Integrating Procedural Textures with Replicated Image Editing

Stephen Brooks*
Faculty of Computer Science
Dalhousie University
Halifax, Canada

Neil A. Dodgson†
Computer Laboratory
University of Cambridge
Cambridge, England

Abstract

Image editing software is often characterized by a seemingly endless array of toolbars, filters, transformations and layers. But recently, a counter trend has emerged in the field of image editing which aims to reduce the user's workload through semi-automation. This alternate style of interaction has been made possible through advances in directed texture synthesis and computer vision. And it is in this context that we have developed our texture editing system that allows complex operations to be performed on images with minimal user interaction. This is achieved by utilizing the inherent self-similarity of image textures to replicate intended manipulations globally. In this paper, we expand the capabilities of replicated image editing by integrating procedural texture generation.

CR Categories: I.3.8 [Computer Graphics]: Applications; I.4.9 [Image Processing and Computer Vision]: Applications; I.3.6 [Computer Graphics]: Methodology and Techniques – Interaction Techniques.

Keywords: interactive image editing, texture synthesis, input amplification.

1 Introduction

Image editing remains a complex user-directed task, often requiring proficiency in design, colour spaces, computer interaction and file management. Furthermore, the demands of this skill set are often exacerbated by an equally complex collection of image manipulation commands. In general, this approach has met with considerable success; however, the complexity of these editing tools requires that the user possess a correspondingly high level of expertise.

Recent research in computer graphics has attempted to semi-automate the process of constructing and editing digital images. Far from offering a massive array of image manipulation controls, these semi-automated systems offer interaction at a higher semantic level, consequently minimizing the amount of user interaction.

Our system is a realization of this approach wherein the user is able to minimally specify alterations to a digital texture image, whilst relying on the system to perform repetitive, time-consuming tasks. Our system is a visual analogue to text string search and replace in that a single editing operation at a given location causes global changes: the same operation is performed on all similar areas of the texture image. Consequently, the style of interaction lies between automation and complete user manipulation.

The paper's structure begins with a synopsis of related work and of our previous work on replicated image editing. This is followed by a discussion of the procedural texture extension to the editing system. We conclude with a commentary on limitations and future directions.

2 Related Work

We begin our overview of related work with constraint-based graphics [Sutherland 1963], where the user places constraints on the output of a graphical system. Another system which manipulates vector based images is the search and replace method of Kurlander and Bier [1998]. Conceptually, this system is most similar to our own. However, both systems differ significantly from ours as they operate on vector images unlike our raster image editing tools.

The interactive evolution of textures using genetic algorithms also lies between manual and automatic design methodologies [Sims 1993]. Based on a Darwinian metaphor, the computer's primary role is to present candidate graphics to the user from the design space. Alternatively, the goal of example-based texture synthesis is to generate another texture image that appears to be from the same source as a given input texture [Ashikhmin 2001; Heeger and Bergen 1995; Wei and Levoy 2000]. And recently, a new class of image editing tool has emerged which employs this form of texture synthesis to perform sophisticated image editing operations including Texture-By-Numbers [Barrett and Cheney 2002; Harrison 2001; Hertzmann et al. 2001]. Other tools use texture synthesis to remove entire objects from scenes [Igehy and Pereira 1997]. Yet another fruitful source of user assistance in image editing has come from advances in the computer vision community. Examples of which are intelligent image selection [Mortensen and Barrett 1995] and snapping [Gleicher 1995] tools.

Perhaps the most extreme form of automation that still permits some degree of user input is the image stylization system of DeCarlo and Santella [2002], which using eye-tracking to assign priority to details for a non-photorealistic rendering of the same image. Another type of application that requires minimal interaction are design gallery interfaces. In this approach, the user makes aesthetic judgments over design alternatives that are pre-computed prior to interaction [Marks et al. 1997].

* sbrooks@cs.dal.ca
† nad@cl.cam.ac.uk

3 Replicated Image Editing

Our system replicates editing operations globally over a texture image [Brooks and Dodgson 2002; Brooks et al. 2003]. Changes made to a particular pixel by the user are made to affect all pixels that exhibit similar local neighbourhoods to that selected pixel, thereby relieving the user of the manual effort of repetition. This allows the following concise texture editing operations:

1. Replicated Painting: altering the colour of similar pixels.
2. Replicated Cloning: cloning of another image or texture onto the texture being altered.
3. Replicated Warping: locally contracting or expanding certain regions of the texture, based on similarity to the current selected pixel.

Painting and cloning are similar operations which paint colours onto the image being edited. In Figure 1 we have a simple case of painting a solid red colour onto each pixel whose neighbourhood is sufficiently similar to the pixel selected by the user. The reader will note the directional control of the tool. By this we refer to the ability to affect a particular side of all of the texture elements at once. Figure 2 shows an example of the replicated cloning of a moss texture (left) onto pixels in the bark texture (right). Moss is cloned onto all pixels in the bark texture that are similar to the user selected pixel. Moving from replicated painting to replicated cloning requires positioning the cloning texture and using the corresponding colour values from the cloned texture instead of a solid colour value over the whole image.

Replicated warping is distinct from the other tools in that it does not affect pixel colour; it instead modifies the shape of image regions under the user's guidance. Those pixels whose local neighbourhoods are within a certain threshold of similarity to the user selected point are expanded locally. Since the overall area remains the same, some regions are compressed while others are expanded. Figure 3 shows the application of replicated warping to an image of leaves. The left image has been altered so that the spaces between the leaves have been expanded, shrivelling the leaves themselves. The right image shows the opposite effect with the leaves expanded almost to the exclusion of the spaces in between.

In order to determine which pixels in the image are sufficiently similar to the pixel selected by the user, the local circular neighbourhood of the chosen selection point is compared against that of every other pixel's neighbourhood in the same image. For replicated painting, the current painting colour is then applied to the selected pixel but also to a subset of all pixels in the image: those that have local neighbourhoods whose difference from the selected pixel are within a certain threshold. The selection point receives full paint opacity, as do all pixels with neighbourhoods identical to it. The distance threshold is set by the user and defines the maximum distance value beyond which the opacity of the applied paint is zero. Between zero distance and the distance threshold the opacity is scaled linearly, meaning that the more similar a pixel is deemed to be to the selected pixel, the greater is the applied opacity.

The original formulation [Brooks and Dodgson 2002] of the distance measure between the two points, p_1 and p_2, using Gaussian neighbourhoods is the L2 norm:

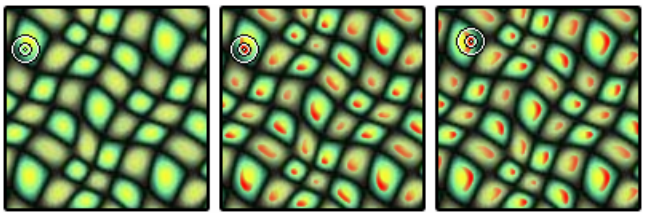

Figure 1: A simple case of replicated painting.

Figure 2: Left: cloning image. Right: moss cloned onto bark.

Figure 3: Replicated warping. Leaves narrowed and expanded.

$$d(p_1,p_2)=\sum_{l=1}^{L}\sum_{j=-k/2}^{k/2}\sum_{i=-k/2}^{k/2}\left(G_l(x_1+i,y_1+j)-G_l(x_2+i,y_2+j)\right)^2$$

where G_l is level l of the Gaussian pyramid, L is the number of Gaussian levels used, k is the window size, $p_1 = (x_1,y_1)$ is the centre of the neighbourhood around p_1 and $p_2 = (x_2,y_2)$ is the centre of the neighbourhood around p_2.

This was subsequently augmented with local wavelet responses, giving the user additional control over the sharpness of the painting tools. With the wavelet responses added, the distance metric becomes:

$$d(p_1,p_2)=\begin{aligned}&\alpha\left(\sum_{l=1}^{L}\sum_{j=-k/2}^{k/2}\sum_{i=-k/2}^{k/2}\left(G_l(x_1+i,y_1+j)-G_l(x_2+i,y_2+j)\right)^2\right)+\\&(1-\alpha)\left(\sum_{l=1}^{L}\sum_{\theta=0}^{2\pi}\left(W_{l,\theta}(x_1/2^{l-1},y_1/2^{l-1})-W_{l,\theta}(x_2/2^{l-1},y_2/2^{l-1})\right)^2\right)\end{aligned}$$

where α is a sharpness weighting value controlled by the user's slider, $W_{l,\theta}$ is orientation θ of level l of the wavelet pyramid and L is the number of wavelet levels used.

Figure 4: Procedurally generated moss texture is cloned into the brick image. The similarity-image is shown to the left.

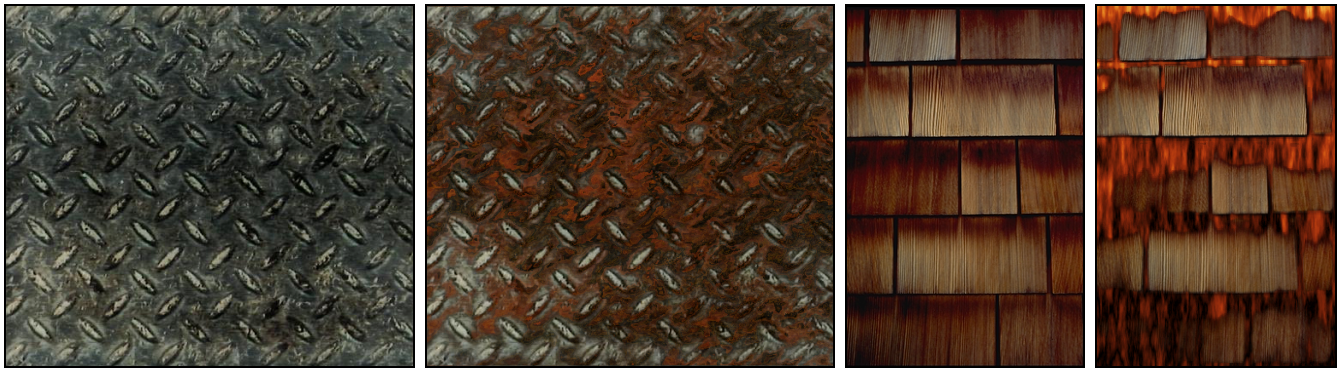

Figure 5: The left pair of images shows a procedurally generated rust texture cloned onto a metal image. The right pair of images shows a procedurally generated fire texture cloned onto a shingle image.

4 Integrating Procedural Textures

In this paper we expand the capabilities of replicated image editing by integrating procedurally generated textures into the cloning tool. Rather than cloning colour content from a second image, we use the level of similarity of a given pixel (to the user selected pixel) as an input parameter for procedural textures.

When the user selected pixel is compared with all other pixels in the same image, a "similarity-map" is generated. This can be visualized with whiter colours for highly-similar pixels and blacker colours for pixels that are not similar. Such a similarity-map is shown to the left in Figure 4. In this example the user has selected a location on the underside of a brick. Once the similarity map, $s(x, y)$, is computed, the values can be directly input into a procedural texture.

All of our procedural textures incorporate fractal noise which introduces a certain degree of natural randomness [Ebert et al. 1994]. A 2D fractal noise function, $f(x, y)$, can be briefly defined as follows:

$$f(x,y) = \sum_{i=0}^{N} \left(noise\left(x \times 2^i, y \times 2^i \right) / 2^i \right)$$

where i is the current octave, N is the number of octaves, and *noise* is a function that smoothly interpolates a grid of random values with cosine or cubic interpolation.

With a fractal noise function in hand, we can now define a number of procedural textures which take the similarity value, $s(x, y)$, at pixel $m(x, y)$ along with the original x and y positional values as input. The first is a moss texture shown in Figure 4 that combines uses the similarity value to control the frequency of the texture. Moss is defined in the following pseudo-code:

```
function moss(x, y, s(x, y)) returns color {

    // low-frequency green for the basic moss appearance
    amount = abs(sin(f ( 5 × x × s(x, y),  5 × y × s(x, y))));
    color  = mixColor(green, black, amount);

    // add small amount of mid-frequency orange
    amount = abs(0.2 × sin(f (25 × x × s(x, y), 25 × y × s(x, y))));
    color = mixColor(orange, color, amount);

    // add high-frequency yellow speckling
    amount = abs(0.8 × sin(f (50 × x × s(x, y), 50 × y × s(x, y))));
    color = mixColor(yellow, color, amount);
}
```

where mixColor(colorA, colorB, amount) returns *amount* of colorA and (1 – *amount*) of colorB. Without the use of the similarity levels, the moss texture would lack visual structure.

The next procedural texture, rust, is similar in structure to moss and is shown in Figure 5. Our rust texture is defined in the following pseudo-code:

```
function rust(x, y, s(x, y)) returns color {

    // low-frequency red for the basic rust appearance
    amount = abs(sin(f ( 5 × x × s(x, y),  5 × y × s(x, y))));
```

```
    color   = mixColor(red, black, amount);

    // add high-frequency orange speckling
    amount = abs(0.5 × sin(f (50 × x × s(x, y), 50 × y × s(x, y))));
    color = mixColor(orange, color, amount);
}
```

The final texture, fire, also shown in Figure 5 is a smoother texture and is compressed in the horizontal direction:

```
function fire(x, y, s(x, y)) returns color {

    // similarity level controls amount of fire
    amount = (s(x, y) × abs(sin(f (20 × x, 6 × y))));
    color   = mixColor(red, black, amount);

    // power of 10 used to narrow yellow areas
    amount = (s(x, y) × abs(sin(f (20 × x, 6 × y)))) ^10;
    color   = mixColor(yellow, color, amount);

    // higher power of 20 used to narrow brightest areas
    amount = (s(x, y) × abs(sin(f (20 × x, 6 × y)))) ^20;
    color = mixColor(white, color, amount);
}
```

Once the procedural textures are computed we directly apply them to the original image, using the similarity level as a weighting value between the original texture colour $t(x, y)$ and the newly generated colour $m(x, y)$. The final colour, $c(x, y)$ is computed as:

$$c(x,y) = (1-s) \times t(x,y) + s \times m(x,y)$$

These procedural textures that have been defined by no means exhaust the possibilities but do illustrate the usefulness of integrating replicated editing with procedural textures.

5 Conclusion and Future Directions

Our system amplifies the user's input by replicating painting, cloning and warping operations over a texture. In this paper we have made replicated editing more flexible and efficient by incorporating procedural texturing into the replicated cloning operation.

An advantage over the prior work on replicated cloning is that we no longer require the user to locate (and possibly mask) an appropriate cloning texture. More importantly, procedural textures are controllable and can now be computed in real-time as hardware accelerated pixel shaders. And, while designing textures does require effort, procedural textures are sufficiently mature and widespread that existing sets of procedural textures can be employed.

Although these replicated editing techniques are not generally suitable for non-texture images, we believe that this can be partly overcome by combining our system with a system of object segmentation. An initial stage would segment an image into separate areas of uniform texture. Once segmented, the effects of our replicated editing system would then be constrained to operate only within the current selected area.

Replicated editing might also be extended to geometric and texture editing operations on a 3D object based on the similarity of local surface curvature instead of, or in concert with, texture similarity. It would need to be determined if the user interface techniques which work for 2D will work equally well for the 3D analogue.

References

ASHIKHMIN, M. 2001. Synthesizing natural textures, in *ACM Symposium on Interactive 3D Graphics*, 217–226.

BARRETT, W. AND CHENEY, A. 2002. Object-Based Image Editing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), 777-784.

BROOKS, S. AND DODGSON, N. A. 2002. Self-Similarity Based Texture Editing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), 653-656.

BROOKS, S., CARDLE, M. AND DODGSON, N. A. 2003. Enhanced Texture Editing using Self-Similarity. *Vision, Video and Graphics*, Bath, 231-238.

DECARLO, D. AND SANTELLA, A. 2002. Stylization and Abstraction of Photographs. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), 769-776.

EBERT, D., MUSGRAVE, F., PEACHEY, D., PERLIN, K. AND WORLEY, S. 1994. *Texturing and Modeling: A Procedural Approach.* AP Professional, Cambridge, MA.

GLEICHER, M. Image snapping. 1995. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, 183-190.

HARRISON, P. 2001. A Non-Hierarchical Procedure for Re-Synthesis of Complex Textures. *WSCG'2001*.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-Based Texture Analysis/Synthesis. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, 229-238.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B. AND SALESIN, D. H. 2001. Image analogies. In *Computer Graphics (SIGGRAPH '01 Proceedings)*, 327-340.

IGEHY, H. AND PEREIRA, L. 1997. Image Replacement Through Texture Synthesis. In *International Conference on Image Processing*, volume 3, 186-189.

KURLANDER, D. AND BIER, E. 1988. Graphical search and replace. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, 113-120.

MARKS, J., ANDALMAN, B., BEARDSLEY, P. A., FREEMAN, W., GIBSON, S., HODGINS, J., KANG, T., MIRTICH, B., PSTER, 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 *Computer Graphics (SIGGRAPH '97 Proceedings)*, 389-400.

MORTENSEN, E. AND BARRETT, W. 1995. Intelligent scissors for image composition. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, 191-198.

SIMS, K. 1993. Interactive evolution of equations for procedural models. *The Visual Computer*, 9(8), 466-476.

SUTHERLAND, I. 1963. *Sketchpad--a man-machine graphical communication system*. Technical Report 296, Lincoln Laboratory, Massachusetts Institute of Technology.

WEI, L. AND LEVOY, M. 2000. Fast Texture Synthesis using Tree-Structured Vector Quantization, In *Computer Graphics (SIGGRAPH '00 Proceedings)*, 479-488.