(L)$ balances two measures of quality: constraint matching (captured by A) and texture seamlessness (captured by B). A computes the match between O_Q and C B computes the visual disparity between different labels along their boundaries.

patches is the set of translations and 90-degree rotations of S. Therefore, each label will correspond to a specific transformation of S.

For example, consider translations alone. Let $(0,0)$ be the lower left corner of O. We can define a translation label as $l = (i,j)$, where (i,j) represents the offset of (the lower left corner of) S in O prior to pasting. Figure 3A shows some examples for a simple brick texture source. The label $l = (-111,-91)$ would correspond to aligning the lower left corner of S with the lower left corner of O, moving S 111 pixels to the left and 91 pixels down, and then pasting S into O. We call this resulting image S_l; note that S_l is the same size as O. Arbitrary patch transformations are handled in an analogous manner. A sample labeling L is shown in Figure 3B. Notice how contiguous patches correspond to contiguous labels. The synthesis result L is a mapping of locations in O to labels in this space.

2) Defining the energy function: Now that we have the labeling L, we need to define the energy function E capturing synthesis quality. We will first define some terminology to express the correspondence of pixels and neighborhoods between the various images we have. Let p be a pixel location in L (or O), and N_p its neighborhood; then the corresponding location and neighborhood in C (and O_Q) are given by $\hat{p}$ and $\hat{N}_{\hat{p}}$, respectively.

The energy function E we define over L requires several derived quantities. Recall that we synthesize two images in parallel: the mapped output O_Q from Q, and the output O from S. O_Q is used to determine if we have matched C, and O is the final output we are computing. Both of these images can be computed from L as follows:

- Let g_O be the function that creates the output O from the labeling L; i.e. $g_O(L) = O$. This is computed by looking up each location's label in L and using it to find the appropriate source transformation to copy from. For example, if $L(p) = l$, then to find $O(p)$, transform S as per the label l, forming S_l, and take the pixel $S_l(p)$.
- Let g_{O_Q} be the function that creates the mapped output O_Q from L; i.e. $g_{O_Q}(L) = O_Q$. This is computed in an analogous manner; if $L(p) = l$, then we transform Q as per the label l, forming Q_l, and take the pixel $Q_l(\hat{p})$.

Given this, the energy function E characterizing the quality of the synthesis result can be defined as:

$$E(L) = \textstyle\sum_p \; A(p,L) + \sum_{(p,q)} \; B(p,q,L(p),L(q)) \tag{2}$$

where the second summation is over all pairs of neighboring pixels (p,q). Note the similarity to Equation 1.

Figure 4 gives an overview of E and its two terms. The A function, the *agreement cost*, captures how well our current synthesis matches the constraint at p. It is given by:

$$A(p,L) = KSSD_{\hat{N}_{\hat{p}}}(g_{O_Q}(L),C) = SSD_{\hat{N}_{\hat{p}}}(O_Q,C) \tag{3}$$

where K is a weighting factor, and $SSD_N(X,Y)$ is the sum of square differences between images X and Y over a neighborhood N:

$$SSD_N(X,Y) = \sum_{p\in N} |X(p) - Y(p)|^2$$

The neighborhood N_p in O matches the constraint if the difference between $\hat{N}_{\hat{p}}$ in O_Q and $\hat{N}_{\hat{p}}$ in C is small. It is possible to define A by only looking at the individual pixels p and $\hat{p}$, but this approach will not capture visual coherence and continuity; therefore, we use neighborhoods.

The B function, the *boundary* cost, captures how visible the boundaries are between different adjacent labels (and therefore different patches) in the output O. Our function is equivalent to the M function of Graphcut Textures. It is given by:

$$\begin{aligned} B(p,q,L(p),L(q)) &= ||S_{L(p)}(p) - S_{L(q)}(p)||_2 \quad (4)\\ &+ ||S_{L(p)}(q) - S_{L(q)}(q)||_2 \quad (5)\end{aligned}$$

Intuitively, this function is capturing the following: if pixels p and q are copied from different patches, then when walking from p to q, will one notice that the patch has actually changed (and vice-versa)? If two neighboring pixels are copied from the same patch, the B cost is 0 for that pair, since their labels are the same.

There are two user inputs affecting the energy function E (Equation 2). One is K, which weights the A and B costs during synthesis. The other is is N, the size of the neighborhoods CMS considers (i.e. the size of N_p and $\hat{N}_{\hat{p}}$). In general, larger N captures larger structures, and larger K enforces a closer constraint match; see Section V-H for details.

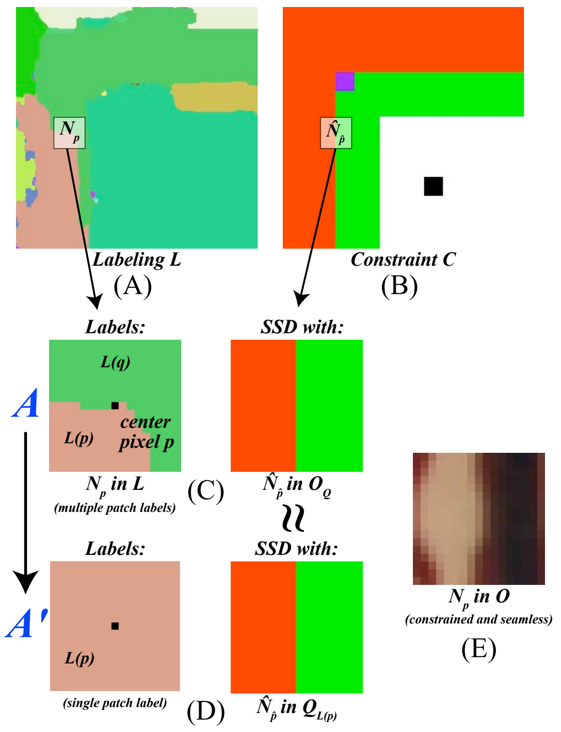


Fig. 5. Graphcut minimization cannot work with the A cost function because of multiple labels in a single neighborhood. Therefore we approximate A with A', which only looks at the center label. **(A)** Constraint neighborhood $\hat{N}_{\hat{p}}$ in C. **(B)** Corresponding neighborhood N_p in L. **(C)** The A function uses $\hat{N}_{\hat{p}}$ in O_Q, corresponding to N_p in L, which has multiple labels. **(D)** The A' function only looks at the center pixel label, so it just uses $\hat{N}_{\hat{p}}$ in $Q_{L(p)}$, which is the patch corresponding to the center label. The neighborhoods in O_Q and $Q_{L(p)}$ are similar after several patch pasting moves, showing that this is a reasonable approximation in practice. **(E)** N_p in the resulting final output O.

3) Graphcut minimization for CMS: There are two issues with applying the graphcut framework directly to CMS. Firstly, we must modify our objective function before we can use graphcut minimization. Secondly, we must introduce a heuristic to make graphcut minimization feasible for our application.

Modifying the agreement cost A**.** The assignment cost U in Equation 1 is nearly suited to represent A. However, the definition of A is not directly applicable because it is defined over a neighborhood of L, which contains multiple labels, rather than over the single label $L(p)$ (as required by graphcut minimization). Therefore, we approximate A with another function A' that does have the correct form. A' evaluates how suitable a patch label is for a location p. At $L(p)$:

$$A'(p, L(p)) = K \ SSD_{\hat{N}_{\hat{p}}}(Q_{L(p)}, C)$$

Intuitively, this treats a neighborhood of pixel locations with possibly different labels as if it were a neighborhood of pixel locations all with a single label: the label of the center pixel. See Figure 5 for an illustration. The reason A' works well is that optimizing the A' and B costs at all locations in N_p tends to force local agreement even at patch boundaries. Therefore, even if N_p contains different patch labels in L, the actual neighborhood $\hat{N}_{\hat{p}}$ in the mapped output image O_Q will eventually look like it came from a single patch.

With this change, we have a new energy function that is amenable to graphcut minimization:

$$E'(L) = \textstyle\sum_p \ A'(p, L(p)) + \sum_{(p,q)} \ B(p, q, L(p), L(q)) \qquad (6)$$

Many labels. CMS presents a challenge for graphcut minimization because the set of labels is extremely large, and the original minimization algorithm tries all labels multiple times until convergence. This is a recognized open problem in the vision community as well [21]. Translations alone require $O(m+n)$ labels, where m is the number of pixels in S and n is the number of pixels in O (the lower left corner of S can occur in every pixel of O, and then every other pixel of S can occur in the lower left corner of O). Adding basic rotations or other affine transformations expands this further. Because of the lack of constraints, Graphcut Textures does not need to paste many patches to produce good output (on the order of tens). By contrast, CMS needs to adapt to all the features in C, and thus it needs to paste many more times (100-500 in our examples). Furthermore, to get the best output, one needs to consider as much of the full label space as possible. Even for small output (e.g. $n = 128 \times 128$) the space is huge.

We apply a heuristic to get around this issue. To select a good α-expansion, CMS first ranks all output locations according to the following match error metric, which is a local version of our original texture quality energy function E:

$$MatchErr(p, L) = A(p, L) + \sum_{(p,q)\in N_p} B(p, q, L(p), L(q)) \qquad (7)$$

The idea behind $MatchErr$ is to find a neighborhood where the labeling has high energy, so that we can search for an α-expansion to improve it. The A cost is evaluated solely at p, and the B cost is evaluated for all adjacent pixel pairs $(p, q) \in N_p$. Now, for every output location p that has a large match error, we rank all candidate labels that could possibly improve the texture at p, based on the following improvement metric:

$$\begin{aligned} Imprv(l, L) &= A'(p, l) + SSD_{N_p}(S_l, g_O(L)) \\ &= KSSD_{\hat{N}_{\hat{p}}}(Q_l, C) + SSD_{N_p}(S_l, O) \qquad (8) \end{aligned}$$

The $Imprv$ metric is trying to guess which α-expansions will successfully reduce the energy at p. The first term is A', which as discussed earlier evaluates the suitability of a particular label for a particular location. The second term tries to predict the seam cost after the α-expansion by comparing S_l and O in this neighborhood.

After ranking all labels, we select the top c candidates (typically $c = 5$). Experimentally, output quality seems to be insensitive to increasing this parameter further. For performance reasons, we have found that $c = 1$ is not the best choice for our algorithm because we search for candidate patches using a *local* measure of improvement, whereas an

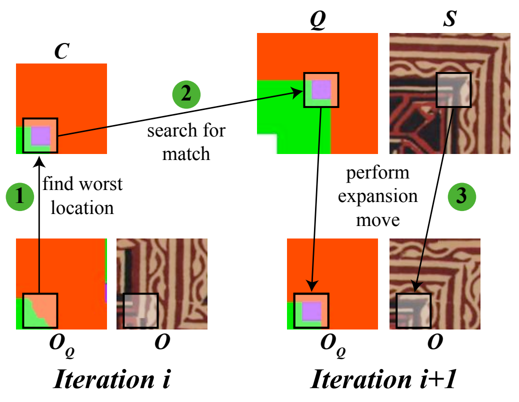


Fig. 6. Overview of a single iteration of CMS. (1) Find a location in O that needs improvement. (2) Find candidate labels that match this region by searching Q for matches with C (constraint) and S for corresponding matches with O (seamlessness, not shown). (3) Perform α-expansions integrating data from S with O. Steps (1)-(3) are repeated until the termination condition is satisfied.

α-expansion move can have a much more *global* effect that our metrics cannot take into account. It is quite possible that the best candidate based on our local measure does not actually result in the best α-expansion for our output location; therefore, it is useful to perform expansions for *all* of the top c matches. Computationally, patch search is slower than α-expansion computation, so it is to our advantage to use one patch search to come up with and try many candidate labels to speed up convergence. For example, this results in a $3\times$ speedup for the cloth synthesis example in Figure 8A.

We avoid picking tightly clustered labels corresponding to similar translations by enforcing that each label lies beyond some minimum cutoff distance d from any other candidate label (like the Poisson distribution in [27]). In our implementation we use $d = 20$.

C. Algorithm

We now describe the full algorithm (see Figure 6).

Step 0: Initialization. To start, all locations in L are assigned to a dummy label that has an artificially large cost. Thus, any label we choose will be preferred to this dummy label in the optimization process. We have found that our initialization of L does not significantly affect performance or convergence properties.

Step 1: Output refinement location. Find the *worst* location p_0 in O by evaluating $MatchErr$ (Equation 7) at each location p and storing the results in a priority queue $HistLocs$. Pick the top location.

Step 2: α-expansion search. Find the best c candidate labels to improve p_0 by evaluating $Imprv$ (Equation 8) at each label and storing the results in a priority queue $SourceLocs$, picking the top c labels.

Step 3: Performing the α-expansion. Perform an α-expansion for each of the c labels as described by [20], using $U = A'$ and $V = B$ as the cost functions.

Step 4: Termination. On each iteration, the algorithm maintains a history of h locations we have tried to improve (h = 5-10 for all our examples). If all c moves combined don't provide enough improvement for the "worst location" p_0 (as computed by Step 1), then we look in $HistLocs$ for the next "worst locations" $p_1, p_2, \ldots\ldots p_{h-1}$, repeating Steps 2 and 3 for each, until we find moves that improve the energy. If none of the h locations can be improved by more than some tolerance energy t, iterative synthesis is terminated. Because K scales the A' energy term, t should be proportional to K. We use $t = 5(K + 2)$ for all our examples; smaller values of t do not appear to improve image quality significantly.

Extensions: CMS easily extends to handle multiple sources and multiple constraints. In general, given sources $\{S_j\}$ and constraints $\{C_k\}$, CMS requires as additional input mapped sources $\{Q_{jk}\}$ to interrelate all sources and constraints. The only change required to the cost functions is that A and A' must now sum over all constraints, instead of just looking at one.

D. Detailed comparisons with other algorithms

Our work combining graphcut minimization and constrained synthesis differs from that in previous / concurrent work in the following ways:

Graphcut Textures [2]. While the mechanisms of our algorithm and Graphcut Textures are similar, the most obvious difference between the two is the domain of application. Graphcut Textures addresses the problem of unconstrained synthesis, where there is no assignment cost to guide patch placement. Therefore, it is necessary in their work to develop a variety of patch searching / pasting heuristics and graphcut extensions to somehow capture the structure of a variety of textures while still maintaining some irregularity in the output. CMS, on the other hand, is focused on constrained synthesis, captured by the addition of the assignment cost term. We use one deterministic heuristic for all results, and our implementation of the graphcut minimization technique is entirely modeled after the original construction in Boykov et. al.

Algorithmically, the presence of an assignment cost results in many more patch pasting moves than in Graphcut Textures, which is the reason we opted for an automated termination condition and c expansion moves per heuristic search, instead of one, as done in Graphcut Textures. One could imagine adding similar automated termination conditions and multiple patch pasting moves to Graphcut Textures without much difficulty, although such additions are probably not useful without an assignment cost.

Interactive Digital Photomontage [14]. Interactive Digital Photomontage uses a graphcut energy formulation containing both an assignment cost and a separation cost to capture the idea of interactive image editing for various user-specified objectives, such as color, focus, and so on. The primary difference it has with CMS is that it performs α-expansions through user guidance and interaction. A user will typically paint a 'single-image objective' capturing some desired attribute, and the interface will provide the user with a limited set of patch pastings (α-expansions) to choose from. Automatic minimization along the lines of CMS (referred to as a 'multi-image objective' in their work) is not typically encouraged because the application is interactive.

BTF surface decoration [15]. Zhou et. al. also use an assignment cost and a separation cost for the application of BTF synthesis on surfaces. Their constraints are typically binary masks, which is ideal for their application (binary masks show presence/absence of a new BTF texture patch). However, instead of α-expansions, they use square patch quilting for synthesis, and perform a graphcut computation to find optimal seams between patches. CMS, on the other hand, uses much more general image constraints, and attempts to run a full optimization in the spirit of Boykov et. al.

Image Quilting [4]. The constrained version of this algorithm is a multi-pass hierarchical technique. First, patches of a certain size in S (say 64×64) and pasted into O in raster order, using quilting. Patches are selected based on how well they match C and the existing output in O. Call the finished output O_{prev}. Then, in the next pass, the patch size is reduced, and the same synthesis algorithm is repeated, this time incorporating match cost with O_{prev} into the patch selection, and also increasing the weight of C. O_{prev} is updated, and this iterates several times. To fully match the constraint, it is important to use very small patches in the final pass.

Image Quilting only demonstrates results for texture transfer, but detail synthesis results have reasonable quality as well. However, this algorithm has two shortcomings. One is that the artifacts of the raster-scan patch quilting approach are often evident in the output, in the form of sharp vertical / horizontal seams or block-like behavior. The other is performance. For example, given a 640×500 image, and a final patch size of 7×7, one needs to perform almost 9000 patch searches just for the final pass. CMS performs around 50-100 patch searches for the full algorithm. Thus, our running time for such an example is about 4 minutes, while the Image Quilting algorithm can take upwards of 7 hours.

Schödl Ph.D. Thesis [13] This algorithm is a proposed extension to [2], where an assignment cost is added to the objective function, and full graphcut minimization is attempted with random label selection (realized as random patch placement). In this case, the only demonstrated results are for artistic filtering, and the random placement algorithm works sufficiently well for such cases. However, as shown in Figures 7A and 8A, this technique is not as effective for solving problems with highly structured constraints (see submitted images for further comparisons). As mentioned in our discussion of many labels, introducing an assignment cost significantly changes the behavior of the optimization framework; it is difficult for random placement to adapt quickly and effectively to the features of the constraint image, especially when both the source and constraint are highly structured. Thus, this approach cannot converge to good solutions as fast as CMS.

Texture Optimization [11]. Concurrent with our work, Kwatra et. al. formulate an optimization framework for unconstrained and constrained texture synthesis. They use an expectation-maximization type algorithm instead of graphcut minimization, and demonstrate impressive constrained results for textures affected by flow fields. Performance times are roughly 7-10 minutes for 256×256 output textures on a machine similar to ours. By comparison, CMS focuses on detail synthesis and image analogies applications, and can synthesize textures as large as 470×1265 in roughly 5 minutes. Kwatra et. al. acknowledge that their new optimization algorithm is more susceptible to getting stuck in local minima than graphcut minimization.

V. Results

Example	Output size	CMS	IA	IQ
Carpet (Fig 7A)	350×433	79	1119	7413
Brick 5× (not shown)	609×609	146	9340	11091
Brick 10× (Fig 7B)	609×609	402	*	*
Cloth (Fig 7C)	470×1265	312	12775	13714
Freud (Fig 8B)	640×500	217	1035	25016
Weave (Fig 8C)	366×554	44	368	1090
Rice (Fig 8D)	231×281	9	103	516
Cloth TBN (Fig 8A)	512×512	297	656	6599
Arch TBN (Fig 8E)	320×462	131	406	5163
Melody TBN (Fig 8F)	640×666	239	**	3689

TABLE I
PERFORMANCE RESULTS OF THE CMS, IA, IQ ALGORITHMS. TIMES ARE GIVEN IN SECONDS. *ONLY CMS CAN RUN THE BRICK 10× EXAMPLE BECAUSE IT SUPPORTS MULTIPLE CONSTRAINTS. **THIS EXAMPLE DID NOT RUN ON THE PUBLICLY AVAILABLE IA CODE.

We show results of our CMS algorithm and compare with Image Analogies [3] (IA), Image Quilting [4] (IQ), and Schödl [13] in Figures 7 and 8. Figure 9 shows additional results. Parameters of our algorithm are given in the figure captions.

A. Performance

We used the publicly available version of IA from NYU's Media Research Lab, and our own implementation of IQ. CMS graphcut minimization was implemented using code from the authors of [26]. All patch searching operations in IQ and CMS used the convolution and summed area table optimizations of [2]. Results were obtained on a Pentium Xeon 3.6 GHz machine with 2 GB RAM.

Table I shows detailed timing results. In general, CMS is able to quickly generate high quality output which other algorithms require a far longer running time to create (Brick 5× detail is an extreme example). For the majority of the examples, CMS is 5-45× faster than IA, and 14-115× faster than IQ. IA's running time tends to be much greater in detail synthesis problems, and in cases where the running time of IA is the same as CMS, such as in Figure 8A, the CMS output is significantly better. IQ's running time is much greater when very small patches are required to closely match the constraint (see Section IV-D), and even with this greater running time it is not always able to match the quality of CMS, such as in Figure 7A. The running time of Schödl's algorithm is dependent on the number of iterations chosen by the user, so for fair comparisons to CMS we ran Schödl for a similar amount of time as CMS. Schödl's results have considerable artifacts in examples with highly structured constraints, such as Figure 7A and Figure 8A. On other examples, it is more comparable to CMS (Figure 8B, submitted images).

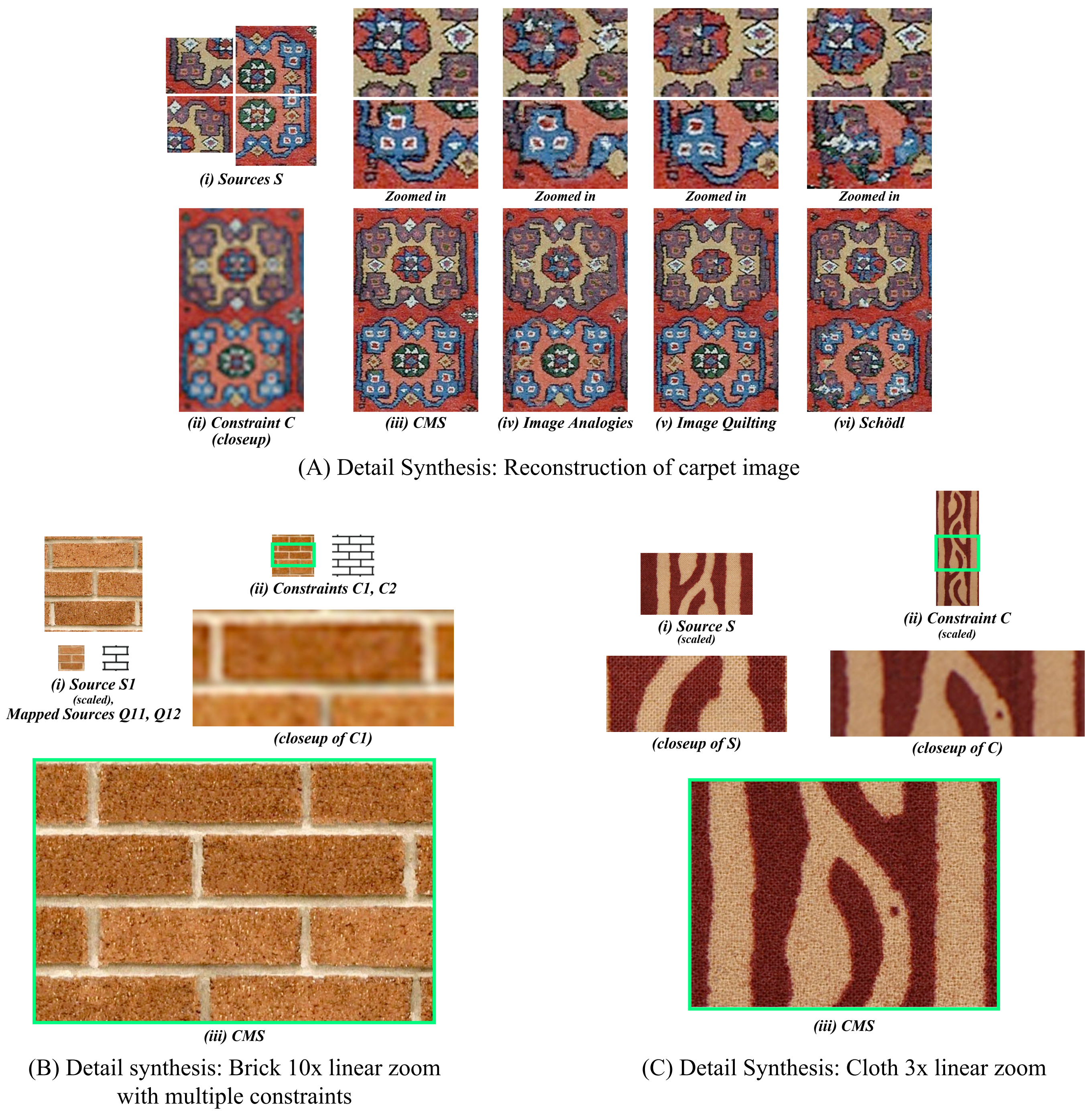

(i) Sources S
Zoomed in
Zoomed in
Zoomed in
Zoomed in
(ii) Constraint C (closeup)
(iii) CMS
(iv) Image Analogies
(v) Image Quilting
(vi) Schödl
(A) Detail Synthesis: Reconstruction of carpet image
(ii) Constraints C1, C2
(i) Source S1 (scaled), Mapped Sources Q11, Q12
(closeup of C1)
(iii) CMS
(B) Detail synthesis: Brick 10x linear zoom with multiple constraints
(i) Source S (scaled)
(ii) Constraint C (scaled)
(closeup of S)
(closeup of C)
(iii) CMS
(C) Detail Synthesis: Cloth 3x linear zoom

Fig. 7. CMS for detail synthesis, with parameters $K = 7$, $N = 7$ (for (C), $N = 3$ due to curves). In all figures, (i) source S, (ii) constraint C, (iii) CMS output, and, if present, (iv) image analogies output, (v) image quilting output, (vi) Schödl output. **(A)** Carpet image is restored from a blurry constraint. **(B)** 10× linear magnification of brick with multiple constraints. Close examination shows how well the output matches subtle color and shadow changes in the low resolution image. **(C)** 3× linear magnification of cloth weave pattern (part of the output shown). CMS is able to adapt to variability in the leaf shapes.

B. Detail synthesis

We demonstrate the ability of CMS to synthesize detail in Figure 7A, where we enhance a blurred carpet (7A-(iii)). For comparison, we used the IA, IQ, and Schödl algorithms to generate similar results (Figure 7A-(iv-vi)). The original constraint image is much larger; only 2 medallions are shown (see submitted images). The IA output is much better than that published in their original paper, because we provided better example data and tweaked some parameters. Even so, CMS performs better than IA and the other techniques as well. Schödl's algorithm performs particularly poorly on the bottom carpet medallion, and IQ has difficulty matching a medallion that is not exactly the same shape as the medallions in the source (see zoom-in). Figure 7C shows 3× linear zoom of

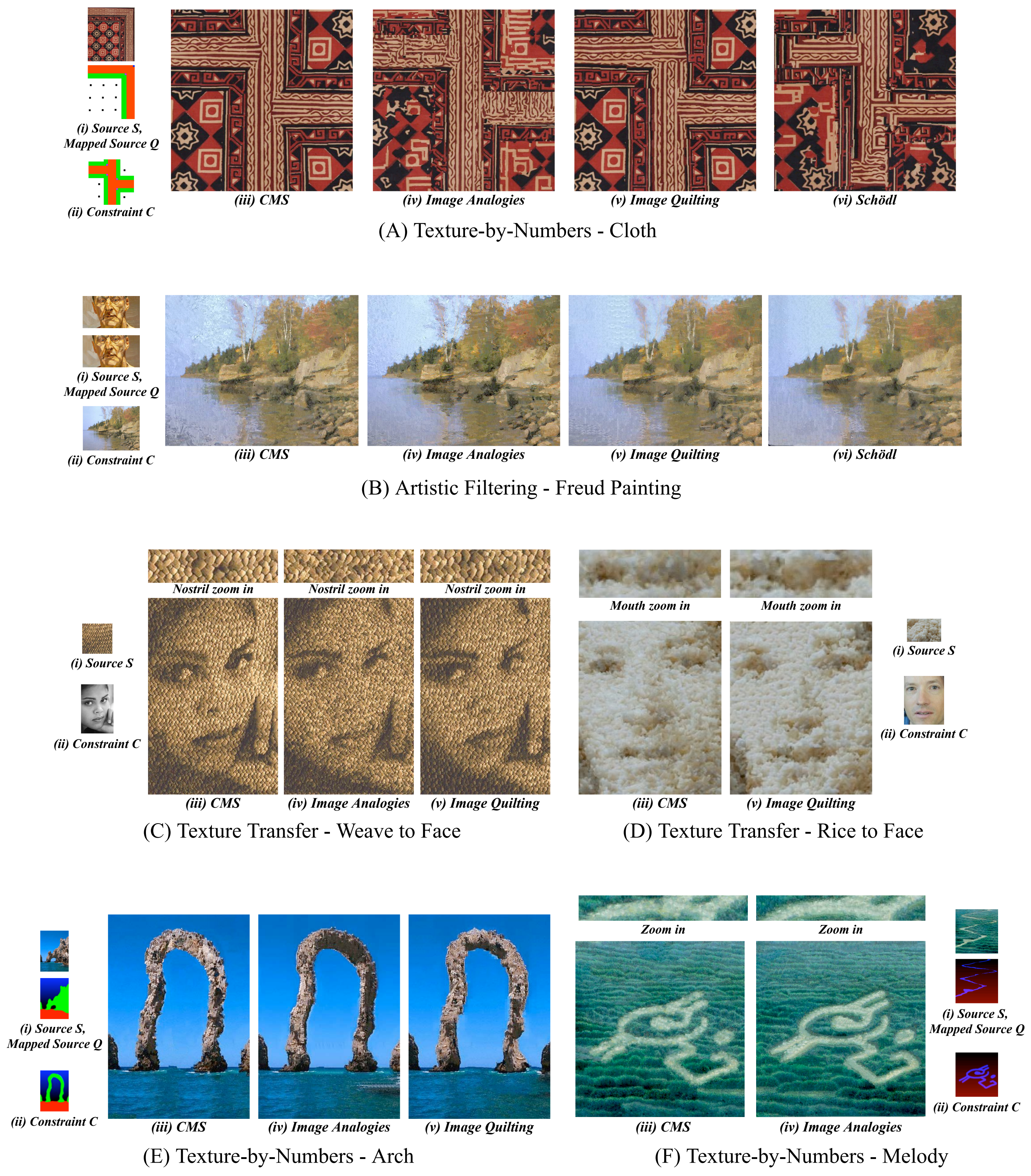
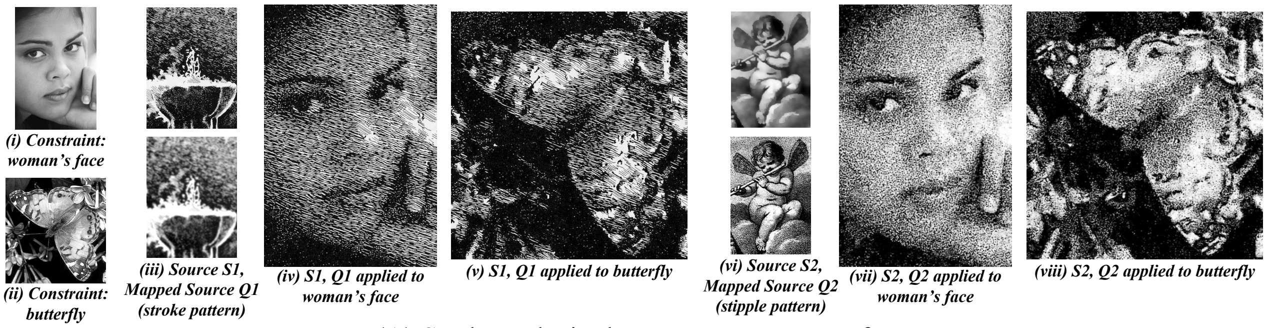
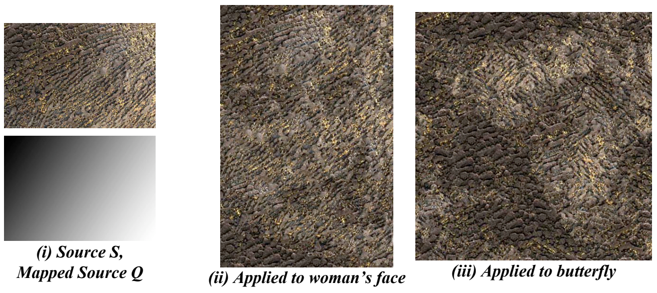


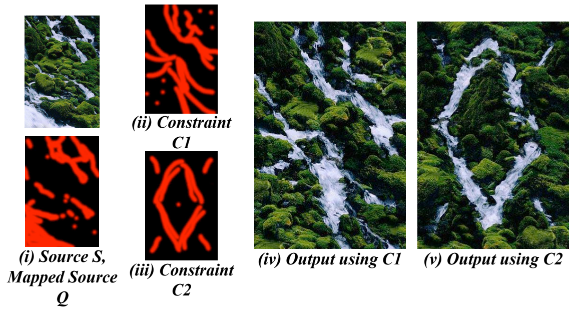
Fig. 8. Analogy examples (parameters in parenthesis). **(A)** Cloth texture-by-numbers. ($K = 0.5$, $N = 9$) CMS produces good output, while IA is under-constrained and synthesizes incorrectly. IQ's fixed patch size makes it hard to capture large and small structures in a seamless texture. **(B)** Artistic filter. ($K = 20.0$, $N = 5$) Results are comparable, but CMS matches fine details like the thin trees better. **(C)** Weave texture transfer. ($K = 0.4$, $N = 4$) CMS is able to both match the constraint and synthesize a coherent weave texture. Note the comparisons of the nostril. **(D)** Rice texture transfer ($K = 0.2$, $N = 5$). Again, CMS matches the constraint closely while finding better patch seams. **(E)** Arch texture-by-numbers. ($K = 10.0$, $N = 2$) CMS avoids synthesizing streaks in the water (improvement over IA) and adapts to the smooth inner contours of the arch without blocky artifacts (improvement over IQ) **(F)** Melody texture-by-numbers. ($K = 0.6$, $N = 1$) Challenging case. CMS is not as flexible at matching the thin strokes in the constraint.


(i) Constraint: woman's face
(ii) Constraint: butterfly
(iii) Source S1, Mapped Source Q1 (stroke pattern)
(iv) S1, Q1 applied to woman's face
(v) S1, Q1 applied to butterfly
(vi) Source S2, Mapped Source Q2 (stipple pattern)
(vii) S2, Q2 applied to woman's face
(viii) S2, Q2 applied to butterfly

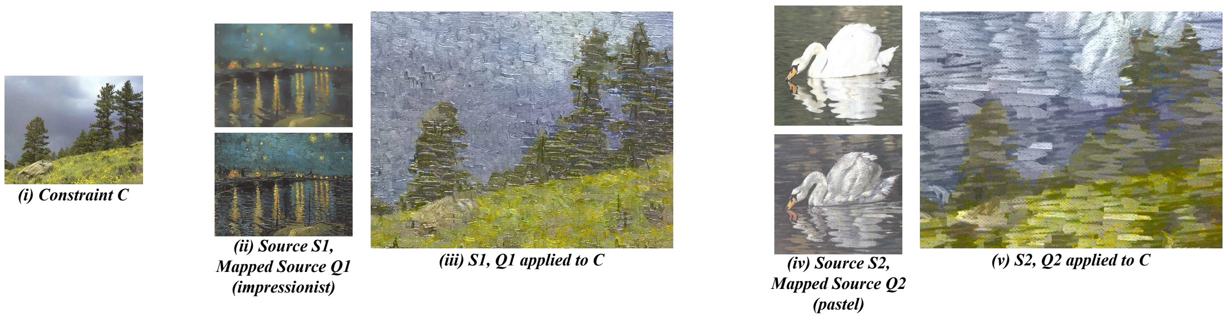
(A) Stroke and stipple pattern texture transfer


(i) Source S, Mapped Source Q
(ii) Applied to woman's face
(iii) Applied to butterfly

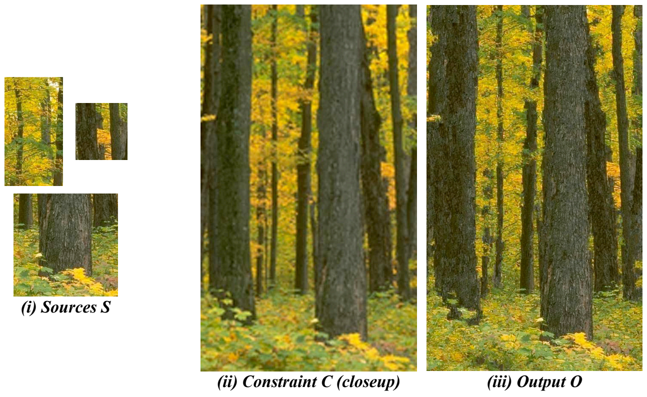
(B) Pottery texture transfer



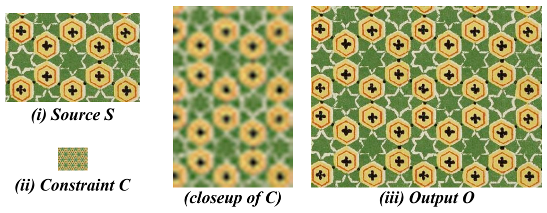
(C) Waterfall texture-by-numbers


(i) Constraint C
(ii) Source S1, Mapped Source Q1 (impressionist)
(iii) S1, Q1 applied to C
(iv) Source S2, Mapped Source Q2 (pastel)
(v) S2, Q2 applied to C

(D) Impressionist and pastel artistic filters



(E) Detail Synthesis: Reconstruction of forest



(F) Detail Synthesis: Rug 8x linear zoom

Fig. 9. Additional results showing the scope of our technique (parameters and running times given for each). **(A)** (iii) Run with $K = 2.0$, $N = 5$. (iv) 159 seconds; (v) 401 seconds. (vi) Run with $K = 4.0$, $N = 5$. (vii) 93 seconds; (viii) 271 seconds. **(B)** (i) Run with $K = 0.5$, $N = 5$. (ii) 36 seconds; (iii) 80 seconds. **(C)** (iv) $K = 0.5$, $N = 5$, 44 seconds. (v) $K = 1.0$, $N = 5$, 64 seconds. **(D)** (iii) $K = 1$, $N = 5$, 59 seconds. (v) $K = 0.5, N = 1.0$, 112 seconds. **(E)** $K = 7$, $N = 7$, 154 seconds. **(F)** $K = 7$, $N = 7$, 26 seconds.

woven cloth. It adds weave detail to the high resolution output while matching the constraint, including the small red spot in the leaf/stalk pattern and various irregularities in the leaf contours (submitted images).

C. Multiple constraints

In Figure 7B we demonstrate the use of multiple constraints for dramatic $10\times$ linear magnification of brick (i.e., $|O| = 100|C|$). The low-resolution brick wall provides one constraint, and the black lines provide a second constraint guiding where the grout should be. CMS maintains the brick color variations and shadows cast by bricks on grout, while also keeping the brick/grout edges reasonably straight, as dictated by the second constraint. This mechanism is different from that of Liu et. al. [28] because CMS has no implicit knowledge of grid structure apart from what is explicitly specified in the constraint.

D. Artistic filters

For the Freud painting example we first use the simple luminance matching technique of Image Analogies to bring the S and Q color spaces into correspondence with C (Figure 8B). We show outputs of the IA, IQ, and Schödl algorithms. The results are comparable, but CMS better preserves details like the thin trees in the center and right, and IQ suffers from some repetitive block copying.

E. Texture transfer

Texture transfer involves a tension between matching the constraint and generating convincing texture. Figure 8C shows that CMS can match the constraint well and generate a nicer weave texture than IA and IQ. IA's output is less coherent than the original weave texture, and IQ's output has visible horizontal / vertical boundaries (see closeup). For the rice texture in Figure 8D, CMS does less repetitive copying and generates better seams than IQ in the mouth region. Note that in these examples, both S and Q are the same image, so only S is shown in the figures.

F. Texture-by-numbers

Some texture-by-numbers examples are shown in Figures 8A, 8E, and 8F. The cloth example (8A) benefits greatly from patch-based techniques. CMS handles the distortion of the cloth border by synthesizing the confluence of the line patterns; IA produces a much less coherent texture. IQ performs better than IA, but due to the sensitivity of its patch size parameter, it is unable to capture all structures in the output seamlessly (the cloth border in particular). In Figure 8E, CMS avoids synthesizing unnatural streaks in the water that arise in the IA output from "garbage" growing [29], a phenomenon where the synthesis algorithm gets stuck in one part of the search space. It also captures the inner contours of the arch better than IQ, which demonstrates noticeable block artifacts from raster-scan square patch refinement.

Figure 8F shows a challenging case for CMS. For this example, the mapped source image Q does not have many regions matching the exact curves of C, so while CMS generates reasonable output, there is still some irregularity in the edges of the symbol in the field.

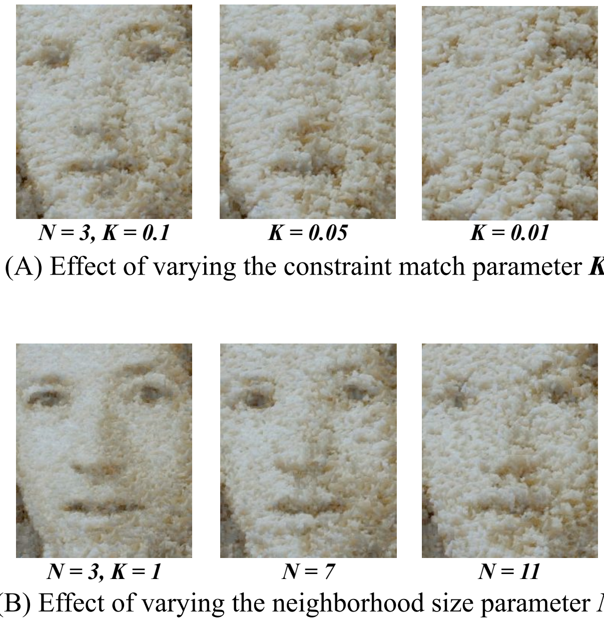

N = 3, K = 0.1
K = 0.05
K = 0.01
(A) Effect of varying the constraint match parameter K
N = 3, K = 1
N = 7
N = 11
(B) Effect of varying the neighborhood size parameter N

Fig. 10. Effect of varying synthesis parameters, demonstrated on rice texture transfer example. **(A)** As the constraint match parameter K tends to zero, CMS gives less and less weight to the structure of the face, degenerating into normal texture synthesis. **(B)** As N is increased, CMS starts favoring large structures over small features. The face is preserved, but the details are lost.

G. Additional results

Figure 9 shows additional results demonstrating the versatility of our technique. Due to space constraints, we could not include comparisons to all of the other competing algorithms; however, we feel our comparisons in Figure 7 and Figure 8 are representative.

H. Discussion

We now discuss strengths and weaknesses of CMS and compare patch-based and pixel-based synthesis with constraints.

Correspondence between Q and C: For CMS, IA, and IQ to make sense as actual analogy computations, there must be some correspondence between Q and C; otherwise, the comparisons between these two images become counterintuitive. Therefore, for applications such as artistic filtering, it is hard to define success. Although CMS supports these applications, the algorithm is intended for use in situations where Q and C actually represent the same type of data, such as in detail synthesis or texture-by-numbers.

Ensuring B has the right properties: The B function (Equation 5) is not a metric, and thus it often violates a condition known as *regularity* [26]. If B is not regular, then the max-flow/min-cut operation in a single minimization step can actually cause the overall energy to go up, because certain flow graph edges will have negative weights. To prevent this from happening, we delete these negative edges from the graph prior to each α-expansion move computation. It can be shown that this modified graph will never result in an expansion move that increases the total energy; for details, refer to [30].

Synthesis parameters: CMS synthesis is controlled primarily by N, the neighborhood size, and K, the constraint match parameter; the settings for our outputs are given in Figures 7, 8, and 9. For a user who has experience with CMS, it usually takes 2-3 tries to find a good set of parameters for a given example. Figure 10 shows the effects of parameter changes on the rice texture transfer example (Figure 8D). In general:

- *Large* N captures larger coherent structures. *Small* N adapts better to curves and other smaller features, especially when the source data does not have exact matches for them.
- *Large* K forces a closer constraint match, which is important for detail synthesis and some filters. *Small* K enforces texture seamlessness, which is important in more loosely constrained applications like texture transfer, at the cost of capturing the structure of the constraint.

Advantage of pixel-based methods: Patches do a very good job of preserving local coherence, but pixels are much better at adapting to radical changes in structure (Figure 8F), which typically cannot be seamlessly composed of multiple patches, even if color variations across the structure are only subtle. Wu et. al. [31] and Liu et. al. [28] introduce warping, which works quite well in small amounts, but heavy deformation causes unwanted blurring. This is an area for future work.

Good source data is needed: If the sources S are chosen such that that Q matches poorly with C, then the resulting output will suffer. For best results, analogy-based synthesis algorithms require enough example data to account for all structures present in C.

VI. CONCLUSIONS

Constrained texture synthesis is a powerful capability with many applications. This work shows how to cast constrained synthesis as energy minimization and optimize it using graphcut minimization. The resulting synthesis algorithm respects constraints and yields high-quality results much faster than existing algorithms. Therefore it should have significant impact on the use of image analogy techniques in practice.

We make the following contributions: we formulate constrained texture synthesis in a principled way as a minimization problem that requires simultaneous optimization of both the agreement between the output and constraint image, and the seamlessness of the patches making up the output. We show how to approximate this energy minimization so that a graphcut minimization algorithm can be used to efficiently find solutions. Finally, we show the particular utility of this algorithm in solving the problem of detail synthesis.

We have demonstrated CMS on a range of applications: detail synthesis, texture-by-numbers, artistic filters, and other image analogies. We have also demonstrated a novel use of multiple constraints for dramatic detail synthesis (10× zoom).

In future work, more general patch and color transformations would extend the power of this technique. Another interesting avenue is using the energy minimization framework to implement perceptual measures of texture quality.

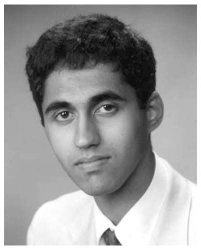
ACKNOWLEDGMENTS

We thank Ramin Zabih and Vladimir Kolmogorov for useful discussions on graphcuts and for providing graphcut code. We also thank Baining Guo, Kun Zhou, the members of the Cornell Program of Computer Graphics, and the reviewers for their helpful comments. Finally, we thank Gus Lott for the cloth texture photograph and the authors of Image Analogies / Image Quilting for making many other examples available on their webpages. This work was supported by NSF grant CCF-0539996 and an Affinito-Stewart Grant.

REFERENCES

[1] G. Ramanarayanan and K. Bala, "Constrained graphcut texture synthesis," Department of Computer Science, Cornell University, Tech. Rep. CUCS-TR2005-1995, April 2005.
[2] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: Image and video synthesis using graph cuts," in *SIGGRAPH*, 2003, pp. 277–286.
[3] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *SIGGRAPH*, 2001, pp. 327–340.
[4] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *SIGGRAPH*, 2001, pp. 341–346.
[5] R. Ismert, K. Bala, and D. Greenberg, "Detail synthesis for image-based texturing," in *Symposium on Interactive 3D Graphics*, April 2003, pp. 171–176.
[6] D. J. Heeger and J. R. Bergen, "Pyramid-based texture analysis/synthesis," in *SIGGRAPH*, 1995, pp. 229–238.
[7] J. S. D. Bonet, "Multiresolution sampling procedure for analysis and synthesis of texture images," in *SIGGRAPH*, 1997, pp. 361–368.
[8] L.-Y. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization," in *SIGGRAPH*, 2000, pp. 479–488.
[9] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum, "Real-time texture synthesis by patch-based sampling," *ACM Transactions on Graphics*, vol. 20, no. 3, pp. 127–150, 2001.
[10] M. Ashikhmin, "Synthesizing natural textures," in *Symposium on Interactive 3D Graphics*, 2001, pp. 217–226.
[11] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra, "Texture optimization for example-based synthesis," in *SIGGRAPH*, 2005, pp. 795–802.
[12] J. Zhang, K. Zhou, L. Velho, B. Guo, and H.-Y. Shum, "Synthesis of progressively-variant textures on arbitrary surfaces," in *SIGGRAPH*, 2003, pp. 295–302.
[13] A. Schödl, "Multi-dimensional exemplar-based texture synthesis," Ph.D. dissertation, Georgia Institute of Technology, 2002.
[14] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen, "Interactive digital photomontage," in *SIGGRAPH*, 2004, pp. 294–302.
[15] K. Zhou, P. Du, L. Wang, Y. Matsushita, J. Shi, B. Guo, and H.-Y. Shum, "Decorating surfaces with bidirectional texture functions," *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 11, no. 5, pp. 519–528, 2005.
[16] S. Borman and R. L. Stevenson, "Super-resolution from image sequences - A review," in *Proceedings of the Midwest Symposium on Circuits and Systems*, Notre Dame, IN, 1998.
[17] W. T. Freeman, T. R. Jones, and E. C. Pasztor, "Example-based super-resolution," *IEEE Computer Graphics and Applications*, vol. 22, no. 2, pp. 56–65, 2002.
[18] C. Liu, H.-Y. Shum, and C.-S. Zhang, "A two-step approach to hallucinating faces: global parameteric model and local nonparametric model," in *Computer Vision and Pattern Recognition (CVPR)*, 2001, pp. 192–198.
[19] L. Wang and K. Mueller, "Generating sub-resolution detail in images and volumes using constrained texture synthesis," in *Proceedings of IEEE Visualization*, 2004, pp. 75–82.
[20] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI)*, vol. 23, no. 11, pp. 1222–1239, 2001.
[21] S. Birchfield and C. Tomasi, "Multiway cut for stereo and motion with slanted surfaces," in *IEEE International Conference on Computer Vision (ICCV)*, September 1999, pp. 489–495.
[22] V. Kolmogorov and R. Zabih, "Computing visual correspondence with occlusions using graph cuts," in *IEEE International Conference on Computer Vision (ICCV)*, 2001, pp. 508–515.

[23] ——, “Multi-camera scene reconstruction via graph cuts,” in *European Conference on Computer Vision (ECCV)*, 2002, pp. 82–96.
[24] C. Rother, V. Kolmogorov, and A. Blake, “Grabcut: interactive foreground extraction using iterated graph cuts,” in *SIGGRAPH*, 2004, pp. 309–314.
[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA: MIT Press and McGraw-Hill, 2001.
[26] V. Kolmogorov and R. Zabih, “What energy functions can be minimized via graph cuts?” *IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI)*, vol. 26, no. 2, pp. 147–159, 2004.
[27] S. Zelinka and M. Garland, “Jump map-based interactive texture synthesis,” *ACM Transactions on Graphics*, vol. 23, no. 4, pp. 930–962, 2004.
[28] Y. Liu, W.-C. Lin, and J. Hays, “Near-regular texture analysis and manipulation,” *SIGGRAPH*, pp. 368–376, 2004.
[29] A. A. Efros and T. K. Leung, “Texture synthesis by non-parametric sampling,” in *IEEE International Conference on Computer Vision (ICCV)*, Corfu, Greece, September 1999, pp. 1033–1038.
[30] C. Rother, S. Kumar, V. Kolmogorov, and A. Blake, “Digital tapestry,” in *Computer Vision and Pattern Recognition (CVPR)*, June 2005, pp. 589–596.
[31] Q. Wu and Y. Yu, “Feature matching and deformation for texture synthesis,” in *SIGGRAPH*, 2004, pp. 364–367.



Ganesh Ramanarayanan is currently a Ph.D. student in the Computer Science Department at Cornell University. His research interests are in texture synthesis and feature-based image enhancement. He received his B.S.E. in electrical engineering from Princeton University. Outside of computer graphics, he aspires to become a performing Carnatic musician.

Kavita Bala is an Assistant Professor in the Computer Science Department and Program of Computer Graphics at Cornell University. She specializes in interactive computer graphics, leading several research projects in interactive rendering, scalable illumination, global illumination, and image-based modeling and texturing. In 2005, Bala co-chaired the Eurographics Symposium on Rendering (EGSR); she has also served on numerous program committees including SIGGRAPH, the Eurographics Symposium on Rendering, the Point-based symposium, and Graphics Interface. She is a co-author of the graduate textbook ”Advanced Global Illumination”.

Bala received her B.Tech. from the Indian Institute of Technology (IIT, Bombay), and her S.M. and Ph.D. from the Massachusetts Institute of Technology (MIT).