Eurographics Symposium on Rendering (2006)
Tomas Akenine-Möller and Wolfgang Heidrich (Editors)

Painting With Texture

Lincoln Ritter[1] Wilmot Li[1] Brian Curless[1] Maneesh Agrawala[2] David Salesin[1,3]

[1]University of Washington, [2]University of California at Berkeley, [3]Adobe Systems

Abstract

We present an interactive texture painting system that allows the user to author digital images by painting with a palette of input textures. At the core of our system is an interactive texture synthesis algorithm that generates textures with natural-looking boundary effects and alpha information as the user paints. Furthermore, we describe an intuitive layered painting model that allows strokes of texture to be merged, intersected and overlapped while maintaining the appropriate boundaries between texture regions. We demonstrate the utility and expressiveness of our system by painting several images using textures that exhibit a range of different boundary effects.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation - Display algorithms I.3.3 [Computer Graphics]: Methodology and Techniques - Interaction techniques I.4.7 [Image Processing and Computer Vision]: Feature Measurement - Texture

1. Introduction

Since the introduction of SketchPad [Sut63] in 1963, drawing or painting with a handheld input device has become one of the most popular interaction modes for creating digital images. Direct painting interfaces are both simple and intuitive; the act of applying digital paint to a digital canvas is closely analogous to the act of applying real paint to a real canvas. Today, every industry-standard image creation and design tool (including Adobe Photoshop, Adobe Illustrator, etc.) allows the user to make marks on a digital canvas by applying digital paint.

Recent advances in texture synthesis have the potential to significantly increase the expressive power of digital painting tools. By enabling the reproduction of arbitrary amounts of a given sample texture, modern synthesis algorithms open up the possibility of using textures as another form of digital paint. The user should be able to choose a set of example textures to form a palette of texture paints that could then be applied interactively. By painting with texture, the user could create images that exhibit the complex detail and subtle variations of the example textures with a relatively small amount of effort.

In this work, we present an interactive texture painting application that allows the user to create digital images by painting with a palette of input textures. At the core of our system is a texture synthesis algorithm that addresses three main challenges.

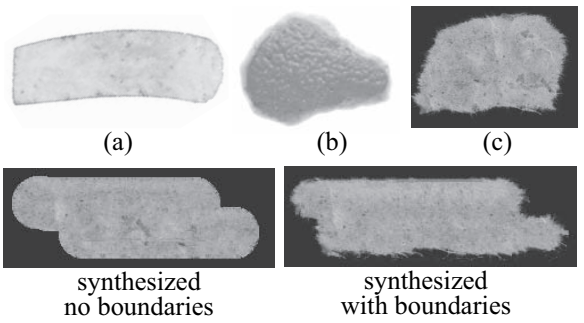


Figure 1: *Boundary effects. Top row shows boundary effects in a stroke of watercolour paint (a), orange peel (b), and torn piece of paper (c). Bottom row shows synthesized paper texture with and without boundary effects. Without boundaries, the edge of the texture looks artificial.*

Boundary effects. The boundaries of many interesting textures have a different appearance than their interiors. Figure 1 shows three examples of such boundary effects. Painting with texture produces images comprised of multiple distinct textures, and in such images, boundary effects are crucial for producing natural transitions between regions of dif-

ferent texture. Without the appropriate boundary effects, interfaces between adjacent textures tend to look artificial. Most existing texture synthesis algorithms generate an infinite expanse of a particular texture without considering the texture's appearance at its perimeter.

Interactive texture synthesis. Synthesizing high-quality textures can be a computationally expensive task that is often difficult to perform at interactive rates. However, an interactive texture painting system must be able to generate texture in a responsive manner, as the user paints.

Combining strokes. In a typical usage scenario, multiple strokes of texture paint will overlap when a user paints with texture. Thus, a texture painting system must define what it means when regions of texture overlap and provide the user with an intuitive interface for combining strokes.

The techniques we have invented to address these challenges represent our primary contribution. Specifically, we have developed a texture synthesis algorithm that extends existing methods to produce textures with boundary effects and alpha information. We present an implementation of this algorithm that performs well enough to enable interactive feedback within the context of our texture painting interface. Finally, we describe a layered painting model for combining overlapping regions of texture paint that is simple enough to be intuitive for the user, and expressive enough to support a wide range of composite texture effects.

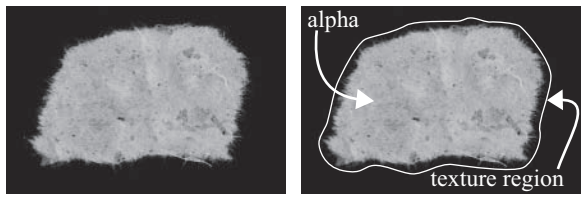
2. Related work

This work draws inspiration from three sources: interactive painting software, non-photorealistic rendering, and texture synthesis.

Applications like Adobe Photoshop, Adobe Illustrator, and the GIMP have become mainstays of digital artwork creation. While very powerful, creating high quality images from scratch using these programs takes a great deal of time and skill. Some systems (like Photoshop) support the incorporation of digital photographs or scans as part of the authoring process, but the user is limited to essentially filtering and copying parts of the existing media. For example, the user might painstakingly piece together new images using the cloning and/or healing brush in Photoshop.

A great deal of work has been done emulating particular artistic media, including watercolour [CAS*97], pen-and-ink [WS94], and ink on absorbent paper [CT05]. Each of these systems has been carefully engineered to reproduce the visual characteristics of a specific medium. In contrast, our system enables the reproduction of a broad range of media through the use of example-based texture synthesis.

Our texture synthesis algorithm is based on the non-parametric sampling methods proposed by Efros and Leung [EL99]. In addition, we incorporate techniques for preserving coherent patches of texture proposed by



RGBA image binary mask

Figure 2: *Input texture. Left shows RGBA input image. Right shows binary mask/texture region along with the alpha channel.*

Ashikhmin [Ash01], who also introduced the concept of user controllable texture synthesis through a user-specified initialization image. Both these and subsequent approaches [EF01, KSE*03, WY04, LH05] have focused on generating an infinite expanse of relatively uniform texture and have not addressed the issue of texture boundary effects.

A few recent results have made great strides towards interactive texture synthesis. Patch-based methods, such as Jump Maps [ZG02], have been used primarily for interactive texture synthesis on surfaces. Patch-based techniques, at least in their current form, cannot be used in our application since they do not provide any mechanism for handling the special appearance of texture boundaries — although this is an interesting area for future work. The pixel-based methods of [LH05] and [WL02] rely on precomputation of neighborhood search candidates to perform parallel synthesis. As currently formulated, these methods are inappropriate for boundary synthesis since the synthesis region needs to be "dilated" at coarser resolutions. This dilation relies, once again, on the assumption that the texture to be synthesized is relatively uniform and infinite in all directions.

Our work most directly relates to the Image Analogies "texture-by-numbers" technique [HJO*01], which works by "filtering" a false-coloured image of labels based on an example labels/filtered-image pair. However, texture-by-numbers treats labels as colour values, comparing them using colour-space distance. This metric can cause poor neighborhood matching and low quality results due to the erroneous combination of the texture channels and label channels. Our work treats labels as ordinal, not metric, values and so is not subject to these limitations. Additionally, texture-by-numbers has no ability to treat pixels from adjacent texture regions independently. Thus, texture-by-numbers is unable to reproduce convincing texture boundaries for interfaces not present in the example pair. This limitation prevents the user from "mixing and matching" a wide variety of textures from disparate sources. Our system circumvents this problem by using a novel energy function that takes texture regions into account.

3. System overview

To create an image using our system, the user first provides a set of example textures that represent a palette of texture

paints. Once these textures are loaded into the system, the user can select a texture and then paint with it on a canvas provided by the application. As the user moves the virtual brush, the system dynamically synthesizes the selected texture in the newly painted region of the canvas.

The palette of example textures represents the only required input to our system. Each texture is specifed as an RGBA image along with a binary mask, created by the user, that identifies the *texture region* — i.e., the region of the image that contains the texture. Locations in the mask with a value of 1 (typically, all parts of the texture with non-zero alpha) represent the pixels within the texture region, called *texture-region pixels*. These pixels define the texture's appearance (see Figure 2). When the texture has a boundary effect, it is defined by the pixels near the perimeter of the texture region. Optionally, the user can provide an additional file that contains the specific parameter values to use when synthesizing each texture. These parameters are described in Section 4. There is no conceptual limit on the size or resolution of the input images, though, in practice, the use of memory-intensive ANN search structures restricts the size of each texture. Most textures we have experimented with are smaller than 400×400.

To fill a painted texture region with the selected example texture, we extend the pixel-based texture synthesis approach used in the Image Analogies system of Hertzmann et al. [HJO*01]. In the following section, we describe the overall algorithm at a high level. The subsequent two sections focus on the specific novel techniques we developed to capture boundary effects and make the synthesis interactive.

4. Pixel-based texture synthesis

Given an example texture A and a target texture region B in which to synthesize the texture, our system proceeds using the basic approach presented in Image Analogies [HJO*01]. Target pixels are synthesized, in scanline order, by copying the RGBA values of pixels from the example texture. To determine which example pixel to copy into a location b in B, the algorithm looks for the pixel in A whose local neighborhood is most similar, under a prescribed energy function, to the neighborhood of b.

To find the most similar example neighborhood, the algorithm performs two searches, a data search, and a coherence search [Ash01]. The *data search* attempts to find the best matching neighborhood over all of A. The *coherence search* attempts to preserve coherent regions of texture in the result by limiting the search to example patches adjacent to previously copied pixels. A coherence parameter κ is used to help choose between the results of these two searches. A specific κ value can be associated with any example texture as part of the input to the system. The default value for κ in our system is 2. The user also has the option of disabling the coherence search.

To capture features at different scales, the algorithm runs at multiple resolutions. Gaussian image pyramids are constructed for both A and B, and the synthesis is performed at each level of the pyramid, from coarsest to finest using multi-scale neighborhoods. At each level, the algorithm initializes the target region with the synthesized result from the previous level. The size and number of levels of the search neighborhood can be adjusted to synthesize textures with features of different sizes. As with κ, a specific neighborhood size and the number of Gaussian pyramid levels can be specified for any example texture. We have found that two-level 7×7 neighborhoods work well for a variety of textures, and we use this as our default neighborhood size.

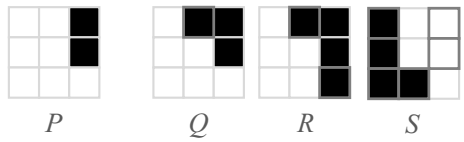


Figure 3: *Neighborhood shape. White squares represent texture region pixels, and black squares represent non-texture-region pixels. The layout of white pixels determines neighborhood shape. If we compare P to Q, R, and S, mismatching pixels (in pink) indicate shape differences. Based on the number of mismatches, Q is most similar to P with one, R is next with two, and S is the least similar with six mismatches.*

5. Synthesizing boundary effects

The key component of the synthesis algorithm is the energy function used to compare two texture neighborhoods. Roughly speaking, we want this energy function to return a lower energy for neighborhoods that look more alike, including neighborhood apprearance at texture boundaries.

As with previous techniques, we consider the colours of the pixels within each neighborhood as part of our metric. In addition, when comparing two neighborhoods that are near texture boundaries, our energy function penalizes differences in "neighborhood shape," by which we mean the layout of texture region-pixels within a neighborhood (see Figure 3). Thus, when the system synthesizes a pixel near the boundary of the target region, the energy function favours pixels near similarly shaped boundary neighborhoods in the example texture. This approach allows our system to capture variations in the texture boundary that depend on the curvature and orientation of the border.

We compute the energy between two neighborhoods as follows. Given two equal-sized neighborhoods P and Q, our energy function computes an energy contribution for each pair of corresponding pixels and then returns a normalized sum of these contributions as the total energy. Pixels are compared in one of three different ways:

1. If both pixels are texture-region pixels, the energy contribution is the sum of squared differences between their RGBA values.

2. If both pixels are non-texture-region pixels, we consider them to be perfectly matched. Thus, the contribution to the total energy is zero.
3. Finally, if one pixel is a texture-region pixel and the other is not, there is a difference in the shape between the two neighborhoods. In this case, the pixels contribute a very large energy value that penalizes the shape mismatch.

Here, we give the formal definition of the energy function. Let P and Q be two neighborhoods of size n comprised, respectively, of pixels $\{p_1,...,p_n\}$ and $\{q_1,...,q_n\}$. We use T_P and T_Q to denote the texture regions associated with these two neighborhoods. Let $d(p,q)$ represent the sum of squared differences between the RGBA values at pixels p and q. Finally, let e_{max} be the penalty for a shape mismatch. Typically, we set e_{max} large enough so that shape mismatches completely override colour matches. The energy $E(P,Q)$ between P and Q is defined as follows:

$$E(P,Q) = \frac{1}{n}\sum_{i=1}^{n} e(p_i,q_i)$$

$$e(p,q) = \begin{cases} d(p,q) & \text{if } p \in T_P \text{ and } q \in T_Q \\ 0 & \text{if } p \notin T_P \text{ and } q \notin T_Q \\ e_{max} & \text{if } p \in T_P \text{ xor } q \in T_Q \end{cases}$$

Note that if all the pixels in both neighborhoods are texture-region pixels, the energy function essentially measures the L_2 colour distance between the two neighborhoods. Thus, away from the texture boundaries, our energy function is identical to the colour-based metrics used in previous algorithms [HJO*01]. However, near boundaries, both shape and colour are used to evaluate neighborhood similiarity.

6. Synthesizing texture interactively

To enable a smooth painting interaction, our synthesis algorithm must generate texture at interactive rates. Recent work by Lefebvre and Hoppe [LH05] presents a very fast GPU-based synthesis technique, but their method does not handle textures with boundaries. To make our system responsive, we combine two strategies that help mitigate the expense of the texture synthesis computation.

Accelerating search for example pixels

The performance of the synthesis algorithm is dominated by the running time of the inner loop, which involves searching for the best (i.e., lowest energy) example pixel to copy into the target region. When the target neighborhood is entirely within the texture region, we accelerate the search using a technique from Image Analogies. In particular, we represent the colours of neighboring pixels as a feature vector and use *approximate-nearest-neighbor search* (ANN) [AMN*98] to find the best match in the example texture. The system also searches in luminance space (or luminance plus alpha) instead of RGBA space to reduce the dimensionality of the problem. The user also has the option of including alpha as an additional search channel.

Unfortunately, ANN cannot be used when the target neighborhood has one or more non-texture-region pixels because the specific pixels to include in the feature vector vary depending on the neighborhood shape. To avoid performing a brute-force search over all boundary neighborhoods in the example texture, we have developed a simple acceleration strategy. The system first counts the number of texture-region pixels in the target neighborhood. Then, the set of example neighborhoods with a similar number of texture-region pixels is found. If there are neighborhoods with an equal number of texture-region pixels, this set is composed of these neighborhoods. Otherwise, the set contains all of the neighborhoods with the next smallest and next largest number of texture-region pixels. Typically, there are only a few dozen neighborhoods in this set. Finally, the system compares each of these neighborhoods against the target to find the best match. To find the candidate neighborhoods efficiently, the system stores the example boundary neighborhoods in a map indexed by the number of texture-region pixels.

The rationale for this scheme comes from the following three observations:

1. Neighborhoods with similar shapes have a similar number of texture-region pixels.
2. If e_{max} is large, only neighborhoods that have similar shapes will be good matches.
3. Neighborhoods at a similar distance from a texture boundary have a similar number of texture-region pixels, even if the neighborhoods have very different shapes.

Based on the first two observations, our search strategy considers only neighborhoods that have the potential to be good matches. Thus, if a good match exists in the example texture, it will be found. If a good match does not exist, it follows from the third observation that our search has a good chance of choosing an example neighborhood that is at a similar distance from the texture boundary as the target neighborhood.

Progressive synthesis

Even with the aforementioned acceleration techniques, the synthesis is too slow to enable a responsive painting interaction. Thus, we adopt the following incremental synthesis strategy. As the user paints, the system generates a low-resolution preview of the painted texture by synthesizing pixels at the coarsest level of the image pyramid and then immediately splatting them onto the screen. A separate thread progressively refines the preview at higher resolutions until the full resolution result has been generated. Since the coarse-level synthesis is typically very fast, preview colours can usually be computed as quickly as the user paints. However, if the system is unable to synthesize all the preview pixels before the canvas is refreshed, the average texture colour

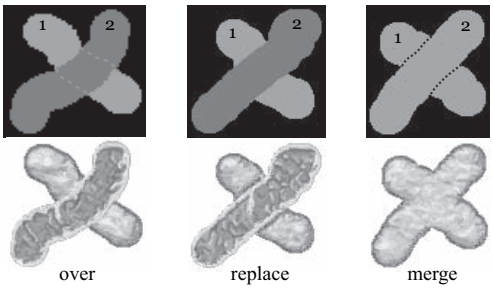


Figure 4: *Layering modes. Top row shows false-coloured strokes, and bottom row shows corresponding synthesized textures. In each case, stroke 1 is painted before stroke 2.*

is copied into the unsynthesized pixels to give the user immediate feedback.

To further improve our algorithm's performance, we borrow a technique from texture-by-numbers that reduces the number of expensive data searches performed during synthesis. The algorithm executes both data and coherence searches for one out of every k pixels. For the other pixels, only a coherence search is performed. Since the coherence search is much faster, this strategy improves performance at the cost of synthesis quality. In our system, the user can adjust the value of k to control this trade-off.

7. Combining strokes

During a typical painting session, the user will apply many strokes of texture paint to the canvas. As each individual stroke is drawn, the system generates boundary effects around the stroke perimeter in the manner described above. However, when two strokes overlap, it is not immediately clear where boundary effects should be synthesized. Depending on the situation, it may make sense to generate boundary effects on zero, one, or both sides of an interface between overlapping strokes.

To give the user control over where boundary effects appear, our system allows strokes to be assigned to different depth layers; within this layered painting model, the user can combine strokes via three distinct layering modes – OVER, REPLACE, and MERGE – each of which results in different boundary synthesis behaviour (see Figure 4).

Over. This layering mode is applied when the user paints over a stroke that resides on an underlying layer. The system synthesizes the newly painted stroke with boundary effects while leaving the bottom stroke unchanged.

Replace. This mode is employed when the user paints with a different texture over a stroke on the same layer, thereby replacing a portion of the existing texture. The system synthesizes boundary effects for both textures along the interface between them. Our system also supports an "eraser brush" that allows the user to remove (rather than replace) texture, thereby exposing the underlying paint. Boundaries are synthesized within the perimeter of the erased region.

Merge. This mode is applied when the user paints over a stroke on the same layer with the same texture. The system merges the strokes to form a larger region of contiguous texture and generates boundary effects around the perimeter of the merged region.

8. Results

To demonstrate the different types of boundary effects that our system can reproduce, we present a few texture paintings that we generated using our interactive painting application. For all of the synthesized textures shown in our results, we used a coherence factor κ between 0 and 5 and a neighborhood radius between 1 and 4.

To produce the watercolour paintings shown in Figure 5(a), we scanned in five watercolour strokes and created the segmented 155×322 input image shown on the left side of the figure. Using this palette, we painted two different trees by layering the textures in a series of washes. Notice that the system is able to reproduce the feathered borders of the top two strokes, as well as the edge-darkening effects exhibited by the other examples in the palette.

The paper collage shown in Figure 5(b) was generated using a palette of different paper textures with torn edges. Each of the three scanned input textures is approximately 200×200. The synthesized result successfully captures the appearance of the torn paper boundaries. Also, by including alpha information with the inputs, we were able to generate semi-transparent paper textures. In particular, notice how underlying colours are visible through the paper edges.

To produce the orange texture result shown in Figure 5(c), we provided the system with two small input textures: a 175×131 image of a piece of orange peel, and a 125×64 image of the orange pulp texture. Note that our system was able to capture the variations in width of the white fibrous material at the edge of the orange peel. Since the pulp texture does not include any boundary effects, none were synthesized in the final result.

To evaluate our algorithm against the Image Analogies texture-by-numbers technique, we performed the following comparison. In our system, we painted an image comprised of five textures taken from various medical illustrations. We then recreated the image using texture-by-numbers by passing in a false-coloured label image, the corresponding medical illustration textures, and the label map associated with our texture painting. The inputs and results generated by our system and texture-by-numbers are all shown in Figure 6.

As illustrated in Figure 6(b), texture-by-numbers does not consistently reproduce the appropriate boundaries at interfaces that do not exist in the example texture, such as the one between the "muscle" texture (light blue label) and the "fat" texture (green label). Since texture-by-numbers considers the actual label colours during synthesis, specific configurations of labels (combined with the propagation of coherent

patches) can result in some boundary pixels being correctly synthesized. For instance, Figure 6(b) exhibits boundary effects along portions of the interface between bone and fat. However, the borders are not reliably synthesized. In contrast, the result generated by our system exhibits the correct boundary effects at all interfaces, as shown in Figure 6(c).

9. Conclusions

In this paper, we have described a system that enables painting with texture. In particular, we have identified the importance of texture boundaries and presented a synthesis algorithm for generating textures with boundary effects at interactive rates. As part of this algorithm, we have described a novel energy function that represents the primary technical innovation of this work. Finally, we have defined a few useful ways in which strokes of texture can interact and described an intuitive painting model that allows the user to induce these interactions.

While our system works well for a broad class of textures, we plan to extend it to handle more structured boundary effects. Currently, our system has difficulty synthesizing highly oriented or anisotropic textures as well as texture regions with wide boundary effects. We believe the addition of some "control channels" (as suggested in Image Analogies) might help solve this problem. Additionally, it might be possible to relieve the user of having to specify texture regions manually by incorporating segmentation and matting techniques into our system. We would also like to improve the responsiveness of our system by investigating additional acceleration techniques such as hybrid patch/pixel-based approaches.

References

[AMN*98] ARYA S., MOUNT D. M., NETANYAHU N. S., SILVERMAN R., WU A. Y.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM 45*, 6 (1998), 891–923.

[Ash01] ASHIKHMIN M.: Synthesizing natural textures. In *I3D* (2001), pp. 217–226.

[CAS*97] CURTIS C. J., ANDERSON S. E., SEIMS J. E., FLEISCHER K. W., SALESIN D. H.: Computer-generated watercolor. *Computer Graphics 31*, Annual Conference Series (1997).

[CT05] CHU N. S.-H., TAI C.-L.: Moxi: Real-time ink dispersion in absorbent paper. In *SIGGRAPH* (2005), ACM Press.

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. In *SIGGRAPH* (2001), ACM Press, pp. 341–346.

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. In *ICCV (2)* (1999), pp. 1033–1038.

[HJO*01] HERTZMANN A., JACOBS C. E., OLIVER N., CURLESS B., SALESIN D. H.: Image analogies. In *SIGGRAPH* (2001), ACM Press, pp. 327–340.

[KSE*03] KWATRA V., SCHöDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph. 22*, 3 (2003), 277–286.

[LH05] LEFEBVRE S., HOPPE H.: Parallel controllable texture synthesis. In *SIGGRAPH* (2005), ACM Press.

[Sut63] SUTHERLAND I. E.: Sketchpad: A Man-Machine Graphical Communication System. In *AFIPS Conference Proceedings* (1963), vol. 23, Spartan Books Inc., pp. 329–346.

[WL02] WEI L.-Y., LEVOY M.: *Order-independent texture synthesis*. Tech. Rep. TR-2002-01, April 2002.

[WS94] WINKENBACH G., SALESIN D. H.: Computer-generated pen-and-ink illustration. In *SIGGRAPH* (1994), ACM Press, pp. 91–100.

[WY04] WU Q., YU Y.: Feature matching and deformation for texture synthesis. *ACM Trans. Graph. 23*, 3 (2004), 364–367.

[ZG02] ZELINKA S., GARLAND M.: Towards real-time texture synthesis with the jump map, 2002.