Image-Based Stained Glass

Stephen Brooks

Abstract— We present a method of re-styling an image so that it approximates the visual appearance of a work of stained glass. To this end, we develop a novel approach which involves image warping, segmentation, querying, and colorization along with texture synthesis. In our method, a given input image is first segmented. Each segment is subsequently transformed to match real segments of stained glass queried from a database of image exemplars. By using real sources of stained glass, our method produces high quality results in this nascent area of non-photorealistic rendering. The generation of the stained glass requires only modest amounts of user interaction. This interaction is facilitated with a unique region-merging tool.

Index Terms— Non-photorealistic rendering, stained glass, texture synthesis, image colorization, image querying.

1 Introduction

Like all major art forms, stained glass offers insights into the aesthetic sensibilities of cultures over time. But unlike some forms, the basic methods of constructing stained glass have scarcely changed over the centuries.

Then, as now, the full outline of the design, or cartoon, for a stained glass window is drawn prior to construction [1]. In the past these cartoons were sketched directly onto tabletops and are now drawn on paper. Within the cartoon the designer is able to indicate, not just the principal outlines of the work, but also the shape and color of the individual pieces of glass to be used, along with the position of the lead strips (calmes) that hold the work together. In time, a unique range of colors could be achieved using enamels of differing pigments, which allow rich details to be painted using clear glass as a canvas. But even when painted, a stained glass window possesses a distinctive style partly due to the unique color ranges produced through the interaction of color enamels, glass and light (see figure 2). The imposition of calmes further separates the appearance of stained glass from other mediums.

In our paper we utilize centuries of stained glass artwork by basing our results on a database of real stained glass images, while the input image acts as the stained glass window's cartoon. This is most easily demonstrated if we refer to the figure 1, wherein an image of two parrots (left) is re-rendered (right) to match a given stained glass style (center).

The primary contribution of this paper is a coherent technique that transforms a given input image into a plausible stained glass rendition. In addition, we have developed an efficient glass filter and a novel multi-scale interface for image segmentation. We also introduce new concepts such as region-by-region colorization and regional querying within a single image.

• *S. Brooks is with the Faculty of Computer Science, Dalhousie University, Halifax, Canada, B3H 2Y5. E-mail: sbrooks@cs.dal.ca*

Manuscript received 20 July 2005, revised 27 Jan. 2006; accepted 6 Feb. 2006.

2 Previous Work

Our stained glass synthesis method requires image querying, warping, colorization and segmentation, as well as texture synthesis. Many of these areas have enjoyed extensive activity recently, and so, we now review only a few cases with particular relevance to the present work.

Texture synthesis is the process of generating a new instance of a texture such that it appears to come from the same source as a given input image or set of images [2], [3], [4], [5], [6], [7]. Texture synthesis has a variety of applications including hole-filling, texture transfer, and texture-by-numbers. Recently texture synthesis has also been applied to image colorization wherein one image's color character is transferred to another [8]. Alternative approaches by Reinhard *et al.* [9] and Levin *et al.* [10] perform colorization through statistical transfer and optimization.

Also related to the present work, in Carson *et al.*'s Blobworld system [11] image segmentation and image querying are brought together for a common purpose. In their work, the system segments an image into regions, called blobs, according to user-defined parameters. The user selects one region of interest, which is then used to search for similar images in a structured database. Another method of image segmentation that we will return to at a later stage is the work of Deng and Manjunath [12] that is based on color quantization and spatial segmentation.

But beyond algorithmic ties to prior work, there are also other systems that are relevant with respect to intent. Commercial software such as Adobe Photoshop® [13] offer Voronoi-based stained glass filters. However, these methods are simplistic, and do not adapt to the underlying content. Recent pioneering work on stained glass by Mould [14] improved upon this by considering image content, though one might argue that the results produced remain some distance from the appearance of real stained glass. We believe that our results exhibit a higher degree of realism in terms of color, shape and textural qualities. The greater realism stems from our use of real stained glass exemplars. This allows us to borrow the color statistics of real glass via regional querying and the calmes appearance through directed texture synthesis. This differs from the

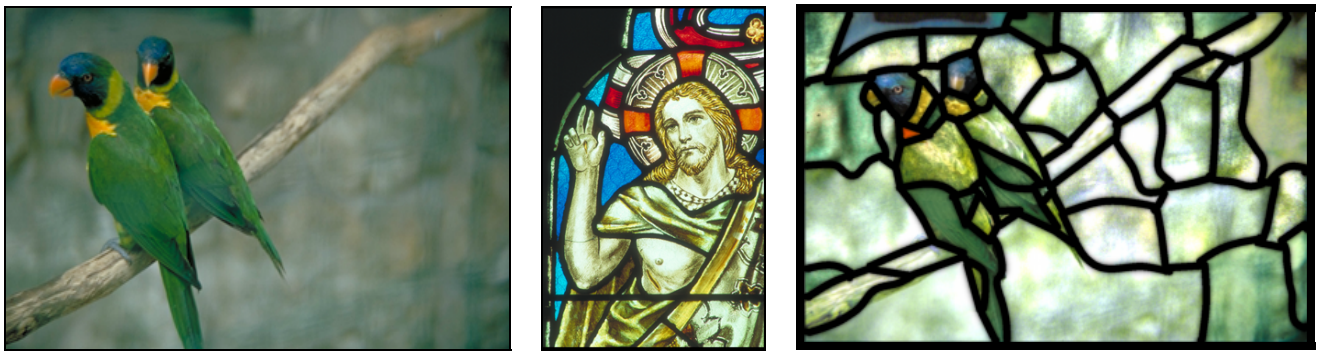

Fig. 1. A real image is converted into a stained glass style. The image of two parrots (left) is re-rendered (right) to match a given stained glass style (center).

prior work of Mould [14] which simply replaces regions with uniform preset colors and generates unrefined calme edges as a linear gradient. Our semi-automatic interface for image segmentation also offers a more robust solution to the generation of stained glass *cartoons*. The fully automatic approach of Mould [14] often segments images in ways that distort the original shapes in the image. Moreover, our glass-like filtering adds further quality to the results.

More generally, there is an array of NPR systems that attempt to mimic a variety of artistic styles [15]. Of these, prior work on mosaics [16], [17], [18] relate to the present work most closely. However, with these methods the rigidity of the pre-defined tiles imposes quite different constraints on the rendering process and produces markedly dissimilar results. The work of DeCarlo and Santella [19] is also relevant as they employ image segmentation to produce a loose, artistic style of image rendering. Other work by Bangham *et al.* [20] utilizes scale-space filtering for the selective removal of details for generating painterly images.

The present work is also related to other semi-automatic systems such as the simple interface for image segmentation introduced by Barrett and Cheney [21]. Other work in object selection includes active contour models [22], intelligent scissors [23], GrabCut [24], and photomontage [25].

3 Motivation

Considering this wealth of prior work, one might ask if, for example, texture transfer alone could be used to generate stained glass results directly. If we consider the left image of figure 3 where texture transfer has been applied to the problem, we can see that this is not the case. The image shows the unconvincing results of texture transferred from the religious stained glass image to the parrot input image. We believe this is because current methods of texture transfer cannot generate the high-level structure of stained glass.

Likewise, if we consider the application of image colorization alone, the results produced are also inadequate. We see in the right image of figure 3 that simply adjusting the color statistics of the parrot image to match the stained glass image does not capture those aspects of stained glass that marks its unique character, namely, distinct regions separated by calmes. Therefore, given the state of the art, there remains considerable scope for new techniques that can mimic the historic art form of stained glass.

4 Method Overview

We present a method which transforms a given input image, I, into a stained glass image using a real stained glass image, T, as a target. With this in mind, we now present an overview of our techniques.

It is likely that any system that proposes to generate stained glass imagery would commence with a segmentation of the input image. Our approach begins by generating multiple segmentations of the input image, I, at a variety of segmentation scales from coarse to fine. These intermediary segmentation images are then used as a basis for a final user-guided segmentation of the input image into regions, I_r. The target stained glass image, T, is also segmented into regions, T_q, in a similar fashion.

Once the input and target images are segmented, each region, I_r, from the input image is coerced to adopt the color statistics of an individual region, T_q, from the target image, on a segment-by-segment basis. The choice of which target region, T_q, is used for colorizing a given input region, I_r, is determined by comparing the color and texture statistics of the given I_r with all available T_q. The T_q that most closely matches I_r is chosen as the target for the colorization of I_r.

Fig. 2. A selection of real stained glass artwork.

Fig. 3. Attempted texture transfer (left) and color transfer (right).

The resulting colorized regions we label J_r.

In addition, we can optionally replace individual regions, I_r, from the input image with images of real stained glass, S_i, rather than produce the colorized regions, J_r (see figure 17 for example solid pieces). As will be described, our method is also able to replace all regions, I_r, that have low levels of texture content with matching solid glass pieces, S_i, from a database of solid glass pieces. This means that uniform areas, I_r, which exhibit minimal texture properties, are optionally replaced with regions of real stained glass. The texture-threshold for replacing a given, I_r, is determined by the user with simple controls.

The next stage involves the synthesis of calmes – the strips between the glass pieces. This is itself a multi-stage process. The first stage smoothes the region edges, eliminating the high edge frequencies produced at the segmentation phase. This produces a smoothed *cartoon* of the calmes that is used to construct an Image Analogies mask for the final synthesis of the lead stripping. Lastly, we apply a warping function on the image to better simulate a glass-like appearance.

As our method of stained glass synthesis is comprised of many stages, for clarity we summarize the essential steps as follows:

1. Segment the input image, I, at multiple scales
2. User-assisted merging of segmented regions, forming final segments, I_r
3. Analyze each region, I_r, in the input image, I
4. Segment target stained glass image, T, forming regions T_q
5. Analyze each region, T_q, in the target image
6. Analyze the database of solid glass pieces
7. For each region, I_r, in the input image:
   a. Find best matching region, T_q, in the target stained glass image
   b. Colorize the input region, I_r, to match target region, T_q, producing region, J_r
   c. Find best matching solid glass piece, S_i, to the colorized region, J_r, in solid glass database, S
8. Replace a (possibly null) subset of the colorized regions, J_r, segments with solid glass, S_i, according to user preference
9. Smooth the region edges
10. Synthesize the calmes
11. Application of the glass simulation function

Each of these stages will be explained in detail in the sections that follow.

5 Image Segmentation

The initial stage for transforming the input image into a stained glass style involves segmenting the input image, I. Our method for image segmentation is semi-automatic. We have taken this approach because we remain unconvinced that current fully automatic segmentation performs correctly in all cases. Others have expressed much stronger views on the matter, with Carson *et al.* stating, "it is a common belief in the computer vision community that general purpose image segmentation is a hopeless goal" [11]. While we do not speculate on the future of image segmentation, this sentiment does suggest that the current state of general-purpose segmentation can rarely be used without significant user guidance.

Often, for an arbitrary image, an automatic segmentation will be 'mostly correct' but will be unsatisfactory in at least a subset of the image. We argue that by relying on the user's innate sense of object composition, we avoid the problems of under-segmentation, over-segmentation and non-intuitive parameter tweaking that limit the applicability of many fully automated methods.

Our approach is related to the simple interface for fragment collection introduced by Barrett and Cheney [21]. In their system, the image is massively over-segmented into fragments called TRAPs. The user then manually collects the sub-object fragments, producing the final segmentation. By allowing the user to determine the segmentation of objects, their system ensures that the final region boundaries are correct for the intended purpose. However, forcing the user to collecting many tiny fragments could become tedious for non-trivial images.

In our system we aim to retain the robust segmentation that is the inevitable product of manual fragment collection, while at the same time assisting the user in the fragment collection process to a much greater extent. We also wish to retain the simplicity of the interface, and not introduce multiple tools for segmentation. Indeed, one could argue that an existing image editing system such as Adobe Photoshop® [13] could be used to segment an image. However, this would force the user to begin the segmentation process without guidance and would likely require the application of multiple types of editing tools.

Since an under-segmentation has excessive fragment collection and an over-segmentation is a chore to collect, we generate a multi-level segmentation in a similar fashion to DeCarlo and Santella [19]. The top row of figure 4 shows a simple example of multi-level image segmentation. This example image has been artificially constructed for the purpose of illustration and is composed of four textured

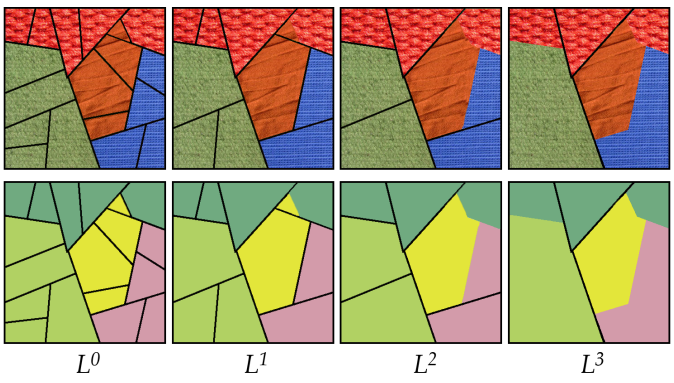

Fig. 4. Top row: a hierarchy of segmentations from fine, L^0, to coarse, L^3 for an artificially constructed image composed of four texture regions. Black edges representing segmentations boundaries at the given level are superimposed over the image. Bottom row: user-collected segmentation shown as four colored regions.

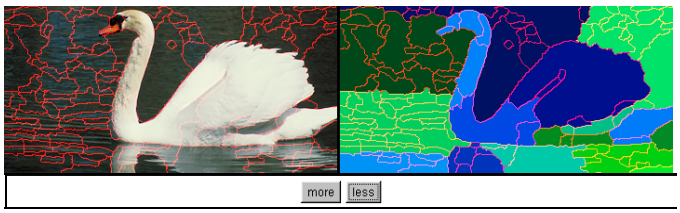

Fig. 5. Screen capture of the semi-automatic region collection interface. The original image is shown left and the current state of region collection is shown right with a single level of the segmentation hierarchy superimposed over both. Pressing the 'more' or 'less' buttons will increase or decrease the amount of overlaid fragmentation.

regions. A hierarchy of segmentations are shown superimposed on the image and are labeled from the finest at L^0 to the coarsest at L^n. We use the notation of L_i^j to refer to fragment i at level j in the segmentation hierarchy. Higher segmentation levels contain fragments (shown as black edges) of larger and larger sizes. Note how in this artificial example there is no individual segmentation level that offers a correct segmentation of the image. L^0 oversegments the image and level L^3 undersegments the image. Levels L^1 and L^2 are more accurate but not completely correct. Furthermore, when dealing with more subtle images, one user's notion of 'correct' may differ from those of others for a given application. We will return to the construction of this multi-level segmentation hierarchy shortly; first we will discuss its use from the user's perspective.

In our interface we allow the user to start with a given segmentation level and subsequently combine or split regions as necessary to form the desired final segmentation. When forming the final segmentation image, the user is able to collect fragments at any level in the segmentation hierarchy to form regions, I_r which are shown as colored areas in the bottom row of figure 4. For example, the user can begin the process at a higher (coarser) level such as L^2, and proceed to lower (finer) levels to fine-tune the region boundaries. Collection at the lowest level of segmentation (produced via the watershed transform) acts as a catchall in that the user can fall back on small fragment collection, should the segmentation perform poorly in a subset of the image. This mode of interaction requires us to differentiate between the fragments, L_i^j, that are automatically generated in the segmentation hierarchy from the regions, I_r, that the user has collected. It is the regions, I_r, which together comprise the final segmentation.

The relationship between fragments in the original segmentation hierarchy and the user-collected regions can be seen in the bottom row of figure 4. The final user-collected regions are shown underneath the computer-segmented fragments. For example, at level L^2 we see six computer-segmented fragments (indicated with their black edges) and four user-collected regions (colored purple, light green, yellow and dark green). At any point in time the user operates at one level, j, of the hierarchy only, seeing the fragments of level j overlaid onto the current user-collected regions. The user is able to move up or down the fragmentation hierarchy with a simple two-button interface.

For further clarity a screen capture of the region collection interface is shown in figure 5. The original image is shown to the left and the current state of region collection is shown to the right. The user can work in either the left or right image areas. A single level of the segmentation hierarchy is superimposed over both the right and left image areas. The level j is controlled with the *more* and *less* buttons, referring to more or less segmentation.

The superimposed segmentation edges are shown in red and since we do not wish the user-collected regions to conflict with the edge overlay, the user regions are assigned colors strictly within the blue and green channels. However, the user has the option to change the edge overlay to green or blue, which will automatically recolor the user-collected regions into the remaining color channels.

In this multi-scale "fragment and collect" approach, the user moves between the multiple levels and performs two simple operations: join fragments to an existing region and form a new region. The two operations are controlled with the left and right mouse buttons, respectively. For adding fragments to an existing region, the user begins by holding down the left button while positioned over the region that s/he wishes to add further fragments to. All subsequent fragments that the mouse moves across are added to the existing region. In our chosen notation, the user selects region I_r with the initial click of the mouse. The mouse then passes over a subset, m, of all fragments L^j at level j. I_r is then updated to:

$$I_r = \bigcup_{m \in L^j} L_m^j \bigcup I_r \tag{1}$$

A typical use of this function is to add small fragments to a large existing region in order to fine-tune the boundary of that region.

The second function involves the same interaction except that the user holds down the right mouse button, and in this case all selected fragments are collected into a new region, s:

$$I_s = \bigcup_{m \in L^j} L_m^j \tag{2}$$

This is useful when a portion of the image has been undersegmented and the user wishes to form new, smaller regions.

We now briefly address the construction of the segmentation hierarchy itself and for this we use two methods of segmentation. The first is a standard watershed transform which generates L^0 of the hierarchy. The second method is the color image segmentation method of Deng and Manjunath [12]. This second method of unsupervised segmentation is used to generate all higher levels of segmentation. We choose this method for two reasons. The first is that color quantization plays a key role in the segmentation process. This is important since real stained glass tends to segment imagery into coherent regions of color. The second reason is that the segmentation process involves the merging of smaller regions into larger regions, which is precisely what is required to construct the segmentation hierarchy. The extent of region merging is controlled with a threshold parameter, t. Our segmentation hierarchy uses 5 levels above the watershed level, L^0, with t set to 0.0, 0.2, 0.4, 0.6 and 0.8, respectively.

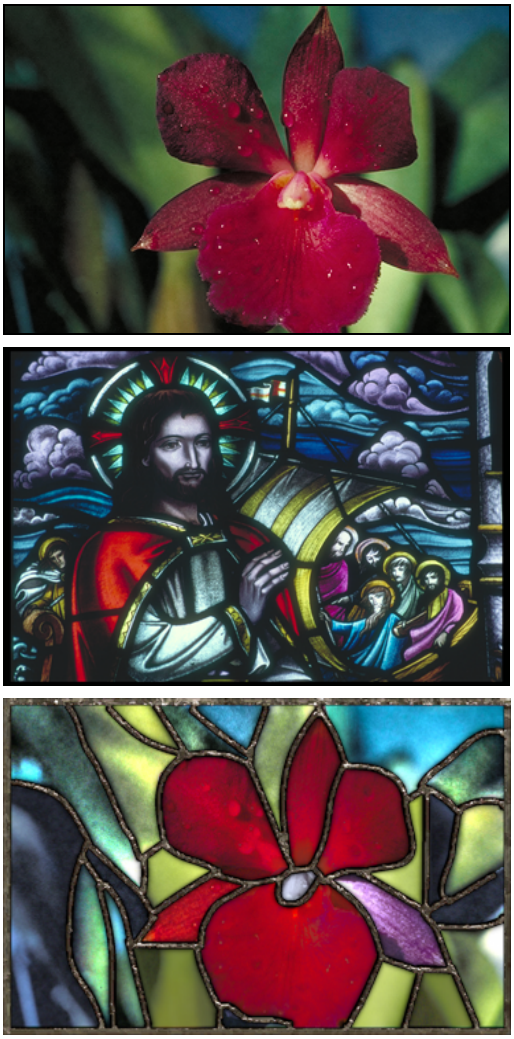

Fig. 6. Color statistics from target stained glass image T_q (middle) are transferred to input image I_r (top) on a region-by-region basis. Results shown with synthesized calmes (bottom).

The segmentation process is performed on both the input image, I, and the target stained glass image, T. In general, the segmentation of the target stained glass image is trivial due to the existing calmes and can be pre-computed for repeated use with many input images.

6 Region Analysis

In section 7 we will discuss the region-by-region colorization of the input image using the color statistics of individual regions, T_q, selected from the target stained glass image. But before doing so, we need to determine which region of stained glass, T_q, we will use to colorize a given region, I_r, of the input image. Our criterion is to select the region T_q which is most similar to I_r. This introduces the familiar problem of image querying but in an unfamiliar form. Instead of querying for whole images in a large database, we query for the closest matching region of stained glass, T_q, within a single image, T.

Finding the closest match between two regions requires the computation of region feature vectors and a metric to compute distances between two vectors. For the computation of feature vectors we adapt the analysis method of Carson *et al.* [11], which reduces our regions to concise sets of color, C_r, and texture, D_r, descriptors.

The color feature vector, C_r, is computed for a region, I_r, as a color histogram with a bin width of 20, in the perceptually-based L*a*b* space. This yields 5 bins in the L* dimension and 10 bins in each of the a* and b* dimensions. This totals 500 bins, though only 218 bins fall within the gamut corresponding to $0 \le (R, G, B) \le 1$. The distance metric between any two color feature vectors C_r and C_q is computed as:

$$d^2_{hist}(C_r, C_q) = (C_r - C_q)^T A (C_r - C_q) \tag{3}$$

where $A = [a_{ij}]$ is a symmetric matrix of weights between 0 and 1 representing the similarity between bins i and j based on the distance between bin centers. Neighboring bins are given weight of 0.5 and bins that are two units away are assigned a weight of 0.25. All other weightings are set to 0. This weighting makes the distance calculation more robust as it allows us to match similar colors that do not exactly fall in the same bins. This is critical since we know in advance that exact matches will not be found.

Unlike color, texture is a local neighborhood property and the texture descriptor for a region must provide a description of the texture features computed over a neighborhood size appropriate to the local structure. Our texture descriptor, D_r, for region, I_r, is comprised of the mean texture contrast, c, computed with automatic scale selection. The texture contrast is derived from the second moment matrix of gradient vectors within a Gaussian window of size σ. The second moment matrix is calculated at pixel (x, y) as:

$$M_\sigma(x, y) = G_\sigma(x, y) * (\nabla I)(\nabla I)^T \tag{4}$$

where $G_\sigma(x, y)$ is a Gaussian kernel with variance σ^2 and ∇I is the gradient of image intensity. The window scale, $\sigma(x, y)$, itself varies across the image and this scaling is automatically determined (see [11] for details on automatic scale selection). Texture with a repeat period of up to 10 pixels is recognized.

At each location, $M_\sigma(x, y)$ is a 2×2 symmetric positive semi-definite matrix and we compute the eigenvalues of $M_\sigma(x, y)$ as $\lambda_1(x, y)$ and $\lambda_2(x, y)$. From the eigenvalues λ_1 and λ_2 the texture contrast is computed as:

$$c(x, y) = 2\sqrt{\lambda_1(x, y) - \lambda_2(x, y)} \tag{5}$$

Fig. 7. Texture contrast (right) computed for a butterfly image (left). The contrast is computed with automatic scale selection.

An example of $c(x, y)$ is shown in figure 7 for a butterfly image. Note that because automatic scale selection has been used, the texture contrast function responds to texture contrast at a variety of scales up to a 10 pixel period. The texture descriptor, D_r, for region I_r is simply defined as $D_r = (\bar{c}_r)$, being the average texture contrast over all pixels in the region. The distance metric between any two texture descriptors D_r and D_q is simply:

$$d^2_{text}(D_r, D_q) = (D_r - D_q)^2 \quad (6)$$

The combined distance metric between two regions, I_r and T_q, for both color and texture descriptors becomes:

$$d^2(I_r, T_q) = d^2_{hist}(C_r, C_q) + \alpha \times d^2_{text}(D_r, D_q) \quad (7)$$

Where α is a constant weighting of the texture distance. In all our examples, we use an α value of 10, which addresses the difference of magnitude between the color and texture distance measures while retaining a greater emphasis on color matching.

Similar to standard approaches to image querying, we have included the texture descriptors for our regions so as to more broadly capture each region's appearance. We argue that this is important from the user's perspective since the perception of image regions is influenced by both color and texture [26]. However, for further user control we have added a simple binary option that determines whether the matching is performed using the default (color+texture) descriptors or with color alone. Offering this choice has a positive side effect in that it makes the matching criteria more explicit and transparent, though it is important to note that all of the examples shown in this paper use the default (color+texture) descriptors.

With the region descriptors computed for all I_r in the input image and all T_q in the target stained glass image, we select the region T_q with the smallest distance, $d^2(I_r, T_q)$.

7 Regional Colorization

Color transfer allows one image to adopt the color characteristics of another. Recently, there have been significant advances in the area of color transfer, some based on texture synthesis [8] and others on the direct manipulation of color statistics. We now introduce the concept of region-by-region colorization wherein image regions I_r are colorized independently, and for this, an unsupervised statistical approach such as those described by Reinhard *et al.* [9] and Hertzmann *et al.* [5] is most appropriate.

To perform the color transfer we coerce the pixel data of the input region, I_r, so that the mean and standard deviation values of the three color channels of I_r are made equal to those of the best matching region, T_q, from the target stained glass image. In order to affect the statistics of individual color channels we must take care to operate in a color space with de-correlated color channels. Otherwise, unwanted cross-channel artifacts would occur. The $l\alpha\beta$ color space introduced by Ruderman *et al.* [27] has been shown by Reinhard *et al.* [9] as having sufficient channel de-correlation for the purpose of statistical color transfer. This $l\alpha\beta$ space is an opponent-color model where l represents an achromatic channel, while α and β are chromatic yellow-blue and red-green opponent channels, respectively.

With the pixel data represented in $l\alpha\beta$ space the transfer of color statistics from stained glass region T_q to the input region I_r proceeds. The mean, μ, and standard deviations, σ, of the each of the three $l\alpha\beta$ color channels are indirectly transferred to I_r by manipulating the $l\alpha\beta$ pixel values of I_r. To this end, the channel means from I_r are first subtracted:

$$l^* = l - \bar{l}, \quad \alpha^* = \alpha - \bar{\alpha}, \quad \beta^* = \beta - \bar{\beta} \quad (8)$$

Next, the data is scaled by the ratio of standard deviations from I_r and T_q, with channel means from T_q added:

$$l' = \frac{\sigma_q^l}{\sigma_r^l} l^* + \mu_q^l, \quad \alpha' = \frac{\sigma_q^\alpha}{\sigma_r^\alpha} \alpha^* + \mu_q^\alpha, \quad \beta' = \frac{\sigma_q^\beta}{\sigma_r^\beta} \beta^* + \mu_q^\beta \quad (9)$$

where (μ_r^z, σ_r^z) and (μ_q^z, σ_q^z) are the means and standard deviations of regions I_r and T_q respectively, over some channel z.

With the most similar region, T_q, found for each region I_r in the input image, the color statistics are transferred on a region-by-region basis, producing new regions, J_r. Figure 6 shows the results of this process. The presence of lead stripping, shown between each region of the output image, will be discussed in the following section.

8 Synthesis of Calmes

A unique feature of stained glass is that it is held together with lead, zinc, brass or copper strips called calmes. The calmes are not always visible in photographs taken of stained glass artwork due to the typically high dynamic range caused by natural light passing through the glass. But they are there nonetheless. To increase the realism of the produced stained glass we have developed an automatic method of synthesizing calmes based on recent advances in pixel-based texture synthesis. In particular, we adapt the Image Analogies framework of Hertzmann *et al.* [5], though any method of directed texture synthesis might serve.

Before the synthesis can begin, some pre-processing of the region boundaries is required to mitigate their excessively high frequencies by fitting each edge to a cubic smoothing spline [28]. Figure 8 provides a comparison of region boundaries shown before (left) and after (right) being fitted with smoothing curves.

Fig. 8. Region boundaries before smoothing (left) and calmes mask after smoothing (right).

Fig. 9. Two pairs of Image Analogy images. Top row: source mask (left) and source image (right). Bottom row: target mask (left) and final synthesis (right).

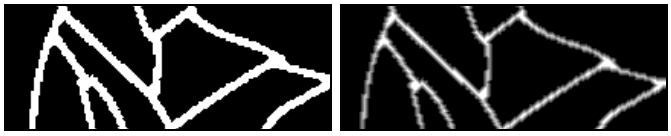

Fig. 10. Target mask with (right) and without (left) linear feathering.

The reader may have noticed that the smoothed set of curves shown to the right in figure 8 already forms an Image Analogies [5] mask, which is needed for the directed synthesis of the calmes. In the notation of Image Analogies, the texture-by-numbers re-synthesis of an image is expressed as a filtering operation. Given a set of three images A, A' and B where A is the unfiltered source, A' is the filtered source and B is the unfiltered target image, we wish to synthesize the new filtered target image B' such that:

$$A : A' :: B : B' \tag{10}$$

By this we mean that we wish to find the analogous image B' that relates B' to B as A' relates to A.

The Image Analogies algorithm synthesizes each new pixel in the output image, in scan-line order, by finding pixels with matching local neighborhoods in the original texture. Where the Image Analogies algorithm differs from regular texture synthesis is with the treatment of the analogous relation of B:B' to A:A'. When selecting the next synthesized pixel, the local neighborhood comparison uses a concatenation of the neighborhood in the input texture, A', with the corresponding neighborhood in the input mask, A.

Figure 9 shows two pairs of Image Analogy images with the original lead calmes image, A', (top-right), its mask, A, (top-left), the target mask, B, (bottom-left) and the final synthesis, B', (bottom-right). The analogy therefore specifies that the input calmes relate to the input mask as the output calmes must relate to the automatically constructed output mask. The output mask for the synthesis process is comprised of the set of smoothed region-edge curves drawn in white with a radial width of 10 pixels. The input calmes are taken from a real image of stained glass and its associated mask is easily created with our segmentation interface. We note that the same interface could be used more generally for constructing image analogies masks.

There are a number of important points concerning the synthesis process. In order to achieve the best possible synthesis, both the input and output masks are structured with linear feathering. This can be seen in figure 10, which shows a zoomed-in portion of the Image Analogies target mask, B. The left image shows the curves that are generated from the cubic-spline fitting process. The right image shows the same curves but with feathering applied. The feathering is applied to both the source mask, A, and the target mask, B, which better structures the synthesis process. The synthesized calmes can be further enhanced if they are synthesized 50% larger than what is required. Afterwards, the output is scaled down to the needed size, which has the effect of removing high frequency discontinuities that can result from pixel-based texture synthesis. However, this super-sampling of calmes does not significantly affect the overall processing time, as only a small percentage of the image area need be synthesized (i.e. the region boundaries).

It is also worth noting that the calmes input image and its mask can be prepared in advance and used repeatedly. In this way the calmes synthesis process can be made fully automatic, requiring no user intervention. Moreover, multiple calmes styles can be prepared in advance. The bottom row of figure 15 shows features copper stripping, which has been sourced from an alternate calmes input image.

After synthesis is complete, a final post process is applied to better integrate the synthesized calmes with the colorized regions. Here we again apply a feathering operation but this time to the colorized image. Pixels in the colorized regions are made increasingly darker (at an exponential rate) as they approach neighboring calmes. This subtle improvement makes it appear that the colorized regions are directly fitted to the calmes as can be seen in figure 11.

9 Simulated Glass Filter

Even with colorization applied and lead stripping in place, images may still require further filtering to match the appearance of real stained glass. In particular, if there is too much texture detail in the original image, the final result may look too photorealistic to achieve a convincing stained glass-like appearance. For these images, a simulated glass filter is applied in two phases:

1. Add Perlin noise [29], [30] to the input image I_r prior to colorization.
2. Apply discontinuity-sensitive warping to generate many small facets of color in each region.

The application of this filter results in a loss of high-

Fig. 11. Colorized regions before (left) and after (right) exponential feathering.

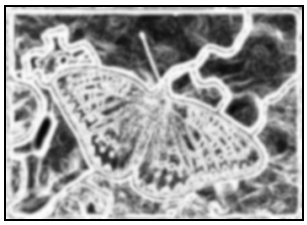

Fig. 12. Absolute magnitude of edge maps produced by Sobel convolution. Gaussian smoothing has been applied.

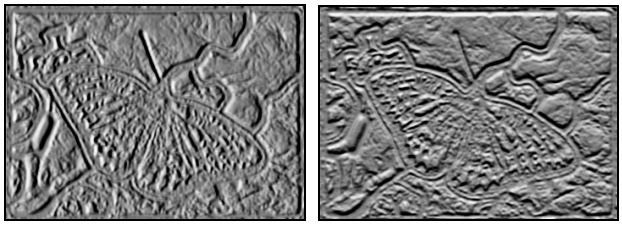

Fig. 13. Horizontal (left) and vertical (right) displacement maps.

frequency details in favor of glass-like facets. Within each facet, changes are made more smooth and gradual with discontinuities pushed to facet boundaries.

The addition of Perlin noise introduces the appearance of imperfections in the glass when colorized. For this we use a standard noise function generated over the four highest frequencies, with all four octaves having the same amplitude of 3/255 (a small percentage of the RGB range). By using only the highest frequencies we introduce fairly uniform and mild imperfections.

The addition of Perlin noise is often sufficient in itself, as is the case with the parrot example in figure 1. But, other images may require stronger approach: facet warping. This new filter is related to the concepts introduced by Arad and Gotsman [31]. In their work they propose an adaptive warping scheme for sharpening and scaling images. The technique operates by "squashing" pixels in edge areas and "stretching" pixels in flat areas. But, instead of contracting only at major edges in the image (which in our case are located under the lead stripping), we contract every minor discontinuity. By designing the warp to be sensitive to small discontinuities, a myriad of small facets of color are produced. Figure 14 shows examples of this.

The filter has multiple stages that distort the input image subject to the presence of discontinuities. The first stage convolves the image with the following Sobel kernels, which respond to edges in the x and y directions:

$$h_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad h_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{11}$$

The combined absolute magnitude map of the two resulting edge images, $G_x(x, y)$ and $G_y(x, y)$, is computed as:

$$|G(x,y)| = \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \tag{12}$$

Next, a Gaussian smoothing kernel is applied to the edge magnitude map, an example of which can be seen in figure 12. The radius of the Gaussian kernel is important here as it has a direct effect on the regularity of the facets: the wider the kernel, the wider and more regular the facets will be. Figure 14 shows the effect of kernel size on the final result. On the left is a single colorized region with Perlin noise added. The center and right images show the same region after filtering is applied. Note how the center image exhibits smaller and less regular faceting due to the smaller kernel size of 0.5. The right image exhibits wider facets using a kernel size of 3. In all examples in this paper a standard deviation (*stdev*) of 3 pixels has been used.

To compute the horizontal and vertical displacement amounts the partial derivatives (d_x, d_y) of $|G(x, y)|$ are estimated by convolution with the following kernels:

$$p_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} \quad p_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{13}$$

The resulting vectors $d_x(x, y)$ and $d_y(x, y)$ shown in figure 13 are used as a displacement map, translating the pixels in the colorized image. The displacement vectors are scaled by a constant factor of 3, though one could provide the user with a slider to control the degree of faceting.

Even for completely smooth regions, the addition of Perlin noise prior to colorization offers a sufficient degree of discontinuity for this filtering operation to work well. Moreover, since the larger edge discontinuities are at region boundaries, excessive distortion of the region interiors does not occur. We also note that the displacement function has a positive side effect on the lead stripping in that the calmes become smoother and less regular. A variety of faceted examples can be seen in figure 15.

10 Solid Glass Replacement

Some stained glass images are entirely comprised of solid pieces of stained glass (without painted detail) and others are formed from a combination of solid glass and painted glass. Similarly, we extend the range of stained glass styles through the replacement of colorized regions with solid glass pieces. A solid glass result is shown in figure 18 and, for reference, a selection of solid glass can been seen in figure 17. When selectively replacing a subset of colorized regions, C, with solid regions, S, it must be determined whether a given region, J_r, should be replaced and, if so, which piece of real solid glass, S_i, will replace it.

We offer two alternative user interfaces for replacement selection. The first simply allows the user to directly toggle solid replacement by clicking anywhere within a region. In the second interface the user controls region replacement by adjusting a slider that dictates the number of regions that must be replaced in the image (see figure 19). For this

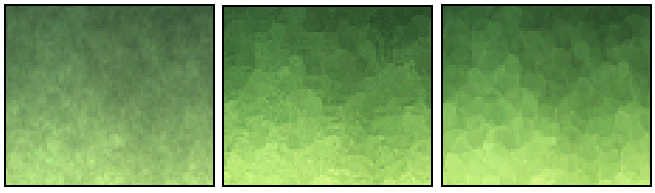

Fig. 14. Left: colorized region. Middle: faceting with Gaussian *stdev* = 0.5. Right: faceting with *stdev* = 3.0.

Fig. 15. A selection of input images (left) is converted into a stained glass style. The target stained glass images are shown in the center column and the final results are shown to the right. Last row: the region matching process for the watery background has been partially randomized. Bottom row with copper calmes, remaining rows with lead.

Fig. 16. A selection of constructed stained glass images are symmetrically tiled across 3D tiffany lampshades. Bottom Left: An example showing a mixture of colorized and solid glass regions.

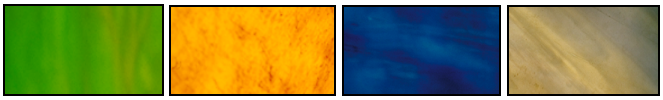

Fig. 17. Solid glass pieces for replacement.

to be meaningful the regions must be ordered in some way. Regions are ordered according to their texture contrast levels as computed in section 6. Regions are ordered from lowest to highest texture contrast. Therefore, when the user sets n number of regions to be replaced the n colorized regions with the least texture contrast are replaced with solid glass. This second interface operates under the assumption that regions of minimal texture are likely to be less important than highly textured regions. The operation of both types of interface is illustrated in the accompanying video.

When a colorized region, J_r, has been selected for replacement, a choice must be made as to which piece of solid glass is used. This returns us to image querying and an analysis of the solid glass database, S, needs to be computed. Texture and color feature vectors are computed for each image S_i in the database, using the same analysis methods described previously in section 6. When searching for a solid match for a colorized region, J_r the same distance metric is also used. But in this case no segmentation is needed, as the solid glass pieces are single regions.

There are a number of final points worth noting that relate to solid glass replacement. The first is that the replacements can be matched to either the colorized regions or the original regions. Matching to the colorized regions maintains the overall style of the target stained glass image, T. The second point is that the simulated glass filtering discussed in the previous section is only applied to the colorized regions and not to solid glass replacement regions, since the solid replacements are real glass to begin with.

We find that the quality of the results is not overly sensitive to the size of the database of solid glass images. Good results can be obtained with a database that has a reasonable coverage of the color space. However, the larger the database, the more accurate the texture matching will be. A last point of efficiency is that the analysis of the database need only be pre-computed once.

11 Results

We present a further selection of results in figure 15. For the examples shown, the average total processing time was approx. 1 *min*. 15 *sec*. on a 3.6 GHz P4, including segmentation and stained glass assembly. User input times for the semi-automatic segmentation varied from image to image. In the cases of the simple flower image in figure 6 and the butterfly image in figure 15, user guided region collection was performed in under a minute. More complicated images such as the parrot image in figure 1 and the architectural image in figure 15 required between 2 and 3 minutes.

In our experiments we found that, at some stages in the process, it is sufficient to work with images down-sampled by 50% in both dimensions. For example, this is applicable to region analysis. Although we super-sample the calmes synthesis, we typically only need to synthesize less than 15% of the total area at region boundaries.

A limitation of our method is that it relies on the input image having a degree of non-uniformity in terms of color. For example, the input image in the last row of figure 15 depicts a white swan on a uniform watery background. Due to this uniformity, the resulting colorization would be uninteresting since all water regions would find the same match in the target image, T. To overcome this we randomize the matching of water regions for colorization (shown right). In a similar fashion to solid glass replacement, we allow the user to toggle whether a region's matching is randomized by clicking on a given region. This is trivial to implement, requiring only that the system select a random match from the target image, T, for each user-click.

A further selection of constructed stained glass images are shown symmetrically tiled across 3D tiffany lampshades in figure 16. These images illustrate one potential use of our results but are not meant to imply that we are proposing a method for the physically based rendering of glass. The top image uses the solid glass example from figure 18. The bottom left image uses a mixture of colorized and solid glass and is derived from the butterfly example of figure 15. The appearance of the middle image in the bottom row is somewhat dominated by the symmetrically tiling, however, if one looks closely, the parrots of figure 1 can clearly be seen. The remaining example is derived from a tree and mountain scene. A convenient side effect of our approach is that the calmes mask can be used as both transparency and bump maps when texturing.

12 Conclusion & Future Work

The simulation of the stained glass form is a challenging problem requiring a multi-stage solution. Producing plau-

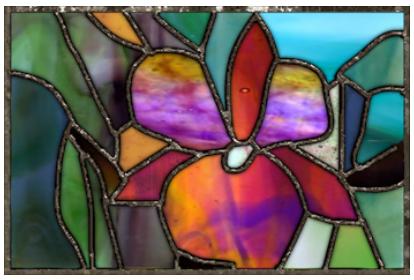

Fig. 18. A solid stained glass example shown with synthesized calmes.

Fig. 19. As the slider values increase, more regions are replaced with solid glass, by order of increasing texture contrast.

sible stained glass renditions has required image warping, colorization, segmentation and querying as well as texture synthesis. This has resulted in a considerable improvement of results over existing approaches.

In addition, there are aspects of our work that are more broadly applicable. Our image segmentation framework could be applied to any segmentation task. For instance, it could used to construct Image Analogies masks [5]. Facet warping could be used as a stand-alone glass filter. And, the concept of region querying within a single image might be used in conjunction with digital photomontage [25] for the automatic replacement of regions with similar queried regions taken from a target image.

Our approach requires a modest amount of human guidance to initially segment the image into regions. Although we argue that robust segmentation under human guidance is worth the cost, it might be preferable to increase automation even further. As most segmentation algorithms could be adapted to our framework, future advances in segmentation might reduce user involvement.

Another segmentation related issue regards the potential use of our system for designing real stained glass templates. Although we have not focused on generating patterns for real stained glass in the present work, future work might integrate constraints in the user-guided segmentation process that would ensure that segments are 'cuttable' in real glass. For this, it may be sufficient to highlight segments that are potentially problematic for cutting. The user could then further dissect the problem segment or this could be automated at the user's request.

Though outside the scope of this paper, other avenues of future research might include the physical simulation of glass with non-homogeneous pigments or the physical simulation of broken or chipped glass. Another possible extension to the current work could allow the user to generate geometric patterns into low-texture regions. An alternate Design Gallery [32] interface might be adopted for exploring multiple stained glass renderings of the same input image, using a structured database of target stained glass imagery. This would be useful for exploring randomized variations as well. It might also be possible to develop a system that converts video into a stained glass style.

REFERENCES

[1] BROWN, G. 1998. *Stained glass windows*. Celtic Cross Press.

[2] COHEN, M., SHADE, J., HILLER, S. AND DEUSSEN, O. 2003. Wang Tiles for Texture and Image Generation. *ACM Trans. on Graphics*, 22, 3,287-294.

[3] EFROS, A. AND FREEMAN, W. T. 2001. Image Quilting for Texture Synthesis and Transfer. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, 341-346.

[4] EFROS, A. AND LEUNG, T. K. 1999. Texture Synthesis by Non-parametric Sampling. In the *7th IEEE International Conference on Computer Vision*, 1033-1038.

[5] HERTZMANN, A., JACOBS, C., OLIVER, N., CURLESS, B., AND SALESIN, D. 2001. Image Analogies, In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 341-346.

[6] KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut Textures: Image and Video Synthesis Using Graph Cuts. *ACM Transactions on Graphics*, 22, 3, 277 -286.

[7] WEI, L., AND LEVOY, M. 2000. Fast Texture Synthesis Using Tree-Structured Vector Quantization. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 479-488.

[8] WELSH, T., ASHIKHMIN, M. AND MUELLER, K. 2002. Transferring Color to Greyscale Images. *ACM Trans. on Graphics*, 21, 3, 277–280.

[9] REINHARD, E. ASHIKHMIN, M., GOOCH B. AND SHIRLEY, P. 2001. Color Transfer Between Images. *IEEE Computer Graphics and Apps.*, 34-40.

[10] LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization Using Optimization. *ACM Transactions on Graphics*, 23, 3, 689–694.

[11] CARSON, C., THOMAS, M., BELONGIE, S., HELLERSTEIN, J. M., AND MALIK, J. 1999. Blobworld: A system for region-based image indexing and retrieval. In the *Third International Conference on Visual Information Systems*, 509-516.

[12] DENG, Y. AND MANJUNATH, B. 2001. Unsupervised Segmentation of Color-Texture Regions in Images and Video. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI '01)*, 800-810.

[13] ADOBE. 2003. *Adobe Photoshop CS Manual*. Adobe Systems, USA.

[14] MOULD, D. 2003. A Stained Glass Image Filter. In *Proceedings of the 13th Eurographics Workshop on Rendering,* ACM International Conference Proceeding Series, 20-25.

[15] GOOCH B. AND GOOCH A. 2001. *Non-photorealistic Rendering.* A.K. Peters Ltd. Publishers.

[16] ELBER, G., AND WOLBERG, G. 2003. Rendering Traditional Mosaics. *The Visual Computer* 19(1), 67-78.

[17] HAUSNER, A. Simulating Decorative Mosaics. 2001. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 573-580.

[18] KIM, J., AND PELLACINI, F. 2002. Jigsaw Image Mosaics. *ACM Transactions on Graphics*, 21, 3, 657-664.

[19] DECARLO, D., AND SANTELLA, A. 2002. Stylization and abstraction of photographs. *ACM Transactions on Graphics*, 21, 3, 769-776.

[20] BANGHAM, J.A. AND GIBSON, S.E. AND HARVEY, R.W. 2003. The Art of Scale-Space. *British Machine Vision Conference*, vol. 1.

[21] BARRETT, W., AND CHENEY, A. 2002. Object-based image editing. *ACM Transactions on Graphics*, 21, 3, 777-784.

[22] KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1987. Snakes: Active Contour Models. *Inter.l Journal of Computer Vision*, 1, 4, 321-331.

[23] MORTENSEN, E. AND BARRETT, W. 1995. Intelligent Scissors for Image Composition. In ACM *SIGGRAPH '95 Proceedings*, 191-198.

[24] ROTHER, C., KOLMOGOROV V., AND BLAKE, A. 2004. GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Trans. on Graphics*, 23, 3, 309-314.

[25] AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, A., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive Digital Photomontage. *ACM Trans. on Graphics*, 23, 3, 294-302.

[26] MANJUNATH, B. S., OHM, J., VASUDEVAN, V. AND YAMADA, A. 2001. Color and texture descriptors. *IEEE Transactions on Circuits and Systems for Video Technology,* 11:6, 703-715.

[27] RUDERMAN, D., CRONIN T., AND CHIAO, C. 1998. Statistics of Cone Responses to Natural Images: Implications for Visual Coding. *Journal of the Optical Society of America*, 15(8), 2036-2045.

[28] REINSCH, C. 1967. Smoothing by Spline Functions. In *Numerical Mathematics*, 10, 177- 183.

[29] EBERT, D., MUSGRAVE, K., PEACHEY, P., PERLIN, K., AND WORLEY, S. 1998. *Texturing and Modeling; A Procedural Approach.* AP.

[30] PERLIN, K. 1985. An Image Synthesizer. In *Computer Graphics* (*SIGGRAPH '85 Proceedings*), 287-296.

[31] ARAD N. AND GOTSMAN C. 1999. Enhancement by Image-Dependent Warping. *IEEE Trans. on Image Processing*, 8, 8, 1063-1074.

[32] MARKS, J., ANDALMAN, B., BEARDSLEY, P. A., FREEMAN, W., GIBSON, S., HODGINS, J., KANG, T., MIRTICH, B., PFISTER, H., RUML, W., RYALL, K., SEIMS, J., AND SHIEBER, S. 1997. Design Galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of SIGGRAPH '97*. 389-400.

Stephen Brooks is an assistant professor of computer science at Dalhousie University. He received his BSc from Brock University, his MSc degree from the University of British Columbia, and his PhD in computer science from Cambridge University in 2004. His research interests include image editing, non-photorealistic rendering, human motion editing and 3D GIS. Stephen also has a background in traditional drawing, with conté crayon, ink and pencil.