Replicated Texture Editing

Stephen Brooks
Marc Cardle
Neil A. Dodgson

Technical Report CS-2005-21

November 15, 2005

Faculty of Computer Science
6050 University Ave., Halifax, Nova Scotia, B3H 1W5, Canada

Replicated Texture Editing

Stephen Brooks
Dalhousie University

Marc Cardle
University of Cambridge

Neil A. Dodgson
University of Cambridge

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. 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 report, we summarize the capabilities of our image editing approach including operations for replicated painting, cloning and warping. In particular, we detail recent enhancements including user-controlled sharpness, Boolean similarity expressions and the adaptive synthesis of cloning textures.

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

1 Introduction

The most broadly applied approach to modelling the complexity of the natural world is to provide the scene designer with sophisticated tools that permit a high degree of control over geometric surfaces and their corresponding textures. This approach has enjoyed considerable success, yet the sophistication of the editing tools requires a comparable level of sophistication from the user. Often, the user must be a highly skilled artist as well as having considerable technical training and experience with computers. These prerequisites are beyond many users.

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-automatic systems offer interaction at a higher semantic level, consequently minimizing the amount of user interaction.

Our image editing framework 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 report's structure begins with a synopsis of related work and is followed by an introduction to the basics of replicated painting and cloning. The discussion then proceeds to a number of improvements to our image editing framework including user-controlled sharpness, Boolean similarity expressions and the adaptive synthesis of cloning textures. Replicated texture warping is then discussed along with super-resolution synthesis. 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], in which the user places constraints on graphical arrangements. 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 of these 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 new texture images that appear to be from the same source as a given input texture [Ashikhmin 2001; Heeger and Bergen 1995; Kwatra *et al.* 2003; Kwatra *et al* 2005; Lefebvre and Hoppe 2005; Wei and Levoy 2000].

And recently, a new class of image editing tools 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] or replace textures [Fang and Hart 2004; Liu *et al.* 2004; Zelinka *et al.* 2005]. And as we are introducing a modified texture cloning tool there are other forms of texture mixing [Bar-Joseph 2001; Matusik *et al.* 2005] and image compositing [Burt and Adelson 1983; Porter and Duff 1984] that deserve mention.

Other semi-automated texture creation systems include Live Paint [Perlin and Velho 1995], which uses the concept of a multi-resolution painting system [Berman *et al.* 1994] to combine procedural textures [Ebert *et al.* 1994]. Dischler *et al.* [1999] describe a unique hybrid approach that combines texture analysis and geometric modeling. Lewis' [1984] early paper presents an interactive procedure for generating textures in the frequency domain.

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. Elder and Goldberg [1998] also offer a novel editing system that operates in an invertible contour domain.

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 uses 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].

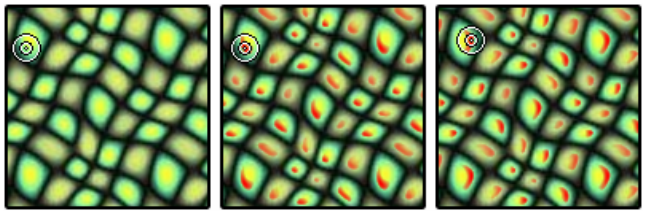

Figure 1: A simple case of replicated painting.

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

Figure 3: Replicated warping with leaves narrowed and expanded.

3 Replicated Image Editing

Our system replicates editing operations globally over a texture image [Brooks and Dodgson 2002; Brooks et al. 2003; Brooks and Dodgson 2005]. 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 (Figure 1).
2. Replicated Cloning: cloning of a second texture onto the texture being altered (Figure 2).
3. Replicated Warping: locally contracting or expanding certain regions of the texture, based on the similarity to the current selected pixel (Figure 3).

3.1 Replicated Painting and Cloning

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.

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, T.

The selection point receives full paint opacity, as do all pixels with neighbourhoods identical to it. The distance threshold, T, 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 paint opacity.

Our distance measure is computed as the weighted sum of squared differences (L2 norm) between each corresponding pixel in circular multi-resolution neighbourhoods surrounding to points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$:

$$d(p_1, p_2) = \sum_{l=0}^{L-1} \sum_{(i,j)\in C} 4^l \times M(2^l i, 2^l j) \times \left(G_l\left(\lfloor x_1/2^l \rfloor + i, \lfloor y_1/2^l \rfloor + j\right) - G_l\left(\lfloor x_2/2^l \rfloor + i, \lfloor y_2/2^l \rfloor + j\right) \right)^2$$

where:

$C = \left\{ (i,j) \in \mathrm{N}^2 \;\middle|\; 0 \le \sqrt{i^2 + j^2} \le r \right\}$
G_l is level l of the Gaussian pyramid,
L is the number of Gaussian levels used,
M is a 2D Gaussian weighting function and
r is the neighbourhood radius.

The use of multi-resolution neighbourhoods in the distance calculation allows us to incorporate a wider area surrounding each pixel at a lower computational cost. This is under the assumption that distant pixels are less important to the similarity calculation and can therefore be approximated more and more coarsely. We then define the similarity level s at pixel $p = (x, y)$ to be:

$$s(p, p') = \max\left((T - d(p, p'))/T, 0 \right)$$

where p' is the user selected point and T is the user-selected threshold. This similarity level is used directly as the opacity of the applied paint or cloned texture.

We note that all of the above calculations are performed in the CIE LAB colour space since Euclidean distances in RGB space do not correspond to colour differences as perceived by human beings [Jackson *et al.* 1994]. When computing the distance measure between two pixels, each pixel is characterized as a vector composed of a concatenation of all LAB values in each of the levels. To increase interactivity, we employ principal components analysis (PCA) to reduce the dimensionality of the concatenated neighbourhood vectors [Jollife 1986]. Using

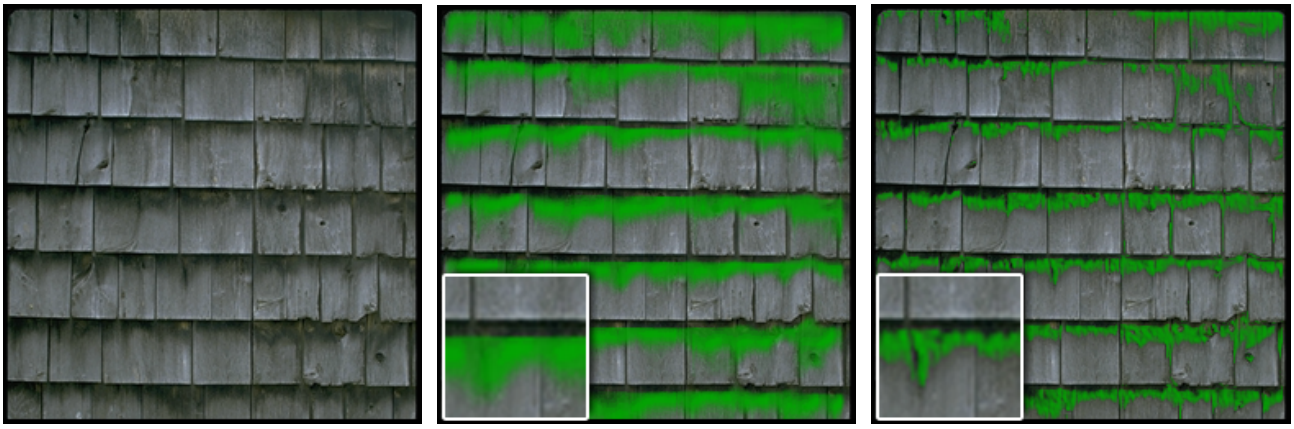

Figure 4: Painting of solid green colour onto textures. Left: original. Centre: Painted using only Gaussian pyramid neighbourhood values. Right: Both neighbourhood and wavelet responses used.

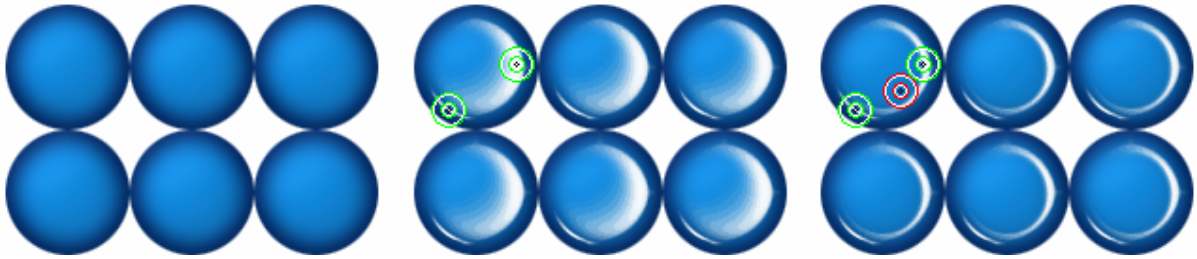

Figure 5: Example use of a multi-point Boolean similarity expression. The green rings denote positively weighted similarity points, and the red is negative. Left: original repeating blue-dot texture. Middle: two positively weighted similarity points used to paint white onto similar pixels. Right: The expression is asking the system to "paint white those pixels that are similar to the green points but dissimilar to the red".

Figure 6: Painting of snow (white paint) applied to a doorway using three similarity points. Middle: Painted using only Gaussian pyramid neighbourhood responses. Right: Both Gaussian and wavelet responses used.

principal component analysis increases the efficiency of computing the distance metric without a significant loss of fidelity. Due to the considerable coherence within the neighbourhood vectors we generally gain an order of magnitude reduction in the size of the vectors without an appreciable reduction of quality. This allows us to achieve interactive rates.

3.2 Painting Sharpness

A limitation of the distance measure, $d(p_1, p_2)$, is that it does not work as well for textures that contain a high degree of randomness or sharp features. This is due to the smoothing tendency of Gaussian-pyramid neighbourhood metrics. To address the limitations of the original distance measure, we have integrated wavelet responses into our distance metric. Moreover, we give the user even finer control by providing a 'sharpness' slider that specifies what proportion of Gaussian neighbourhood versus wavelet responses are to be used in the distance calculation.

To improve the distance metric we include multi-scale responses from a steerable pyramid transform of the image being edited [Simoncelli *et al.* 1992]. Like the Gaussian pyramid, this transform decomposes the image into several spatial frequency bands. It also further divides each frequency band into a set of orientation bands which respond to rotationally varying edges. With the wavelet responses added, the distance metric becomes:

$$d_G(p_1, p_2, l) = \sum_{(i,j)\in C} 4^l \times M(2^l i, 2^l j) \times \left(\begin{array}{l} G_l(\lfloor x_1/2^l \rfloor + i, \lfloor y_1/2^l \rfloor + j) - \\ G_l(\lfloor x_2/2^l \rfloor + i, \lfloor y_2/2^l \rfloor + j) \end{array} \right)^2$$

$$d_W(p_1, p_2, l) = \sum_{\theta=0}^{3} \left(\begin{array}{l} W_{l,\theta}(\lfloor x_1/2^l \rfloor, \lfloor y_1/2^l \rfloor) - \\ W_{l,\theta}(\lfloor x_2/2^l \rfloor, \lfloor y_2/2^l \rfloor) \end{array} \right)^2$$

$$d(p_1, p_2) = \sum_{l=0}^{L-1} (1-\beta) \times d_G(p_1, p_2, l) + \beta \times (d_W(p_1, p_2, l))$$

where:

$d_G(p_1, p_2, l)$	is the Gaussian pyramid component,
$d_W(p_1, p_2, l)$	is the steerable wavelet component,
$\beta \in [0..1]$	is controlled by the user's sharpness slider,
$W_{l,\theta}$	is orientation θ, level l of the wavelet pyramid,
L	is the number of pyramid levels used.

Since the wavelet transform responds strongly to edges at varying orientations, by placing more emphasis on wavelet responses the user can thereby cause the self-similarity tool to react more strongly to sharp features in the texture during editing and avoid the problem of excessive smoothing that can result from relying solely upon Gaussian pyramid neighbourhoods.

Figure 4 shows the painting of a solid green colour onto a wood shingle texture. The original image is shown to the left. The centre image shows a result using only Gaussian pyramid neighbourhood values and the image to the right shows a result using both neighbourhood and wavelet responses. As can be seen from the zoomed inset images, by incorporating the wavelet responses into the similarity distance metric, the self-similarity tool is able to respond to the finer edge details in the original images. The user can therefore dictate the extent to which these edge details influence the final outcome. An additional example is shown in Figure 6.

3.3 Boolean Similarity Expressions

We now describe an extension to our editing framework that allows the user to select multiple similarity points within the texture which together comprise a Boolean similarity expression. In Figure 5, we see an example of using multiple similarity points. To the left is shown a repeating blue-dot texture which has been constructed for the purpose of illustration. In the centre the same blue texture is shown painted white using two positive (green) similarity points. On the right the Boolean similarity expression now includes two positive similarity points and one negative (red). Note how the negative similarity point restricts the application of the white paint. In this way, the user can specify that pixels must be like pixel *A* or pixel *B* but not pixel *C*.

To compute the opacity level when using a Boolean similarity expression, we simply sum the combined opacity level from the positive similarity points and subtract the opacity of the negative ones. The final value is then clamped to be within the range of 0.0 and 1.0. The formula for the opacity of any pixel, *p*, given two user-selected sets of positive, *A*, and negative, *B*, similarity points is then:

$$opacity(p, A, B) = \min\left(\max\left(\sum_{i=1}^{n} s(p, A_i) - \sum_{j=1}^{m} s(p, B_j),\ 0 \right),\ 1 \right)$$

where:

- n is the number of positive similarity points and
- m is the number of negative similarity points.

An example of painting snow (white paint) onto areas of a doorway using multiple similarity points is shown in Figure 6.

3.4 Adaptive Generation of Cloning Textures

There exist in nature multi-layered textures in which the structure and position of the secondary texture(s) is dependant on the primary texture. An example of this is shown in Figure 8 in which the spatial structure of the secondary fire texture is dependant on the underlying wood texture. However, as presented thus far, replicated cloning does not have the ability to adapt the cloning texture to spatially match the target texture. In Brooks *et al.* [2003], we addressed this issue by developing a semi-automatic process for re-arranging a cloning texture (fire) to better match a target texture (wood) prior to cloning.

This is achieved by semi-automatically constructing Texture-by-Numbers [Hertzmann *et al.* 2001] masks of both the cloning texture and the image being cloned onto. The construction of these masks are themselves based on the self-similarity of the textures. These masks are then used in a guided re-synthesis prior to cloning. Sample results of this two stage solution are shown in Figures 7 and 8 where the cloned texture is first re-arranged using texture-by-numbers re-synthesis. More details of this process are available in the original paper [Brooks *et al.* 2003].

An alternate solution to the problem of matching the cloning texture to the target texture is presented in Brooks and Dodgson [2005]. 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 generating procedural textures.

For replicated painting and cloning, the user selected pixel is compared with all other pixels in the same image. This produces a "similarity-map" which can be visualized with whiter colours for

Figure 7: Snowy leaf texture is re-ordered and cloned onto a rusting ring.

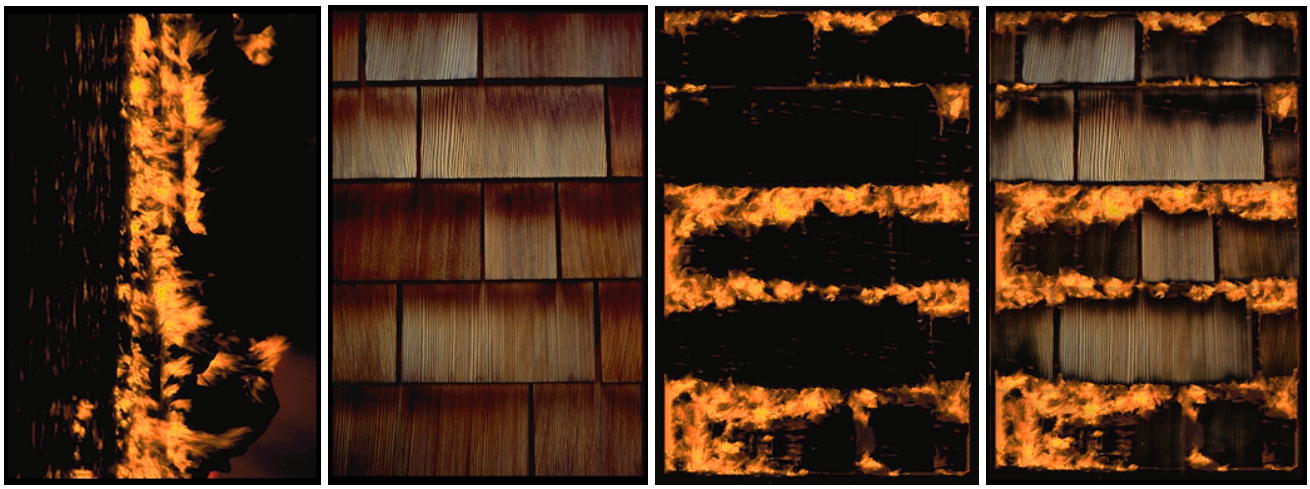

Figure 8: Fire texture is re-ordered and cloned onto wood shingles.

Figure 9: Procedurally generated moss texture is cloned into the brick image. The similarity-image is shown to the left.

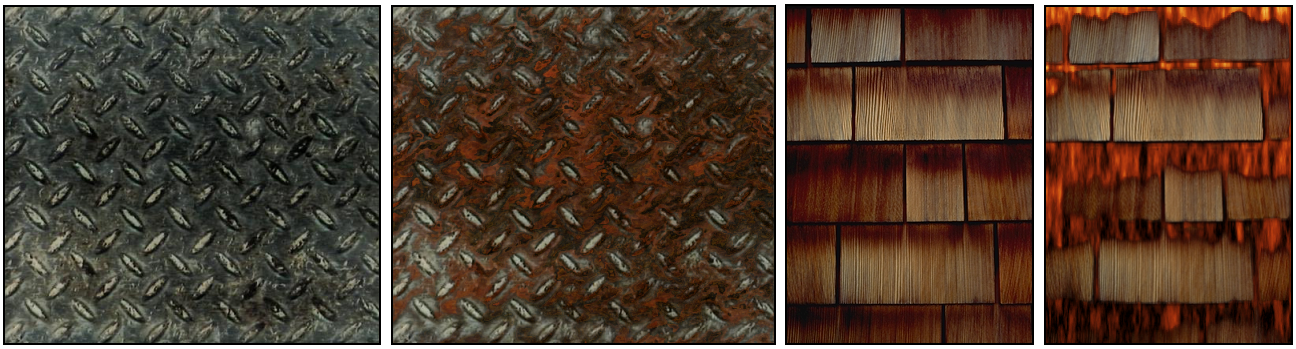

Figure 10: The left pair of images shows a procedurally generated rust texture cloned onto a metal image. The right pair of images shows a procedurally generated fire texture cloned onto a shingle image.

highly-similar pixels and blacker colours for pixels that are not similar. Such a similarity-map is shown to the left in Figure 9. In this example the user has selected a location on the underside of a brick. Once this similarity map, $s(x, y)$, is computed, the values can be directly input into a procedural texture.

Our example 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 p along with the original x and y positional values as input. The first example is a moss texture shown in Figure 9 that 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 10. 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 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:

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

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.

The advantages of using procedurally-based texture cloning are efficiency, flexibility and ease of use. However, it does require that procedural functions can be constructed that realistically simulate the cloning textures. This is not always the case.

3.5 Replicated Warping

Replicated warping is distinct from painting and cloning 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. The question becomes how to convert scalar similarity values, $s(x, y)$, derived from neighbourhood distances into 2D area expansions (Figure 11).

To accomplish this we borrow the interactive image-warping scheme of Keahey *et al.* [1997]. In their notation, the grid of similarity values defines a magnification function, M, from which a 2D grid displacement function, F, must be derived. The magnification function, M, is essentially the derivative of the desired F, and a numerical algorithm is used to approximate the integration of M, yielding an estimate, F_C, at each iteration. The corresponding approximate magnification function, M_C, can be directly computed from F_C, allowing the resultant error, $M_E = M - M_C$, to be calculated. F_C is then further modified on a vertex-by-vertex basis. Effectively, the neighbouring vertices are moved outwards in F_C where $M_E > 0$, and drawn inwards where $M_E < 0$, yielding a better approximation. From this, a 2D transformation is produced that is both symmetric and centred around magnification maxima. The algorithm benefits from optimizations detailed by Keahey and our implementation converges in less than 0.05s.

Self-similarity scalar values are directly used to interactively drive area magnification. If in painting mode the pixel would have received 75% opacity, the local area instead increases by 75%. 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. Further examples of replicated warping are shown in Figure 12.

Depending on the texture and on the amount of expansion, the warped texture can suffer a loss of high frequency detail. We overcome this by re-synthesizing detail into expanded areas, using the newly warped texture as a constraining image for super-resolution synthesis as described in Hertzmann *et al.* [2001]. Close-up images of enhanced details are shown in Figure 13.

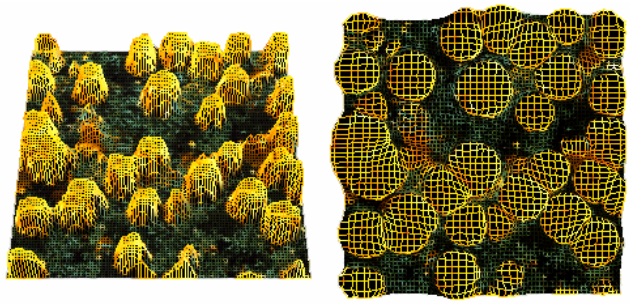

Figure 11: From scalar similarity values (left) to 2D texture warp (right).

Figure 12: Similarity Based Warping: 1st column shows contracted regions, 2nd column contains original textures and 3rd column contains expanded regions. Final results in 4th column have been enhanced with super-resolution synthesis using details from the original textures.

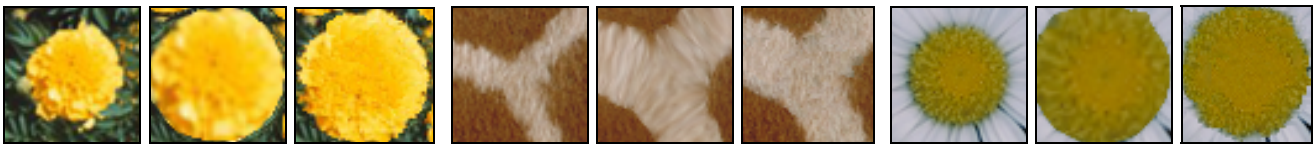

Figure 13: Loss of high frequency details and re-synthesis. Left images are taken from the original textures with the warped versions shown in the middle. Results of super-resolution synthesis are shown to the right.

4 Conclusion and Future Directions

Our unique image editing framework amplifies the user's input by replicating painting, cloning and warping operations over a texture. This framework has benefited from a number of recent improvements including user-controlled sharpness, Boolean similarity expressions and the adaptive synthesis of cloning textures.

Although these replicated editing techniques are not generally suitable for non-texture images, we believe that this might be overcome by combining our system with systems that perform image segmentation and shape-from-shading.

Currently our approach works best for textures which are uniformly lit. Non-uniform lighting leads to poorer results. We believe that this restriction might be addressed by integrating similarity based editing with a photo editing system such as that of Oh *et al.* [2001] which permits texture illumination correction.

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.

BAR-JOSEPH, Z., EL-YANIV, R., LISCHINSKI, D., AND WERMAN, M. 2001. Texture Mixing and Texture Movie Synthesis Using Statistical Learning. *IEEE Transactions on Visualization and Computer Graphics*, 7, 2, 120-135.

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

BERMAN, D., BARTELL, J., AND SALESIN, D. 1994. Multiresolution Painting and Compositing. *ACM SIGGRAPH 94*, 85-90.

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.

BROOKS, S. AND DODGSON, N. A. 2005. Integrating Procedural Textures with Replicated Image Editing. *Proceedings of ACM GRAPHITE*. (to appear)

BURT, P., AND ADELSON, E. 1983. A Multiresolution Spline with Application to Image Mosaics. *ACM Transactions on Graphics*, 2, 4, 217-236.

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

DISCHLER, J., AND GHAZANFARPOUR, D. 1999. Interactive Image-Based Modeling of Macrostructured Textures. *IEEE Computer Graphics and Applications*, 19, 1, 66-74.

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

ELDER, J., AND GOLDBERG, R. 1998. Image Editing In the Contour Domain. *IEEE Computer Vision and Pattern Recognition*, 374-381.

FANG, H. AND HART, J. C. 2004. Textureshop: Texture Synthesis as a Photograph Editing Tool. *ACM Transactions on Graphics,* 23(3), 354-359.

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.

JACKSON, R., MACDONALD, L AND FREEMAN, K. 1994. *Computer Generated Color: A Practical Guide to Presentation and Display*. John Wiley & Sons.

JOLLIFE, I.T. 1986. Principal Component Analysis. *Springer-Verlag*, New York.

KEAHEY, A., AND ROBERTSON, E. 1997. Nonlinear Magnification Fields. *IEEE Symposium on Information Visualization*, 51-58.

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

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.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Transactions on Graphics,* 24(3), 795-802.

LEFEBVRE, S. AND HOPPE, H. 2005. Parallel controllable texture synthesis. *ACM Transactions on Graphics,* 24(3), 777-786.

LEWIS, J. P. 1984. Texture Synthesis for Digital Painting. *Computer Graphics*, 18, 3, 245-252.

LIU, Y., LIN, W., AND HAYS, J. 2004. Near-regular texture analysis and manipulation. *ACM Transactions on Graphics,* 23(3), 368-376.

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.

MATUSIK, W., ZWICKER, M., AND DURAND, F. 2005. Texture design using a simplicial complex of morphable textures. *ACM Transactions on Graphics,* 24(3), 787-794.

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

OH, B., CHEN, M., DORSEY, J., AND DURAND, F. 2001. Image-Based Modeling and Photo Editing. *ACM SIGGRAPH 2001,* 433-442.

PERLIN, K., AND VELHO, L. 1995. Live Paint: Painting With Procedural Multiscale Textures. *ACM SIGGRAPH 95*, 153-160.

PORTER, T., AND DUFF, T. 1984. Compositing Digital Images. *Computer Graphics*, 18, 3, 253-259.

SIMONCELLI, E. P., FREEMAN, W. T., ADELSON, E. H. AND HEEGER, D. J. 1992. Shiftable Multi-Scale Transforms. *IEEE Transactions on Information Theory, Issue on Wavelets 38*, 587–607.

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.

ZELINKA, S., FANG, H., GARLAND, M., AND HART, J. C. 2005. Interactive Material Replacement in Photographs. In *Proceedings of Graphics Interface*, 227-232.